FELIX 2002

Command Language Reference

MARCH 2002

*All updated documentation (**User Guide**, **Tutorials**, and FELIX **Command Language Reference**) for the latest release of FELIX is available at the Accelrys website documentation library:*

http://www.accelrys.com/doc/life/index.html

Accelrys Inc.
9685 Scranton Rd.
San Diego, CA
92121-3752

Tel: 858 799 5000
Fax: 858 799 5100

Copyright (U.S. version)

Restricted Rights Legend

Trademark Acknowledgments

Permission to Reprint, Acknowledgments, and References

Accelrys usually grants permission to republish or reprint material copyrighted by Accelrys, provided that requests are first received in writing and that the required copyright credit line is used. For information published in documentation, the format is “Reprinted with permission from *Document-name*, *Month Year*, Accelrys Inc., San Diego.” For example:

> Reprinted with permission from *FELIX 2002 User Guide*, March 2002, Accelrys Inc., San Diego.

Requests should be submitted to Accelrys Scientific Support, either through electronic mail to **support@accelrys.com** or in writing to:

Accelrys Scientific Support and Customer Service
9685 Scranton Road
San Diego, CA 92121-3752

To print photographs or files of computational results (figures and / or data) obtained using Accelrys software, acknowledge the source in a format similar to this:

> Computational results obtained using software programs from Accelrys Inc.—dynamics calculations were done with the Discover® program, using the CFF91 forcefield, ab initio calculations were done with the DMol program, and graphical displays were printed out from the Cerius^2 molecular modeling system.

To reference a Accelrys publication in another publication, no author should be specified and Accelrys *Inc.* should be considered the publisher. For example:

> *FELIX 2002 User Guide*, March 2002. San Diego: Accelrys Inc., 2000.

Contents

Command Reference 97

Symbol Reference 251

How To Use This Book

FELIX 2002 *FCL Command Language Reference* is a complete guide to the command language that underlies the FELIX program. It discusses the basics of the language, symbols and expressions, macros, real-time displays, menus (and how to customize them), the database, and commands. Appendix C, *Macro Examples* contains some useful macros.

This reference is available only online at Accelrys's website:

http://www.accelrys.com/doc/life/index.html.

> **Note:** If you are prompted for a username and password at the Accelrys documentation website, use the following:
>
> Username: **science**
>
> Password: **faster**

An index and table of contents are provided, and you may use the searching capabilities of your browser to locate information.

You can also use the Accelrys Site Search at:

http://www.accelrys.com/search.html

Here, select **All Documentation** in the **Search Area** list. Enter term(s) in the **Search** field, then click **Search**.

To print individual chapters of this guide, use your browser's printing capability.

> **Note:** Appendix A, *Command Reference*, and Appendix B, *Symbol Reference* are very large files. Before printing them, you may want to be certain you need all the information these sections contain

The online help found at the Accelrys website provide more information about the commands in the menu interface. For information

about using FELIX 2002, please consult the FELIX *User Guide* and FELIX *Tutorials* books.

Who should use this guide

This guide is intended for advanced FELIX users who are interested in modifying the user interface, writing macros, and doing other operations that require the FELIX command language FCL.

What FCL does

Some of the things FCL allows you to do are:

Modify the menu interface

Write your own macros

Modify existing macros

Things to be familiar with

You will probably want to familiarize yourself with a few things before working with FCL:

The windowing software on your workstation.

Use of the mouse on your workstation.

The FELIX 2002 menu interface.

Workstation requirements

Before you begin, be certain that you have these things available on your workstation:

An installed and licensed copy of FELIX 2002.

A directory in which you can create subdirectories and files.

Related books

You can find additional information about general molecular modeling, structure determination, and NMR data analysis in several other books published by Accelrys:

NMRchitect – Describes the theory of NMR data analysis and describes how to use the NMRchitect software to analyze NMR data in the Insight II envirnment.

Insight II – Describes the Insight II molecular modeling program.

System Guide – Provides step-by-step instructions for installing and administering Insight II products in your operating environment.

Typographical conventions

Unless otherwise noted in the text, this guide uses the typographical conventions described below:

Names of most items in the FELIX 2002 interface are presented in **bold** type. For example:

> Select the **Process/ID Data Processing...** menu item.

FELIX command lines are represented in a `courier` font. If the line contains keywords you must type exactly, they are given in **bold courier** font. If a word is something you must replace with a specific value (like a filename), it is indicated in ***italic bold courier font***. For example:

```
> maxtrixname rw control
```

1 Introduction

What is FCL?

The FELIX Command Language (FCL) combines spectroscopic data-processing commands, a computer programming language, and a graphical user interface (GUI) protocol. This blend of tools gives FELIX its flexibility and adaptability as an NMR data-processing environment.

You can issue an FCL command inside the Command Window of FELIX main window to call a specific action. For example, you can enter:

```
>   mat yuin.mat w
```

to open a writable NMR matrix called ***yuin.mat***.

Using a text editor, you can compose a document (called a *macro*) containing a series of FCL commands, save the document as a text file, then execute the macro as needed from the Command Window to complete a certain task. For example, you can enter:

```
>   .yuin
```

in the Command Window to execute a macro called ***yuin.mac***. See Chapter 4 for more details about composing macros.

Virtually the entire GUI layout and every action invoked when you select a menu item or click an icon in FELIX are macro-based. You can customize the FELIX layout and actions by changing the relevant macros. To see which macros are being called during run-time, select the

Preference/Macro Debug menu item, then select **Show macro names** in the dialog box.

The FELIX *User Guide* gives a detailed description of how to use the standard interface to do virtually all of your NMR processing. Most users will never require more functionality than that provided by the standard user interface.

This FELIX *Command Language Reference Guide* is designed for advanced users who want to enhance, modify, and extend the abilities of FELIX for their special requirements. This guide contains complete descriptions of all the elements that comprise FCL.

FCL commands are two or three letters long, and in this documentation are presented in **boldface type**. Commands are case sensitive, although command arguments are not. (See "Case sensitivity" in Chapter 2, *FCL Basics*.) Symbols are also **boldfaced**, while parameters are *italicized*.

Using this guide

- This chapter gives a brief introduction to the major components of FCL.
- Chapter 2, *FCL Basics*, explains the syntax of FELIX command lines and the behavior of the FELIX command interpreter.
- Chapter 3, *Symbols and Expressions*, discusses the use of symbols and arithmetic expressions.
- Chapter 4, *Macros*, details writing macros and introduces the directives that control macro execution.
- Chapter 5, *Menus and Control Panels*, illustrates the building of real-time display macros that generate animated graphics, showing the effect of changing command parameter values in real time.
- Chapter 6, *The Database and Tables*, presents a detailed explanation of the database commands and explains how to store and extract information.

- Appendix A, *Command Reference*, lists the FCL commands and gives short descriptions of how they are used.
- Appendix B, *Symbol Reference*, lists the symbols used by FCL and gives descriptions of how they interact with FCL commands.
- Appendix C, *Macro Examples* gives a few examples to get you started writing your own macros.

NoteAll references cited in this guide are listed in Appendix A, *References* of the FELIX *User Guide*.

FCL command line

This section describes some of the basics of FCL and the conventions used in this document to describe the command syntax. When a command is shown in its general form, each parameter is indicated by an *italicized_word*. For example, the descriptive explicit forms of the command that defines the width of a spectrum (**swidth**) are shown below.

`def swidth` *`spectrum_width`* *(as described in the manual)*

`def swidth 4386` *(as you would enter it)*

To execute an FCL command, press <Enter> after you have finished typing it.

The commands within FCL consist of two- and three-letter acronyms that symbolize the action of the command. Each command performs some function or changes the state of the program in some way. Two examples of FCL commands are shown below:

```
ft

hcp
```

ft causes FELIX to perform a Fourier transform on the data in the workspace, and **hcp** generates a hardcopy output of the current spectrum display.

A list of the FCL commands along with a detailed description of their functions is found in Appendix A, *Command Reference*.

Symbols and expressions

FELIX allows you to use symbols to represent any value or parameter. This symbolic substitution allows flexibility within the program and within the macros, which in turn generates automatic processing within FELIX. There are two distinct types of symbols within the program: *reserved* symbols and *user* symbols.

Reserved symbols

Reserved symbols report the current status of the program or workspace. For example, when you read an FID or spectrum, the reserved symbol **datfil** is defined automatically as the name of that file. At a later point in time (usually within a macro), that same file may be read by using the reserved symbol **datfil** instead of entering the filename explicitly.

In addition, reserved symbols are used to define items such as the spectrum width (**swidth**), the number of contour levels (**nlevel**), and the datatype (**datype**). The quantity and names of reserved symbols are constant. They are initialized by FELIX when the program begins and cannot be deleted

User symbols

User symbols are any other symbols that are named and initialized by you or a macro while FELIX is running. Examples include the loop counter defined by the **for** command and the result of the **eva** command. You can initially define as many user symbols as you want, and you can delete them at any time.

FELIX can also evaluate complex mathematical expressions. Sometimes you may want to express a number or a symbol value arithmetically in terms of other numbers or symbols. FELIX performs expression evaluation, where an expression may contain numbers, symbols, and the arithmetic operators plus (+), minus (-), multiply (*),

and divide (/). Expressions may also contain trigonometric, logarithmic, and datatype functions like sin, cos, log, int, etc. Expressions must be enclosed in parentheses and may contain internal parentheses defining the order of evaluation. The expressions used in FELIX are similar to those found in the BASIC and FORTRAN programming languages.

See Appendix B, *Symbol Reference*, for more detailed information on individual reserved symbols. See Chapter 3, *Symbols and Expressions*, for more information on general symbol use and expression evaluation.

Macros

Macros are an essential feature of the FELIX data-processing approach. A macro is a text file containing a series of FELIX commands that directs the program to perform a specific task. In addition to direct commands, a macro can contain symbol substitution, arithmetic expressions, and flow-control statements (i.e., for loops, if/then/else statements, goto's, and subroutine calls). In effect, macros combine the capabilities of a programming language with the standard capabilities of FELIX, allowing you to design customized processing procedures.

FELIX comes with a library of macros capable of many common processing tasks. Using most text editors, you can modify FELIX macros to fit your specific needs. For example, the apodization function in the example macros will probably be customized by each user.

In addition to using macros for custom data processing and data analysis tasks, macros are also used to implement the FELIX GUI. Advanced FELIX users will create menus and control panels to design a custom environment.

Menus

The FELIX GUI contains a variety of menubars, pull-down menus, icons, and control panels that interact with the user to simplify data analysis and processing. The menu interface covers most aspects of NMR data processing, display, and analysis.The standard FELIX GUI exceeds the needs of most research groups.

While Accelrys has tried to include and place related processing and applications conveniently near each other within the GUI, it couldn't predict the needs of specific users. You can customize the FELIX interface to speed and simplify your NMR data processing and analysis by editing the menu interface macros provided or by writing your own.

For example, you can add a menu item to execute one of your own macros from within the interface. While this minor change to the menu system will not affect other aspects of the interface, it will enhance the power of the program for your specific application.

If your applications require repeated access to certain pulldowns, that pulldown can be removed from the top menu bar and placed elsewhere on the screen for easy access.

Creating a modification to the menu interface takes less than 15 minutes for someone experienced in writing FELIX macros. Making these changes require no access to source code.

The database

The FELIX database provides fast and efficient sorted access to large sets of spectrum data. Many FCL commands directly generate or read whole datasets, like *N*D peakpicking and volume integration. Generating baseline points creates a dataset in the database, and the commands for baseline correction read that same dataset of baseline points to flatten the spectrum's baseline.

FCL also includes an extensive set of core database commands that provide explicit access to every primitive function the database was designed to perform. You can have complete “read and write” access to any information stored in the database, whether it was generated by an FCL command or by a macro you wrote. This provides robust functionality with respect to storing, retrieving, and manipulating datasets of virtually any type.

One of these core database functions is the entity editor. It functions like a spreadsheet, giving you a graphical way to edit, add, and delete information in a dataset.

Please see Chapter 6, *The Database and Tables*, for more detailed information about the FELIX database.

2 FCL Basics

This chapter describes the essentials of using the FELIX command language (FCL). The intent is to provide a foundation for using FCL commands in simple and advanced implementations.

FELIX 2002 is a command- or event-driven software application. Software in this class waits for you to issue a command, acts upon that command, completes the task, and returns to a standby condition, awaiting the next command.

FCL is the fundamental interactive element in FELIX 2002. FCL is composed primarily of commands and their associated parameters, with a precise and simple syntax. There are more than 200 distinct FELIX commands, and nearly 200 associated reserved symbols.

Although many users interact with FELIX solely through the graphical user interface (GUI), the interface is mostly written using FCL.

Accessing FCL

You can directly access FCL by clicking the primary mouse button in the command window within the main FELIX window. If the command window was closed, you can open it by selecting **View/ Command Input** from the main menu.

Virtually any task that can be performed with the GUI can also be accomplished by using FCL directly. Whether you want to conduct novel processing and / or analysis procedures that are not available through the GUI, or to conduct standard tasks directly, you always have the option of issuing FELIX commands, either as discrete command statements or as a group of commands in a macro file.

Commands and command arguments

As an example, a typical FCL command statement might be:

```
rn myfile.dat
```

Here, the command **rn** (which specifies that a data file be read) is separated by a space from an argument, which in this case specifies the datafile to be read.

Many FELIX commands expect and require one or more associated input arguments, although not all of them do. For example, in these commands:

```
rn myfile.dat

ft

dr
```

only the **rn** command requires an argument, whereas the **ft** and **dr** commands execute without additional arguments.

FCL operates in two distinct functional states: **command-line** mode and **macro** mode. Command-line mode is a fully interactive text interface, as is used in the examples above.

If you are in command-line mode and you issue a command without associated arguments when that command normally requires them, then FELIX prompts you to provide the appropriate input. For example, if the **rn** command is issued without a datafile name, FELIX asks for one, as shown below:

```
rn

file=myfile.dat
```

Many FELIX command arguments are reserved symbols (see Chapter 3, *Symbols and Expressions*) and, where a reserved symbol has already been defined, as above (assuming you issued all the above commands), the parameter prompt displays the current value of the symbol. You can then accept the current value (by pressing <Enter>) or supply an alternative value (a filename in this case). If the command argument(s) have no current value, the prompt appears simply as:

```
rn

file=
```

Unlike command-line mode, macro execution mode is not an interactive state. Therefore, in designing a macro you must ensure that all commands used in the macro file contain all the appropriate input. This is not to imply that macros must be written using specific literal input designations, however. Symbolic input and output are discussed in Chapter 3, *Symbols and Expressions*.

Case sensitivity

FCL is case-insensitive; that is, FELIX 2002 does not distinguish whether input is upper or lower case. In fact, unaugmented character strings that are composed of mixed or uppercase text are interpreted as if they are all lowercase. For example, the following command statements are equivalent:

```
rn myfile.dat

rn MYFILE.DAT

rn MyFiLe.DaT
```

Of course, you may prefer to distinguish between upper or mixed case in FELIX. To do so, you must signal that you want the input string to be interpreted literally, by enclosing the string in single quotation marks. For example, each command statement below is distinct from the others:

```
rn 'myfile.dat'

rn 'MYFILE.DAT'

rn 'MyFiLe.DaT'
```

For each of these statements, FELIX tries to find and read a datafile whose name is *exactly* as shown.

Caution: Remember that, although FELIX 2002 is case-insensitive with respect to command arguments, the *commands themselves must contain only lowercase characters*. For example:

```
rn myfile.dat
```

is a perfectly valid command phrase. However, FELIX cannot interpret the following:

```
RN myfile.dat
```

or:

```
Rn myfile.dat
```

In addition to providing case-sensitivity, the single quotes may also be used to generate strings containing spaces. For example, using the **def** command (which allows you to define a symbol value) together with a string in single quotes, you can define phrases as symbol values:

```
def mytext 'This is a test message'
lis mytext


User Symbol      Value
mytext           This is a test message
```

Errors in FCL commands

If FELIX cannot recognize a submitted command, it prints an error message together with what it interpreted as an erroneous command in the output window. For example:

```
RN myfile.dat
Not a command: RN
```

Whenever FELIX indicates that a command phrase is invalid in some way, a special symbol named **status** also records the event by assuming a non-zero value.

> **Note:** The symbol is a variable. You can check its value by typing the command: `lis status` in the output window.

In this example, the event is said to have generated "bad status." There is little significance to this occurrence if you are in command-line mode, since a subsequent successful command execution resets the **status** to zero (clear status).

The effects are more problematic when such an error occurs during execution of a macro file. However, if "bad status" occurs during execution of a macro, no subsequent commands in the macro are executed. This feature is designed to protect datafiles and to avoid catastrophic failure of FELIX in the event of a fatal command error. The status may always be cleared manually by setting **status** to zero. (See Chapter 3, *Symbols and Expressions*.)

Display and context effects

Although many of the 1D vector-manipulation utilities (e.g., **ft**, **rev**, **exc**, and **red**) act on the entire contents of the 1D workspace regardless of whether the entire vector is displayed, some FELIX commands are display- and context-oriented.

For example, the **pic** utility, which discriminates and records local extrema in 1D, 2D, and *N*D spectra, specifically alters its function to agree with the most recently displayed data. If an expanded region of a 1D vector is drawn and the **pic** command is issued, as in:

```
exp

pic

entity = pic:1d_picks

select mode = 0
```

Here, the **pic** command searches for local extrema only along the portion of the 1D workspace that is currently displayed and then records the results in the default entity pic:1d_picks.

If, on the other hand, you are considering an expanded region of a 2D matrix file, that is:

```
cp
```

```
pic

(0=Pos,1=Abs_Mag,2=Neg): 0

entity = xpk:peaks

select mode = 0
```

then the **pic** utility searches only the displayed region for 2D local extrema, based on the distinct 2D / *N*D extrema-discriminating criteria that are specified using the **pic** command arguments (see the **pic -- Peak pick and label** command in Appendix A, *Command Reference*). The results are then recorded in the default 2D / *N*D entity xpk:peaks.

Other commands that are display- and / or context-sensitive include **drx**, **fit**, and **int**.

Line continuation

An individual command statement may be longer than is convenient or desirable (maximum allowed length is 256 characters). When writing a macro, you may want to continue long command statements over several lines to improve their readability. To do so, indicate that the end of a line is not the end of a command statement by ending the line with a special line-continuation symbol. In FELIX that symbol is the vertical bar **|**, which is entered by pressing <Shift>+\ on most keyboards. Using the **pic** command example above, we might alternatively have submitted the following equivalent command statements:

```
pic 0 xpk:peaks 0
```

or:

```
pic 0 |

xpk:peaks 0
```

or:

```
pic 0 |

xpk:peaks |

0
```

Note: Line continuation does not work in command-line mode. It works only in macro mode. The continuation line symbol must be separated by at least one space from the last character in the command phrase. That is:

```
pic 0|
xpk:peaks|
O
```

is interpreted literally as the command phrase:

```
pic 0xpk:peaks0
```

which cannot be interpreted because it does not contain the required spaces

File prefixes and suffixes

FELIX store data, graphical objects, processing parameters, and analysis information in a variety of external files. The convention of naming datafiles with the suffix *.dat* and matrix files with the suffix *..mat* is an artifact of the mechanism by which FELIX accesses the files it needs for various processing and analysis activities. For example, if you submit the command statement:

```
rn myfile
```

the FELIX interpreter recognizes that the **rn** command has been invoked. FELIX then affixes the directory prefix (i.e., path) to the file using the **datpfx** symbol and appends the *.dat* file extension to the file before it begins searching for that file.

Alternatively, if you submit the command statement:

```
rn myfile.dat
```

FELIX affixes the appropriate directory prefix to the file, but does not add a suffix.

This approach provides maximum flexibility and compatibility with other naming conventions. For example, if you submit the command statement:

```
re oldfile.ser
```

FELIX affixes the appropriate data directory prefix and searches for a file with the older *.ser* datafile suffix.

Macro file formats

Comment symbols

It is often convenient to enhance the readability of macro files by using *comments*. Comments may describe the macro function or remind you of important datafiles or argument dependencies.

The FELIX comment symbol is the semicolon (;). Below is a simple macro that demonstrates the use of comments:

```
ty Here is my macro! ; Type a message

;ty This line will not appear

rn myfile.dat ; Read my data file

dr

end
```

During execution, this macro prints only the line: "Here is my macro!"; then the macro reads the specified data file and draws it. The macro does not display anything following any semicolon on a line.

Tabs and spacing

Spaces and tabs may be freely used in macros to enhance readability, as shown below:

```
c**mymac.mac

; This macro counts to 10

      for loop 1 10

           ty Count=&loop
```

```
next
end
```

3 Symbols and Expressions

Symbol substitution is one of the most important and useful tools in FELIX 2002. Symbols are used in much the same way as variables in algebra; you can define processing algorithms that incorporate a symbol, and later you can decide on a value for the symbol. You can also store important numbers in symbols so that they may be quickly recalled later.

FELIX symbols are text strings that are evaluated at command execution time. Symbol names may be up to eight characters long and may contain any alphanumeric character. Any single symbol value can contain up to 128 characters.

To display the current symbol definitions and their values, use the list command (**lis**). The **lis** command makes use of the wild card character (*). For example:

```
lis h*
```

shows values for only those symbols beginning with the letter h. The result is displayed in the FELIX text-prompt (or *parent*) window. The output from the above example might look like:

```
1    hafwid 2
2    hardx0 0
3    hardxs 0
4    hardy0 0
5    hardys 0
6    hardmo 21
7    hilim1 0
8    ....   ....
9    harddv temp.plt
```

To delete or purge one or more of the user symbols, uses the **pur** command. This command also makes use of the wildcard character (*). For example:

```
pur count
```

deletes the user symbol **count**, and:

```
pur loop*
```

deletes all user symbols that begin with the letters "loop".

> **Caution:** Giving the **pur** command with no parameters does nothing. However, be aware that:
>
> ```
> pur *
> ```
>
> *deletes all user symbols*. This should only be done very carefully, if at all.

Reserved symbols

An important subset of the symbols used by FELIX are the *reserved* symbols. FELIX uses several reserved symbols to hold information about the internal structure of the program and data. Some reserved symbols define the current graphical state of the program, and others remember the current datafile names and display features. By referring to the proper reserved symbols, your macros can find out such things as whether a matrix is open, how large it is, the last contour level drawn, and whether the last macro command was executed successfully.

Types of reserved symbols

Reserved symbols generally fall into one of two classes: those that affect a subsequent command, and those that are updated as a result of a command's execution.

Reserved symbols that define the graphical state of the program

An example of reserved symbols that define the graphical state of the program are those symbols that are related to the contour plot command (**cp**). For each contour plot, certain reserved symbols define how the contour plot appears. These reserved symbols include those that define the contour level (**level**), the number of levels (**nlevel**), the contour level mode (**clmode**). Whenever the contour plot command is executed, FELIX looks at the current values of specific reserved symbols to determine what should be plotted.

Similarly, when you try to read a 1D datafile with the **re** or **rn** command, FELIX looks at the values of the symbols **datfil** and **datpfx** to determine the current datafile's name and the current data directory.

The values of the reserved symbols described above are usually set within control panels, but they may also be defined with the **def** command (see below).

Reserved symbols that are updated as a result of a command's execution

The second class of reserved symbol is exemplified by the **datsiz** symbol. This symbol contains the current data size as a number of points.

Although you can change the value of **datsiz** using control panels or the **def** command (see below), the symbol's value is automatically updated to the current data size after reading a datafile or loading a 1D vector from a matrix. Likewise, when FELIX reads a 1D datafile, the reserved symbol **datype** is changed automatically to reflect the appropriate datatype, whether it is real or complex.

Other reserved symbols of this class include **d1size** (the size of the current matrix in the D1 dimension) and **matfil** (the name of the current matrix). These are changed when you open a matrix.

A complete list and definitions of the reserved symbols are given in Appendix B, *Symbol Reference*. In addition, each command in Appendix A, *Command Reference* describes the reserved symbols that affect that command and those symbols that are updated by execution of the command.

User-defined symbols

You may also define *user symbols*. In general, user symbols are employed to save specific values so that they may be recalled later. In addition, user symbols are applied to remember the current state of the menu interface. For example, when a toggle or switch value is set, user symbols are included to remember that state or value.

Defining symbols and their values

FELIX includes several ways to define symbols and their values.

def command

One method to define symbols and their value is to use the define symbol command (**def**). For example, to set the value of the symbol **filnam** to the string "**test**", enter:

```
def filnam test
```

This command is used throughout FELIX macros, especially in the macro routines used for data processing. The **def** command may also be used to define reserved symbols such as **level**, which defines the lowest plot level for subsequent 2D plots.

cdf command

The **cdf** command (*conditional define*) lets a macro define a symbol only when that symbol does not yet exist. This gives greater flexibility when writing macros, because you can make sure a symbol exists and has a valid value without changing its previously defined value. Thus, you can make a macro that is exe-

cuted many times, yet only the needed symbols initialize the first time through, without any special effort.

eva command

Another way to define symbols is the **eva** command. It evaluates arithmetic expressions, mathematical functions, and database functions, and stores the resulting value in a symbol. More information on expressions and functions appears later in this chapter.

As the result of a command

Many FELIX commands accept a symbol name as the last parameter on their command line and define that symbol's value to reflect the answer or status of that command's result. For example, the command:

```
xpk who &pksent -1 whoitm
```

returns the item number of the cross peak that the user selected with the crosshair cursor. The symbol **whoitm** gets defined to the value of a cross-peak item number. If no cross peak was selected, **whoitm** gets the value zero.

Another example shows how to convert a data position in points into ppm:

```
ppm 0 0 512.0 ppmval
```

This command calculates the ppm position for datapoint 512 in the current 1D workspace and defines the symbol **ppmval** as that value.

A third example shows how to extract the reference information for one dimension of a matrix using the **rmx** command:

```
rmx -2 rfreq rwid raxis rpnt rval rtext
```

This command statement extracts the reference information for D2 from the current matrix. It defines six symbols with values corresponding to spectrometer frequency, spectrum width, etc.

Defining symbol values from control panels

The most common mechanism for defining symbol values in FELIX 2002 is from within its control panels. These graphical user interface (GUI) components prompt you for symbol values by using text explanations, switches, toggles, buttons, and scrolling list boxes. FELIX presents all the available choices in a format that simplifies making a selection, while minimizing the possibility of allowing invalid input. Almost every element in a control panel defines a symbol to reflect its state or contents. A more detailed description of defining symbols using control panels is found in Chapter 5, *Menus and Control Panels*.

Symbol substitution

Symbol substitution is one of the most important and useful tools within FELIX 2002. To access and use the value of a symbol, you must precede the symbol name with an ampersand (&). Whenever FELIX encounters an ampersand in a command line, it attempts to perform symbol substitution on the text

string that follows the ampersand. For example, if the symbol **filnam** currently has the value "test", the command line:

```
rn &filnam
```

is interpreted as:

```
rn test
```

before it is executed. In this example, FELIX attempts to read a datafile named *test.dat*. The symbol **filnam** has been replaced by the text string "**test**" in the macro.

You can use the value of the reserved symbol **datsiz** in a command, as follows:

```
sb &datsiz 0
```

which applies a zero-degree shifted sinebell function over the number of points specified by the symbol **datsiz**.

Complex symbol substitution

Some applications require more complex symbol substitutions. The following examples illustrate a few of the many possibilities.

First, a symbol can be concatenated with normal text or numbers. For example:

```
rn test&number
```

reads a file named *test1.dat* if the value for the symbol **number** is equal to the number one. In a macro, the symbol **number** could be changed at strategic points, allowing you to read a series of datafiles with a single read statement containing a symbol variable. Such use of symbols may be necessary when transforming 3D data, because the initial data in these large matrices may be contained within more than one serial file.

Second, two symbols can be concatenated by placing the symbols adjacent to each other. If the symbol **filnam** had a value "abc" and the symbol **number** had a value "001", the command statement:

```
ty &filnam&number
```

would be equivalent to **ty abc001**. In this manner, several symbolic components can be combined into a single string.

Finally, symbol substitution is recursive, as illustrated by the following example. If the symbol **filnam** had a value of "abc" and the symbol **abc** had a value of "hello", the following command statement:

```
ty &&filnam
```

would produce "hello". In this case, the sequence of substitution is:

1. Find the string for the symbol **&filnam** and replace it with its value "**abc**", thereby leaving "**&abc**" on the command line
2. Since an ampersand remains on the command line, do another symbol substitution.

In the example, the ""**&abc**" string is replaced by the value of the symbol **abc**, namely "hello".

It is also possible to delay symbol substitution until later, by adding an exclamation point (!) to the beginning of a symbol name. For example, the command statement:

```
def test !&filnam
```

defines the symbol **test** to literally be **&filnam**. The most common use of this symbol-definition method is in combination with the macro **put** command for building macros from the FELIX command language.

Local symbols

FELIX 2002 supports two classes of user symbols: *local* and *global*.

Local (user-defined) symbols are used only in the macro in which they are defined and are automatically purged when that macro is finished. In addition, multiple macros can use the same (local) symbol names, without any risk of changing another macro's local symbols.

All local symbols begin with an underscore, "_". This is the only difference between local and global symbols.

Defining and using local symbols is the same as for global symbols. The following example revises the simple macro *test1.mac* to use a local symbol instead:

```
1   c**test1.mac
2   for _count 1 5
3         ty Count = &_count
4    next
5   ex &return
6   end
```

As soon as this macro is finished, the symbol **_count** becomes undefined.

> **Note:** In writing macros, all loop counters and other symbols used locally in the macro are preferred over local symbols. This keeps the size of the symbol table from getting too large and minimizes the chance of macros causing unintended side effects on other macros

Global symbols

FELIX 2002 supports global user symbols in addition to local reserved symbols and local user symbols (see above). This feature greatly enhances the flexibility of working with multiple connected frame layouts.

By default, the value of a symbol is local to the graphics frame where it is defined. This allows a frame to act independently, without affecting the content or behavior of other frames. To share symbols between frames, it is necessary to explicitly export or import the symbol values to or from another frame.

While this notion of local symbols is good for working with multiple independent frames, it can cause significant problems when working with multiple connected frames all operating on a common dataset.

Global user symbols address this problem. All global user symbols are always known to all frames, and defining the value in any one frame defines that symbol globally in all frames.

Defining global symbols

To identify a symbol as global rather than local, begin the symbol name with the character string "**g_**". All symbol names beginning with "**g_**" are global, and all other symbols are local. There are no global reserved symbols, only global user symbols. All the rules for symbol substitution are exactly the same for global symbols, and a global symbol may be used anywhere a local symbol may be used.

> **Note:** For optimal speed when using FELIX, do not use more global symbols than needed. Local user symbols have somewhat faster lookup access, and too many global user symbols can slow down all symbol lookup access. A good rule of thumb is to limit the number of global symbols to no more than ten percent of all user symbols.

Arithmetic expressions

FELIX 2002 allows arithmetic expressions to replace any numeric parameter. An arithmetic expression may contain the arithmetic operators plus (+), minus (-), multiply (*), divide (/), and power (^), with left and right parentheses being used to define the order of operations. The syntax for expressions within FELIX is exactly the same as in FORTRAN or BASIC. FELIX also supports the unary (-) operator, which negates the variable, symbol, or expression immediately to its right.

When used for numeric parameters, arithmetic expressions must be enclosed in parentheses without any space between the numbers, operators, and parentheses. This allows FELIX to evaluate the expression before passing the parameter value on; otherwise, the parameter would contain non-numeric characters and be uninterpretable.

Sometimes you may want to force evaluation of an arithmetic expression: the **eva** (evaluate) command is useful for doing this. For example:

```
eva temp (4*3+1)
```

forces the value of the symbol **temp** to be **13**.

> **Remember:** Symbol values are just character strings. FELIX does not interpret them as numbers until they are used as parameters by a command.

Expressions are even more powerful when they are combined with user-defined symbols. For example:

```
eva temp (4*3+&shift)
```

determines a number for the symbol **temp**, whose value depends on the current value of the symbol **shift**. The ability to evaluate arithmetic expressions in combination with symbols provides a very powerful method of data manipulation.

Expressions are most often used within FELIX for determining data characteristics. For example, to determine the number of points currently displayed on the screen, the following expression can be used:

```
eva scnsiz (&last-&first+1)
```

The value for the symbol **scnsiz** is the difference between the symbols **last** (the last-displayed datapoint) and **first** (the first-displayed datapoint). After the above expression, your macro would usually contain a sequence of commands using the value of **scnsiz**.

A slightly more complex expression finds the value of the midpoint displayed on the screen:

```
eva midpt ((&first+&last)/2)
```

Another use for expression evaluation is to manipulate values found in the database. For instance, specific elements can be loaded into symbols, with the resulting symbols being evaluated via expressions.

A simple example is to find the separation between two resonances found in a 1D spectrum. First, load the first two resonance positions into the symbols **cen1** and **cen2** using the standard database-access methodology:

```
dba element load pic:1d_picks.1.cenpnt cen1

dba element load pic:1d_picks.2.cenpnt cen2
```

Next, evaluate their separation, using symbol evaluation:

```
eva separ (&cen2-&cen1)
```

To view the resulting symbol for the separation, use the **lis** command:

```
lis separ
```

Or within a macro, use the **ty** command to print comments along with symbol values to the parent window:

```
ty Separation between item 1 and 2 is: &separ
```

In a macro, you may also check to ensure that the resultant value is a positive number, and if it's not, take its absolute magnitude:

```
if &separ lt 0 then

    eva separ (abs(&separ))

eif
```

A more complicated example of expression evaluation makes use of an exponential or power evaluation. The example shown below uses NOESY cross-peak volumes to measure distance constraints. This example is used for measuring proton-proton distances from an NOE volume at a single mixing time by using the sixth-power dependence ratio as it relates to a fixed distance (**dissum**) and volume (**volsum**):

```
dba element load &volent.10.vol1 volum1
```

```
eva dist ((((&dissum)^6.0)*&volsum/&voluml)^(1.0/6.0))
```

The distance may be further modified to reflect upper and lower bounds:

```
eva lower (&dist-(&dist*0.1))

eva upper (&dist+(&dist*0.1))

ty Lower bound is: &lower

ty Upper bound is: &upper
```

Integer vs. real expressions

FELIX 2002 supports both integer and floating-point arithmetic expressions. Several algorithms require integer division.

When any of the operands in the expression has a decimal point, then the expression is evaluated as real and the result is real. If no decimal points are present in the operands, then the expression is evaluated as integer and the result is integer. Thus there is a difference between the following two values:

```
eva int (17/2)

eva real (17.0/2.0)
```

The value of **int** is **8**, and the value of **real** is **8.5**.

Complex mathematical functions

FELIX 2002 allows some arithmetic functions to be evaluated and saved as symbols. These mathematical functions can appear anywhere that simple expressions may be used. The arithmetic functions currently supported are:

abs - absolute value

acos - arccosine

asin - arcsine

atan - arctangent

cos - cosine

exp - exponential

float - turn integer to real

int - truncate real to integer

log - natural logarithm

log10 - logarithm base 10

nint - round real to nearest integer

ran - random number generator

sqrt - square root

sin - sine

tan - tangent

The trigonometric functions require all angles to be in degrees. These functions are combined using the standard evaluation nomenclature to calculate resultant values.

The **ran** function returns a random value between zero and the value of its argument. For example, **ran(1.0)** returns a value between zero and one, and **ran(500.0)** returns a value between zero and five hundred. It may be used in any arithmetic expressions and in the **eva** command:

```
eva value (ran(1.0)
```

Working with the example above for finding the absolute value of a negative peak separation, we can use the following sequence of commands in a macro:

```
..
if &separ lt 0 then
eva separ (abs(&separ))
eif
...
```

In other words, the above "if / then / else" sequence could be replaced by the absolute value calculation, since only positive values are acceptable in this instance.

Another example is used to ensure that each dimension of a new matrix has a size of root 2:

```
1   ...
2   eva n2size (log(&d2size) / log (2.0))
3   eva n2int (nint(&n2size))
4   if &n2size eq &n2int onward
5   ...
```

Again, examples such as these may be added to your own macros.

Database functions

FELIX 2002 supports one other class of functions that evaluate the state of database components.

These functions are only valid in the **eva** command and cannot be combined on the same command line with other functions or expressions. The form of all database functions is **db$*nnnn*(*arg*)**, where *nnnn* is the function name and *arg* is the argument passed to the function. More information can be found in Chapter 6, *The Database and Tables*.

The function:

```
eva open db$open()
```

determines whether a database is open. The symbol **open** is set to zero (not open) or one (open).

The function:

```
eva exist db$entity(xpk:peaks)
```

determines if an entity with the name xpk:peaks exists in the current database. The symbol **exist** is set to zero if no entity with that name exists and to non-zero if the entity does exist.

The complete list of database functions is shown below:

db$open()	is a database open?
db$entity(*ent*)	does an entity named *ent* exist?
db$val(*ent.item.elem*)	extract one value from database
db$is_item(*ent.item*)	does an item number *item* exist?
db$big_item(*ent*)	return the biggest item number in entity
db$schema(*ent*)	return the schema name for this entity
db$occur(*ent*)	return the occurrence count for this entity
db$element(*ent,elem*)	find an element number by its name

4 Macros

Macros are an essential feature of the FELIX data processing philosophy. Macros combine the capabilities of a programming language with the standard capabilities of FELIX 2002, allowing you to design customized processing procedures.

A macro is a text file containing a series of FELIX commands, which together perform a specific task. You can edit a macro file using most text editors. In addition to direct commands, a macro can contain symbol substitution, arithmetic expressions, and flow control statements (that is, **for** loops, **if** / **then** / **else** statements, **goto**'s, and subroutine calls).

FELIX 2002 comes with a library of macros that execute many common processing tasks. You can modify these macros to fit your specific needs. For example, the apodization function in the example macros in Appendix C, *Macro Examples*, will probably need to be different for each user.

In addition to using macros for custom data-processing and -analysis tasks, macros are also used to implement the FELIX 2002 graphical user interface (GUI). In fact, the advanced user will find it easy to create control panels and menus to design a custom environment.

Macro directories

You must store custom macros in directories where FELIX can find them. The FELIX 2002 macros that drive the GUI are found in the macros folders under the installation directory (by default it is **C:\Program Files\Accelrys\Felix 2002**. This directory is defined by the reserved symbol **macpfx**. The user macros can be stored in a directory defined by the reserved symbol **macpf4**. To keep things simple, you should initially keep all user-written macros in the directory defined by the symbol **macpf4**. The maximum length for a

directory pathway is 128 characters, the same as the maximum length for a single symbol value.

When FELIX searches for macro files, it first looks in the current working directory, then in the directory defined by the symbol **macpfx**. This symbol is given an initial value in the *.felixrc.ini* configuration file the installation directory (by default it is **C:\Program Files\Accelrys\Felix 2002**) that defines the initial directory pathway so the GUI runs by default. If a named macro is not found in the directory defined by the **macpfx** symbol, FELIX then searches in the directory defined by the symbol **macpf1**. This logic continues up to the directory defined by the symbol **macpf9**.

Writing macros

You can write or edit a macro while still running FELIX 2002. To create or change a macro, it is easiest to use Notepad or another text editor. Any editor may be used as long as the macro files *do not* contain formatting characters and *do* contain standard carriage control.

It is standard for the first line of any macro to contain the macro's name, though this item is optional. FELIX ignores any macro line that starts with **c****. The beginning of a typical macro is shown below. Note that the line numbers at left are used for reference only, and should not be included in the macro:

```
1    c**users.mac
2        set 0
3        dr
4        ...
```

Macros may also contain tab characters and blank lines, so that they may be organized so as to enhance readability. In addition, any characters that come after a semicolon (;) are ignored by the macro interpreter. This makes it easy to add comments to macros as shown below:

```
1    c**users.mac
2
3
4
5        fra o 10 10 10 10     ;Open first graphics frame
6        fra t 1
7
8
9
10        fra o 30 30 300 300     ;Open second graphics frame
11
12
```

```
13
14        set 1
15        sb 600 90
16        dr     ;draw sinebell window
17
18
19
20        ret ;return to GUI
21        end
```

Since macros are simply a series of FELIX commands, you must tell the program when to return control from the macro interpreter to the command interpreter. To do that, add an **ret** statement before the last **end** statement.

This ensures that the interface returns to the appropriate graphical state upon completion of the macro.

Executing macros

Once you have created a macro, the standard way to execute it is to issue the execute macro command (**ex**) followed by the name of the macro. For example, the command sequence shown below executes a macro named *test1.mac*. FELIX searches for the macro in the current working directory and then in the directory defined by the symbol **macpfx**, followed by the directories defined by the symbols **macpf1**, **macpf2**, **macpf3**, ..., **macpf9**.

```
ex test1.mac
```

or:

```
ex test1
```

This command may be issued interactively in the command window, or, alternatively, may be part of an existing macro. The following macro illustrates how a logic operation can determine subsequent macro tasks.

```
1    c**color.mac
2        if &color eqs red then
3            ex red_draw.mac
4        els
5            ex blue_draw.mac
6        eif
7        ...
8        end
```

Another way to execute a macro from the command window is to input the name of the file preceded by a period. This is a quick shorthand notation that can replace the above example. For example:

```
> .test1
```

executes the macro named *test1.mac.*

Passing arguments to macros

The mechanics of this feature are based on local symbols. Everything following the macro name on the command line is made available to the called macro in the form of local symbols.

When the called macro begins executing, one or more new local symbols are defined. This applies when the macro is executed by **ex**, **cal**, or **exr**. These symbols always have the names: **_args**, **_arg1**, **_arg2**, etc. The first of these, **_args**, is the number of arguments passed to this macro. When no arguments are given, **_args** is zero. Otherwise, there are additional symbols **_arg1**, **_arg2**, etc., each of which is the value of the n^{th} argument passed to this macro.

> **Caution:** Do not use an argument number greater than the number of arguments actually passed to the macro.

For example, the command line:

```
> ex test 13 4
```

causes the macro named test.mac to begin executing, with the following local symbols:

```
_args = 2
_arg1 = 13
_arg2 = 4
```

The command line:

```
> exr find 'Look for' h* n*
```

causes the find.mac macro to begin executing, with the following local symbols:

```
_args = 3
_arg1 = Look for
_arg2 = h*
_arg3 = n*
```

Loops

FELIX macros allow loop structures that are very similar to the loops in the BASIC and FORTRAN programming languages. Loops allow a set of operations to be carried out iteratively. Loops are necessary for processing multidi-

mensional data for which each vector along a dimension is processed in exactly the same way. Rather than typing in the same sequence of commands several thousand times, you can type in the processing sequence only once and use a loop to apply the specified sequence any number of times.

The basic layout of the loop structure is shown below:

```
1    for symbol start_value end_value increment
2        ...
3        ...
4    next
```

The argument ***symbol*** is the name of the loop counter whose value is incremented to reflect the current integer status of the **for** loop.

The argument ***start_value*** defines the initial value of the loop counter, and the argument ***end_value*** defines the last value for the loop counter. Both these arguments are inclusive with respect to their values, may be positive or negative, and may be a real number or an integer. If either of these values is a real number, the actual value used is the closest integer value.

The argument ***increment*** is the loop counter increment. This defaults to 1 if omitted, but may be set to any non-zero integer. All arguments for the loop sequence may be in the form of symbols.

Macro loops are related to symbol substitution, in that each loop maintains a counter symbol that is incremented each time through the loop. A simple example of a macro containing a loop is:

```
1    c**test1.mac
2    for count 1 5
3        ty count = &count
4    next
5    ex &return
6    end
```

The loop structure is defined by the **for** and **next** commands. Commands enclosed between the **for** and **next** are executed while incrementing the value of the specified counter symbol for each loop (the symbol **count** is incremented from **1** to **5** in steps of one). In this macro, the only command inside this loop is the type command (**ty**), which prints text defined by its command line. The output of the above macro loop is:

```
count=1
count=2
count=3
count=4
count=5
```

The output created by this macro illustrates the operation of **for** loops. Each time through the loop, the **for** counter (**count** in this example) is set to a value corresponding to the number of times the loop has repeated. Symbol substitu-

tion replaces the **&count** parameter with the value of the **for** loop counter, and the **ty** command explicitly types out the text following the **ty** command.

For loops may be nested up to 16 levels deep; however, each loop must have a unique counter symbol. An example of a nested **for** loop is:

```
1    c**test2.mac
2        for row 1 3
3            for col 1 4
4                ty row=&row col=&col
5             next
6        next
7        ex &return
8        end
```

The inner loop (which affects the symbol **col**) increments through all its values, the outer loop (**row**) increments one, and then the inner loop increments through all its values again. Thus the **ty** command is executed 12 times, for all the combinations of the **row** symbol from **1** to **3** and the **col** symbol from **1** to **4**. Nested loops are useful for processing multidimensional spectra.

Interrupting a macro

You may want to interrupt macro execution prior to completion. Getting stuck in a lengthy loop is not uncommon when writing new macros for novel data processing. To stop a running macro, type <Ctrl>+\ (press the <Ctrl> and backslash keys simultaneously) or <Ctrl>+c.

For more elaborate handling of interrupt signals, you may use the **esc** command to escape from a loop. To do this, you must include a specific command structure in your macro that uses the escape command (**esc**), with the format:

```
esc symbol_name
```

Each time the ***esc*** command is encountered, the program updates the argument ***symbol name***. If the <Esc> key is not in the keystroke queue (i.e., you don't press <Esc>), the value for ***symbol name*** is set to **0**. If the <Esc> key appears in the keystroke queue (meaning you *did* press <Esc> on your keyboard), the value for <symbol name> is set to **1**. As shown in the following example, you can then include logical branching in your loop to specify how the macro is halted:

```
1    c**escape.mac
2        for loop 1 9999
3            ty &loop
4            esc out
5            if &out ne 0 quit
6        next
```

```
7    quit:
8        end
```

Loop application: Reversing a matrix

The following is a simple macro used for reversing each 1D vector in a matrix. This example illustrates many of the features described above. It uses the reserved symbol **d2size** to ensure that each vector along the second dimension is acted upon accordingly:

```
1   c**reversed2.mac
2         for row 1 &d2size   ;for each row in the matrix
3   loa 0 &row          ;load the row
4   rev                 ;reverse the data points
5         sto 0 &row    ;store the row back
6         ty Row= &row $ ;type the row number,no scrolling
7         esc out       ;check for escape key
8         if &out ne 0 quit ;quit if escape hit
9         next                ;loop to next row
10 quit:
11        end
```

Loop application: Accessing the database

The following is a macro for deleting cross-peak footprints along the first dimension for a defined region. A macro such as this might be used to remove cross-peak footprints that are part of a T1 streak, which may be defined by the user symbols **d1low** and **d1high**:

```
1    c**t1streak.mac
2         dba list zero 1 xpk:peaks ;zero list 1
3         get 'D1 low point: ' d1low ;get low limit
4         get 'D1 high point: ' d1high ;get high limi
5         dba list range 1 xpk:peaks cen1 &d1low &d1high
pknum
6         for loop 1 &pknum ;for each peak number
7               dba list load 1 &loop peak
;load the first peak number
8               drx xpk:peaks.&peak 0 ;undraw the footprint
```

```
9                 dba item delete xpk:peaks.&peak
;delete that footprint

10            ty Deleting peak number: &peak          ;talk
about it

11            esc out ;check for escape

12            if &out ne 0 quit ;quit if escape

13        next

14  quit:

15  ret

16  end
```

Branching statements

go statements

Macros allow unconditional branching using variations of the FORTRAN goto or the switch statement in C. The structure of this command is:

```
go label
```

where the argument label is a specific label identified in the macro.

Labels are identified by adding a colon (:) to the end of the label name. For example, the statement:

```
go quit
```

looks for the label shown below:

```
quit:
```

The **go** command is very useful when working within interactive macros that prompt you for replies or that need to make decisions based on the values of symbols that are set with toggles or switches.

Below is an example of a simple menu that uses a **go** statement in this way. The only argument of the **go** command is a label name, which may be explicit or symbolic; for example:

```
1    c**gomacro.mac

2        ....

3        ....

4        go quit

5        ....

6        ....

7    quit:
```

```
8         ty All done!
9         end
```

When the **go** command is executed, the macro jumps to the specified label (in this case, quit:) and continues executing.

Labels may be up to sixteen characters long and represent a specific place within a macro file. Within any macro, each label name must be unique. When a label is branched to, the commands that follow it are executed. In the example shown above, after the **go** branch to the quit: label, the message ***All done!*** is printed to the screen. If the specified label is not found within the macro, the macro continues on to the next line as if no **go** statement existed.

gto statements

Macros also allow branching using a case-type go to command (**gto**). This command determines branching based on a fixed integer, which is usually a number defined by a symbol. The format of this **gto** command is:

```
gto symbol_value value0 value1 value2 valueN
```

If the argument *symbol_value* equals **0**, the macro branches to the label defined by *value0*. Likewise, if the argument *symbol_values* equals **2**, the macro branches to the label defined by *value2*.

In the FELIX 2002 GUI, the **gto** branching statement is used to make decisions based on symbol values defined by control panel toggles or switches. For example, a symbol defined by a toggle can be used to choose between Fourier transform options. For this example, if the particular symbol called **fttype** (defined by a specific toggle) has a value of **0**, a FFT is specified. If the symbol **fttype** has a value of **1**, a BFT is needed. We could then create the following macro to make use of this information:

```
1    c**toggle.mac
2         mnu p transform 20 4
3         if &button eq 0 quit
4         gto &fttype fft bft
5    fft:
6         ft
7         dr
8         go quit
9    bft:
10         bft
11         dr
12         go quit
13    quit:
14         ret
```

```
15        end
```

gif statements

Macros also allow branching using the arithmetic if command (**gif**). This command determines branching based on the sign of a fixed integer, which is usually defined by a symbol. The format of the **gif** command is:

```
gif symbol_value value_lt_0 value_eq_0 value_gt_0
```

In practice, if the argument ***symbol_value*** is less than **0**, the macro branches to the label defined by ***value_lt_0***. If the argument ***symbol_value*** equals zero **0**, the macro branches to the label defined by ***value_eq_0***. And if the argument ***symbol_value*** is greater than **0**, the macro branches to the label defined by ***value_gt_0***. The **gif** statement is commonly used in GUI macros.

if statements

Macros can contain **if** statements for conditional branching. **if** statements are similar to **go** statements in that they can result in a branch to a specified macro label, but differ in that the branching may or may not happen, based on the specified condition. The first type of **if** statement consists of a conditional and a target label:

```
if conditional label
```

Another type of **if** statement, the *if/then/else*, is used to control the execution of blocks of code depending on the state of the conditional. The form of the if / then / else is:

```
if conditional then

...

else

...

eif
```

The most basic conditional consists of two strings and a relational operator and has a value of either true or false. The conditional is evaluated by comparing *string1* with *string2* using the specified relational operator. A typical **if** statement is:

```
if &number eq 0 quit
```

This command compares the value of the symbol **number** to **0** using the **eq** (equal to) relational operator. If the symbol **number** equals **0**, the value of the conditional is true, and a branch is made to the label quit:. If the symbol **number** does not equal **0**, the conditional is false, no branch is taken, and the next command in the macro is executed.

Conditionals use two different types of relational operators. Arithmetic relationals interpret the strings as numbers, and string relationals interpret the strings as ASCII characters. Because numbers and ASCII strings do not sort in the same order, you must be aware of the ASCII code when using string rela-

tionals other than **eqs** and **nes**. A list of the relationals is shown below for both arithmetic and string characters:

Arithmetic relationals:

eq -- equal to

ne -- not equal to

lt -- less than

gt -- greater than

le -- less than or equal to

ge -- greater than or equal to

String relationals:

eqs -- equals

nes -- not equals

lts -- less than

gts -- greater than

les -- less than or equal to

ges -- greater than or equal to

Conditionals can be combined using **and** and **or** operators to test more than one relation at a time. The **and** operator yields a true result only when it operates on two true conditions. The **or** operator yields a true result if either conditional is true. Examples of conditionals containing the **and** and **or** operators are:

string1 operator string2 and *string3 operator string4*

string1 operator string2 or *string3 operator string4*

An example macro that uses **if** commands is given below. This macro plays a guessing game using a binary search and uses a variety of flow-control commands that are discussed above.

```
1    c**test4.mac

2        ty Guess a number between 1 and 10=

3    ready:

4        get "Are you ready? (y/n):" yesno

5        if &yesno nes y ready
;ask again if not ready

6        def lo 1 ;define the necessary symbols

7        def hi 10

8        def old 0

9    again:

10        eva middle ((&hi+&lo-1)/2) 1
;guess middle of range

11        if &hi eq &lo done        ;we are done
```

```
12        if &middle ne &old guess
13        if &middle eq &hi mlo
14        def middle &hi
15        go guess
16    mlo:
17        def middle &lo
18    guess:
19        get "Is it bigger than &middle?" yesno
20        def old &middle
21        if &yesno nes y no
22        eva lo (&middle+1)
23        go again
24    no:
25        def hi &middle
26        go again
27    done:
28        ty The number is &middle.
29        ret
30        end
```

This macro uses **if** commands containing both numeric and string relational operators. The string relational **nes** is used to distinguish between the user responses, while the numeric relations **eq** and **ne** are used to compare numbers. The **eva** (evaluate) command is used twice to force evaluation of expressions.

Another macro example using **if** statements follows. In this example, the **gv** command gets the value of a datapoint from the workspace and places it in a specified symbol, to find the largest, smallest, and average value in the workspace:

```
1    c**test6.mac
2        gv 1 bigpt
3        def smlpt &bigpt
4        def tot 0
5        for point 1 &datsiz
6            gv &point thispt
7            if &thispt gt &bigpt newbig
8            if &thispt lt &smlpt newsml
9            go loop
```

```
10        newbig:
11            def bigpt &thispt
12            go loop
13        newsml:
14            def smlpt &thispt
15        loop:
16            eva tot (&tot+&thispt)
17        nex
18        eva avg (&tot/&datsiz)
19        ty Biggest=&bigpt
20        ty Smallest=&smlpt
21        ty Average=&avg
22        ret
23        end
```

This example first sets the symbols **bigpt** and **smlpt** to the value of the first datapoint and the symbol **tot** to zero. The macro then loops through all datapoints (from 1 to &**datsiz**), comparing values with **bigpt** and **smlpt**. If any point is larger than **bigpt** or smaller than **smlpt**, the new large or small point is saved. The sum of all point values is accumulated in **tot**, which is used to calculate the average.

if/then/else statements

Macros can also contain **if/then/else** structures for more orderly flow control. The **if/then/else** statements may be nested up to 16 deep. The basic **if/then/else/eif** structure is:

```
if conditional then
...
els
...
eif
```

The condition is evaluated by comparing *string1* and *string2* using the specified relational operators described above. A typical if / then / else / eif statement is:

```
if &number eq 0 then
    ty ----------
    type Symbol not defined
els
    eva count (&value/&number)
```

```
    ty ----------
    ty Symbol count is: &count
eif
```

In the example shown above, the if / then / else / eif structure is used to prevent a divide by zero error if the value of the symbol **number** is 0.

ifx statements

Macros can contain **ifx** statements for conditional macro execution. **ifx** statements are similar to **if** statements in the arguments required; however, they differ in that they result in the execution of another macro instead of branching to a label in the same macro. The basic **ifx** command consists of a conditional and a macro filename:

```
ifx conditional macro_name
```

The conditional is evaluated in the same way as in the **if** statement. A typical **ifx** statement is:

```
ifx &number eq 0 root
```

This command line compares the value of the symbol **number** to **0** using the **eq** relational operator. If the symbol number equals **0**, the macro named root.mac is executed.

exr statements

The **exr** command (execute and return) is a powerful way to execute macros. This command brings the concept of a subroutine call, as used in high-level programming languages, into the FELIX command arsenal.

The **exr** command preserves where you are in the current macro, loads and runs the new macro, and resumes exactly where it left off when the new macro finishes. Macros executed via **exr** may be nested up to eight levels deep.

Macro-specific commands

Several commands are used only within macros. For example, the **esc** command described above is useful only in a macro. There is no reason to check for an <Esc> key unless a macro is running, since otherwise there is no process to interrupt. Some other examples are shown below.

com statements

Communicate statements (**com**) pause an executing macro, prompt the user for a single command line, execute that one command, and then restart the macro after the pause point. For example, the **com** command can be used to test a range of window functions as they affect a dataset:

```
1    c**read1d.mac
```

```
2        rn test.dat
3        com
4        bft
5        dr
6        end
```

In the above example, at the **com** command (line 3), a FELIX prompt appears and waits for the user to enter any FELIX command they desire.

err statements

Error statements (**err**) define an error trap destination for a macro. If any command sets the value of the reserved symbol **status** to non-zero, it means that command didn't function properly. If you use an **err** statement, the macro resets the status to **0** and continues executing where the error trap indicates. There are currently two types of **err** commands. If the command:

```
err label label
```

is encountered, the macro branches to the specified label. If the command string:

```
err macro macro
```

is encountered, macro interpretation is transferred to the specified macro. Error traps are most commonly used in macros that read user data.

ty and tym statements

The **ty** statement types the following text to the output window; the **tym** statement types the following text to the status bar. This text can be used for detailing the status of any macro.

For example:

```
tym All done!!
ty Data size is: &datsiz
```

> **Note:** Symbols may be substituted in any macro command.

Using FELIX to build macros

Macros may also be created from within FELIX 2002 using simple program commands. This technique is useful for saving a sequence of commands for execution later. For example, spectrum annotations are saved to a macro file. When you want to redisplay the annotations, simply execute that annotation macro.

Before building a macro using FELIX commands, check for the presence of a macro file with the name you want to use. Do this in your macro with the inquire (**inq**) command, which accepts the input string:

```
inq prefix file_name symbol
```

The *prefix* argument reflects the filetype based on the prefix, whether it is a macro (mac), annotation file (.ann), datafile (.dat), or matrix (.mat) file.

For files without prefixes, set the *prefix* parameter to **nul**. Based on the prefix type, FELIX uses the directory defined by the *pfx symbol to locate the file. For example, if *prefix* is mac, FELIX looks in the directory defined by the symbol **macpfx** to determine whether a file named *filename* exists. If the file is found, the symbol defined by the argument *symbol* is given a value of **1**. If the file is not found, the symbol defined by the argument *symbol* is given a value of **0**. Below is an example that uses the **inq** command to determine whether a file named test exists in the macro directory:

```
1    c**inquire.mac
2        def filnam test
3        inq mac &filnam exist
4        if &exist eq 1 fileok
5        ty ----------
6        ty File &filnam not found.
7        go quit
8    fileok:
9        ty ----------
10        ty File &filnam exists.
11    quit:
12        ret
13    end
```

Before building a macro using FELIX commands, you must first open that file. The open file command (**opn**) performs this function and accepts three arguments:

```
opn prefix file_name overwrite/append
```

The ***prefix*** parameter defines the prefix for the new file and places that file in the directory defined by the corresponding **pfx** symbol. As discussed above, if ***prefix*** is mac, the resulting macro file is created in the directory defined by the symbol **macpfx**.

The ***file_name*** argument is the name of the file to be created. The ***overwrite/append*** argument defines whether you want to overwrite an existing file (0) or append to an existing macro file (1). Setting ***overwrite/append*** to **1** is useful for modifying macros that already exist, without recreating them from scratch. Two files can be opened simultaneously, providing one file is opened for output only.

Once a macro file has been opened for writing, the **put** command (**put**) is used to add text to the file. The **put** command adds all text following it to the file, including spaces. Quotes are not needed. Sample **put** statements are:

```
putdef annsize &annsiz

putdef thick &thick

putdef level 1.0
```

Open macro files must be explicitly closed before execution, using the close file command (**cls**), which requires no arguments.

Since you are creating a macro, it is very important that before closing it, appropriate end and return statements are added to the file. The **end** command must be the last line in a macro in order for the program to switch from the macro interpreter to the command interpreter. Similarly, the **ret** statement must be at the end of a macro if you want it to return to the GUI. An example of building a macro is shown below:

```
1    c**build.mac
2        def return root
3        opn mac cplevel 0
4        putc**cplevel.mac
5        putdef level &mylevl
6        putdef nlevel &mynlev
7        putdef pencol &mycolr
8        putdef cycle &mycycl
9        putex &return
10        putend
11        cls
12        end
```

The resulting macro, named *cplevel.mac*, is created in the directory defined by the symbol **macpfx**:

```
1    c**cplevel.mac
2        def level 1.2
3        def nlevel 10
4        def pencol 2
5        def cycle 3
6        ex root
7        end
```

This macro can now be executed.

When building macros in this way, you may want to place a symbol-definition statement point to another symbol instead of a value. In the example above, if you changed line 8 in the *build.mac* file to read **putdef cycle !&newcyc**, line 5

in the **cplevel** macro would read **def cycle &newcyc**. Delayed symbol substitution such as this greatly increases the power of the program.

Some simple macro applications

Reading files

FELIX macros are the most powerful and innovative feature of this NMR processing software. By writing your own macros, you can automate anything that you can do with FELIX manually.

Your ability to transform and manipulate data automatically using macros is limited only by your imagination. For example, if you want to read a datafile (**rn**), apply an exponential window function (**em**), Fourier transform (**ft**), phase correct (**ph**), and draw a spectrum (**dr**), you could write the following macro and name it
process.mac:

```
1    c**process.mac
2        get 'Enter file name:' filnam
3        rn &filnam
4        em 3
5        ft
6        ph
7        dr
8        cl
9        ret
10        end
```

The **get** command prompts you with "**`Enter file name:`**" and waits for a datafile name to be input. The read command (**rn**) reads the file (notice the use of the symbol **filnam**). Then the data are multiplied by a 3 Hz exponential window (**em** 3). The data in the workspace is then Fourier transformed (**ft**), phase corrected (**ph**), and displayed on the current graphics terminal with the draw command (**dr**). The **ex &return** statement returns the macro to the GUI. The end command (**end**) is mandatory at the end of every macro; this tells FELIX to stop executing commands from the macro and accept input from you again. Instead of manually entering the six commands listed above, you can simply enter:

```
> .process
```

or:

```
> ex process
```

from the command line.

Plotting multiple files

Macros can be used to perform repetitive tasks by constructing loops that process several datafiles in a similar manner. For example, if you want to generate hardcopy plots for a series of related 1D spectra, you might construct a macro similar to the macro *mplot.mac*:

```
1    c**mplot.mac
2    again:
3        get 'Enter file name (quit):' filnam
4        if &filnam eqs quit then
5            ty Returning to &return
6            ex &return
7        else
8            rn &filnam
9            em 3
10            ft
11            ph
12            dr
13            hcp
14         eif
15         go again
```

As in the previous macro, you are first prompted for a filename, then the data are multiplied by an exponential, Fourier transformed, phased, and drawn. The spectrum on the screen is then plotted on the current hardcopy device with the hardcopy plot command (**hcp**). After the first spectrum is plotted, the *mplot.mac* macro branches back to the label again: and prompts for a second datafile, which is then processed and plotted. In this macro, to quit you enter **quit** instead of an actual filename.

5 Menus and Control Panels

The FELIX 2002 graphical user interface (GUI) contains a variety of interactive menu bars, pulldowns, and control panels that simplify data analysis and processing. Because the GUI consists of a set of FELIX macros that use specific commands to build and display graphical menus, you can easily modify it. To customize the GUI to meet the needs of your laboratory, edit the macros provided, or write your own.

The menu commands

Before developing your own menus, you first need to become familiar with the commands for menu interaction. The FELIX 2002 menu components can be divided into two separate categories: **menu items** and **control panels**. Menu components are described in detail in this section.

Menu items

FELIX 2002 menus contain items that, when selected, execute commands in macros. Menu-type items can be displayed as horizontal menu bars, vertical pulldowns, or icons. Menu items are selected by moving the cursor onto the desired item and clicking the primary mouse button.

Control panels

The second category of menu components includes control panels; i.e., **dialog boxes**. Control panels are separate windows that can be placed anywhere on the screen.

Changing the menubar interface

When FELIX 2002 starts, it looks for a file called *root.mot* to display items on the menu bar and pulldowns. The *.mot* file(s) must be present in one of the current menu directories (defined by the symbol **motpfx**). The syntax for a *.mot* file is:

```
mode item_name mnemonic accelerator accelerator_text
call_back call_back_data filename
```

Modes

mode can have different values depending on the type of menu being defined. A list of modes follows.

menubar

The **menubar** mode is for the items displayed on the menu bar itself. Here is an example of the use of the **menubar** mode:

```
menubar File F NULL NULL NULL NULL file.mot
```

where *File* is the name of the item that appears on the menu bar; *F* is the hot key that is used for the command (which is underlined on the menu bar); and *file.mot* is the name of the file that contains the items listed under the **File** pulldown.

popup

The **popup** mode is used when an item in the menu bar has a cascading menu attached to it. The following is an example for the use of this mode:

```
popup "Plottype" t NULL NULL "view Plottype" NULL
plottype.mot
```

where *"Plottype"* is the text that appears on the pulldown or cascading menu; *t* is the hot key; *"view Plottype"* is text that is given for this menu and is just for your reference (it is not used); and *plottype.mot* is the name of the file that contains the items to appear on the cascading menu.

separator

The **separator** mode draws a line to separate groups of related commands commands in a pulldown or cascading menu; for example:

```
separator separator NULL NULL NULL NULL NULL NULL
```

where *separator* is the text for the separator name. This can be any text you desire.

toggle

The **toggle** mode places a toggle into a pulldown or cascading menu; for example:

```
toggle "Draw Peaks" D Ctrl<Key>k Ctrl+k "ex draw-
peaks" NULL pksdrw
```

where *"Draw Peaks"* is the label for the toggle that appears on the menu; *D* is the hot key; *Ctrl<Key>k* is the accelerator that is bound to this menu item; *Ctrl+k* is the text that appears next to the text for the menu item; *"ex drawpeaks"* is the call-back macro that is called when this menu item is selected or when its accelerator or hot key is pressed; and *pksdrw* is the name of the symbol that is updated when the state of the toggle is changed.

item

The **item** mode is used when you want to execute the call-back without any cascading menus. It has the following general format:

```
item LABEL HOTKEY ACCELKEY CALLBACK CBCONTROL NULL DEPEND
```

where *LABEL* is the text that appears on the menu; *HOTKEY* is the hot key for this menu item; *ACCELKEY* is the accelerator; *CALLBACK* is the command that is called when this menu item is selected. CBCONTROL is a flag that controls how many times the callback is excuted if it is a table menu, and DEPEND is a symbol that controls whether this item is to be displayed (when DEPEND is not 0) or to be invisible (when DEPEND is 0).

The following is an example:

```
item "Plot" P Ctrl<Key>p Ctrl+p "ex plot" CBcontrol NULL status
```

keypad

The **keypad** mode is used to define actions that are performed when you press the keypad keys; for example:

```
keypad NULL NULL NULL NULL "exm keypad" NULL NULL
```

where *"exm keypad"* is the command that is called when you press the keypad buttons.

mouser

The **mouser** mode is used to define the popup menu (called *context* or *shortcut* menu) that appears when the right mouse button is pressed; for example:

```
mouser NULL NULL NULL NULL NULL NULL mouser.mot
```

where *mouser.mot* defines the menu items that comprise the right-mouse menu.

mousem

The **mousem** mode is used to define the action performed when you press the middle mouse button; for example:

```
mousem NULL NULL NULL NULL "exm cursorval 4" NULL
NULL
```

where *"exm cursorval 4"* is the call-back that is called when you press the middle mouse button when the cursor is inside a frame.

Changing the iconbar interface

The file used to display icons on the icon bar is *icons.mot*. The icon file(s) must be present in one of the current menu directories (defined by the symbols **icopfx**). The following is the syntax for the icons.mot file:

```
mode filename call_back ballon_help_text param1
param2 param3 param4
```

where *mode* can be one of the following.

icon

The **icon** mode is used to display the icons in the icon bar. ;. It has the following general format:

```
icon ICONFILE CALLBACK TOOLTIP NULL TOOLTEXT CBCONTROL NULL DEPEND
```

where ICONFILE is the icon file that contains the 16 x 15 pixel 16-color bitmap graphic for the icon; CALLBACK is the command that gets called when you select this icon; TOOLTEXT TOOLTEXT is the explanation of the command than appears on the status bar when the cursor is placed above the icon; CBCONTROL is a flag that controls how many times the command is excuted if it is a table menu, DEPEND is a symbol that controls whether this item is to be displayed (when DEPEND is not 0) or to be invisible (when DEPEND is 0).

The following is an example:

```
icon zoom.icon "ex limits 1" "Zoom" NULL "Zoom in onto a defined area"
CBcontrol NULL status.
```

separator

The **separator** mode is used to separate the icons; for example:

```
separator separator NULL NULL NULL NULL NULL NULL
```

Parameters 1-4 are currently not in use.

option

The **option** mode is used to place a combo box on the icon bar; for example:

```
option plottyp.mnu "ex plottype 0" "plottype" plot-
typ NULL NULL NULL
```

where *plottype.mnu* is the menu file that contains the text to display on the popup menu; *"ex plottype 0"* is the call-back that is called when this popup is activated; *"plottype"* is reserved for balloon help but is not currently displayed; *plottyp* is the symbol that is changed when you select a different item on the popup.

Control panels (dialog boxes)

There are many applications in NMR processing where you need to enter information, such as filenames and various processing parameters. For defining this information from within the FELIX 2002 GUI environment, you can create customized control panels (dialog boxes) from ASCII text files.

Control panels are separate windows. They can be placed anywhere on the screen. When opened, a control panel appears at the same coordinates where the previous control panel was placed.

Types of control panel commands

Within the file that defines a control panel are two types of commands:

- **Output commands**. These are commands that provide the graphical information needed to design the control panel. They define the size of the control panel, the text for the control panel header (*h), the text for user prompting (*c), and coordinates for displaying highlighted boxes (*x). These commands are for output only. For more specific information, see below.
- **Input commands**. These are commands for setting symbol values based on input from the user. These commands create edit (entry) boxes ("fields") (*f), toggles (*t), switches (*s), list boxes (*l), buttons (*b), combo boxes (*p), file-select tools (*o), list boxes (*v) with multiple items on a line (so-called variable lists), and slider-combos (*a). For more specific information, see below.

Working with control panel commands

There are two different control panels, *modal* and *modeless*. A modal control panel freezes the FELIX main window until you close the control panel. A modeless control panel, however, allows you to access the main window without closing the contro panel.

Modal control panels

To open a modal control panel, issue the following command statement with appropriate arguments:

```
mnu p menu x_origin y_origin
```

mnu p keywords in the statement tells FELIX to read a menu file containing a control-panel definition. The *menu* argument is the name of a menu file that must be present in one of the current menu directories (defined by **mnupfx**, **mnupf1**, and other symbols). By convention, menu files use the .mnu extension.

In previous versions of FELIX, *x_origin* and *y_origin* was used to define the position of the upper left corner of the box in character units. In FELIX 2002, this is ignored.

For example, the command line:

```
mnu p xopen 20 4
```

opens a control panel defined by a menu file *xopen.mnu*. See the following section for details about menu files.

A modal control panel is closed when you click one of its buttons, or the close button [x] on the upper-right corner of the panel.

Modeless control panels

To open a modeless control panel, issue the following command statement with appropriate arguments:

```
mnu a menu x_origin y_origin
```

A menu file that defines a modeless control panel has almost identical formats as used in a modal control panel, except for the following requirements:

- You must 'hold' the macro execution flow explicitly if you display a modeless control panel and do not want the macro execution to continue until the modeless control panel has been closed.

  An example is during 2D processing, when D1 has been Fourier tranformed and you choose to phase D1 interactively. The control panel for phasing is modeless; by default the macro would go on for D2 processing without waiting for you to finish the D1 phas-

ing. To hold the macro execution flow, you can add the following command in the macro next to the “mnu a” command:

```
wai -1 menu
```

This will force FELIX to hold the macro execution until the control panel defined by *menu* file has been closed.

- You normally want to define callback macros to the input controls, which trigger some actions when you interact with the controls in the control panel. See the Callback macros section below.
- You must close the modeless control panel explicitly. Unlike a modal control panel, clicking the buttons in a modeless control panel does not automatically close the panel. You must associate the buttons with a macro that closes the panel with the following command:

```
mnu remove menu
```

The macro file *ez1dtransform.mac* is a good example that allows you to display a modeless control panel for real-time phasing. The corresponding menu file is *rtphase.mnu*. Most of the input controls trigger a callback macro *rtphase.mac*. To find those files, see the following section **Find your way through the menu interface**.

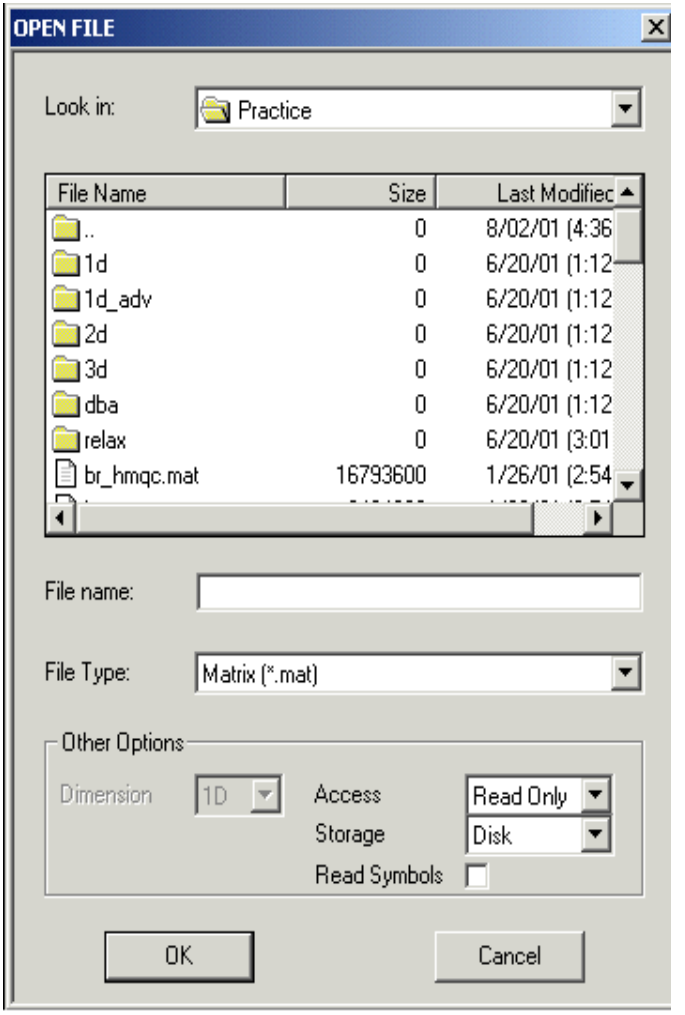
Control panel files (menu files)

A menu file defines the components in a control panel and their layout and actions to trigger. A full list of the file *xopen.mnu*, and the control panel it creates, is shown below.

```
c**xopen.mnu

23 40

*h 1 1 14 'OPEN FILE'

*o 2 2 40 5 5 &mytype&ftype &newpfx&ftype new-
fil(de:ftype.ge.0)

*c 3 19 5 'File Type'

*p 12 19 16 7 'Matrix (*.mat)' 'Felix Data (*.dat)'
'DBA (*.dba)' 'Insight II Molecule (.car)' 'PDB Mole-
cule (.pdb)' 'Other Data (Bruker, Varian, JEOL, FFW)
(*)' 'Macro (*.mac)' ftype
```

```
*c 3 20 5 'Dimension' (de:ftype.eq.1)
*p 12 20 8 2 '1D' 'ND' datdim (de:ftype.eq.1)
*c 20 20 5 'Access' (de:ftype.eq.0)
*c 20 21 5 'Storage' (de:ftype.eq.0)
*p 27 20 10 2 'Read Only' 'Write' macces
(de:ftype.eq.0)
*p 27 21 10 2 'Disk' 'Memory' matmem (de:ftype.eq.0)
*x 2 22 40 5
```

This file creates the following control panel:



The first line that starts with “c**” defines the name of the file and is optional. The second line defines the dimension of the control panel but it is ignored by FELIX 2002. Instead, the dimension of the control panel is automatically optimized by FELIX.

Each of the remaining lines defines a control or a group of controls in the control panel, and optionally the dependency, exit status, and

callback macro associated with the control. The formats for such commands are listed below.

> **Note:** Text after ";" in a line is treated as comment and is ignored by FELIX.

Output commands

These are commands that provide the graphical information needed to design the control panel.

Header text

The *h command in the control-panel file specifies the header text. The format of this command is:

```
*h x_origin y_origin number_of_characters 'header text'
```

The *header text* must be enclosed in single quotes. By default, the header is left-justified within the control panel and thus the parameters *x-origin* and *y-origin* are ignored. Header text may contain characters or use symbol substitution for customizing the header.

Labels

The *c command in a control-panel file is used to place text. Text in control panels is most commonly used for labeling the required input for entry boxes. (The command that defines an entry box is not coupled to the text that labels it.) The format of a character string command within a control-panel file is:

```
*c x_origin y_origin number_of_characters descriptor_text
```

Using the *c command, character strings can be placed throughout the control panel. Character strings are limited to 64 characters and must be enclosed in single quotes.

Highlight boxes

To enhance the appearance of control panels, you may uses highlighted rectangles. These rectangles are often used to group related

switches or entry boxes. Within a control panel definition, these rectangles are defined as:

```
*x x_origin y_origin width height text
```

The *x_origin* and *y_origin* parameters define the bottom left corner of the highlighted rectangle. The *width* and *height* parameters define the rectangle size in character units. *text* is an optional string to be placed on the left top line of the highlighted box.

Input commands

Input commands define information to be input to the program. These commands include functions for entering and updating symbol values, setting the value of the reserved symbol **button**, and choosing between switches and toggles.

Edit boxes

To generate an edit box for data input, the *f command is placed within the control-panel file. This has the format:

```
*f x_origin y_origin length type symbol
```

Again, the *x_origin* and *y_origin* values are used to place the box using character units. The *length* defines the length of the box in character units. The *type* defines what the edit box expects. A *c* means a character must be input into the edit box, an *i* means an integer, and an *r* means a real number.

The *symbol* can be either a reserved symbol name or a user symbol name. In either case, the current value of the symbol is displayed in the box. If that symbol has not yet been defined, then a new user symbol is created and is defined to have the value 0 or blank, depending on whether the field is numeric or character.

> **Note:** The box itself contains no prompting information. Hints to the user must be indicated separately using *c* commands as discussed above.

List boxes

A **list box** is an object that appears within control panels that lets the user select and scroll through values from a displayed list.

To display a list box the ***l** command is placed within the control-panel file. This command has the format:

```
*l x_origin y_origin width height list_file select symbol
```

The *x_origin* and *y_origin* values are used to define the lower left corner of the list box in character units. The *width* and *height* define the width and height of the list box in character units. The *list_file* is the name of the ASCII (text) file whose contents are to appear in the list box. *select* is the name of the symbol that defines which line is selected from the list. The value of *select* is 1 if the first line in the list is selected, 2 if the second line is selected, and so on. The *symbol* is the name of the symbol whose value is updated to reflect the selected line.

For example, if the selected value was *test.mac*, *symbol* will be *test.mac* after the "test.mac" line is selected from the list box. From list boxes, you therefore have two means of getting output: *select* defines the number of the line in the list box that was selected, and *symbol* represents its actual alphanumeric value. When exiting from a list box, either of these values can be used to make a later decision.

List boxes are especially useful for selecting files or matrices to be read and displayed. The text file named by *list_file* may be of virtually any form or origin. In many places in the menu interface, FELIX shows a list box of filenames. To do this, FELIX issues a command to obtain a directory listing and store it in a file, then shows the contents of that file in the list box. You could just as easily build a file using the **opn**, **put**, and **cls** commands.

Variable List Boxes

A **variable list box** is useful when you want to get only a certain word from the selected line in a list box. To display a variable list box the ***v** command is defined in the following format:

```
*v x_origin y_origin width height list_file symbol1 num_of_word symbol2
```

All the parameters are the same as for a regular list box (see the previous section), except that a value *num_of_word* is added between symbol1 and symbol2 to define which word from the select line is to be received by symbol2. For example, if *num_of_word* is **2**, the

selected line is “H104 N104”, then symbol2 received a value of “N104”.

Radio buttons (switches)

Radio buttons (or switches) are used within control panels for setting the values of symbols that integer numbers. By convention, a radio-button option is selected when its circle is highlighted. The command for building switches in control panels takes the format:

```
*s x_origin_1 y_origin_1 spacing # item1 ... itemN
symbol
```

The # parameter defines the number of radio buttons. The origins define the coordinate positions of the radio buttons. *symbol* defines the symbol for the switch value. If you select the first radio button, the symbol is **0**. Selecting the second radio button sets the symbol to one, and so on. The *spacing* parameter sets the vertical distance between the radio buttons. From an application point of view, symbol values set with switches are used within macros for defining branching points or parameter values. If that symbol has not yet been defined, then a new user symbol is created and defined to have a zero value. When the control panel is first drawn, the radio button corresponding to the current value of the *symbol* is highlighted. Only one radio button per switch can be highlighted. When another radio button is selected, the previous one loses the highlight and the new one becomes highlighted. The contents of *item1* ... *itemN* are the text prompts for the radio buttons.

Toggles

Toggles operate much like switches, but they can be only on or off, and take up less space within the control panel. The command for making toggles in control panels takes the format:

```
*t x_origin y_origin length # text_1 text_2 symbol
```

The origins are used to place the toggle using character units. The *length* is the width of the toggle in character units and is obsolete. Since the actual toggle does not contain any text, *text_1* and *text_2* are obsolete and can be blank. The # defines the number of toggle options, which is usually 2, since toggles can be only on or off. The *symbol* is the reserved or user symbol used for storing the current value of the toggle.

From an applications point of view, symbol values set with toggles are used within macros for defining branch points. In addition, toggles have applications for telling the user the current state of yes-or-no symbols. For example, the value for the **drwbox** symbol is either 0 or 1, depending on whether you want a box around your spectrum to be drawn.

When the control panel is first drawn, the toggle value corresponding to the current value of *symbol* is displayed. As with entry boxes, there is no text prompt outside the toggle. The ***c** command is often used to put text near the toggle to supply appropriate information.

Buttons

Buttons are similar to toggles and switches, in that they change symbol values. Buttons differ, however, in that they all change the reserved symbol called **button** and then exit a control panel. (A control panel that doesn't contain a button cannot be exited!) Buttons are usually used for branching within macro routines. The most common use of buttons is the **OK/Cancel** choice presented on many control panels.

The command for placing a button within a control panel is:

```
*b x_origin y_origin length text value
```

or

```
*bt x_origin y_origin length text value
```

The origins are used to place the button. The *length* defines the length of the button. The height of the button is 1.3 times of a character height if "*b" is issued, or 1.0 times of a character height if "*bt" is issued. The *text* is the button text, and the *value* defines the value of the **button** reserved symbol when that button is selected.

Buttons are usually used for branching within macro routines. The most common use of buttons is the **OK/Cancel** choice presented on many control panels. By convention, buttons that have *text* = **CANCEL** always set the *value* argument to zero, and buttons with *text* = **OK** always sey the *value* argument to one. When the value is set to zero, FELIX automatically ignores the changes made to the values of all symbols that are relevant to this control panel. In effect, when **Cancel** button is pressed, no change is made to the contents in the control panel.

Pulldowns (combo boxes)

Pulldowns (**combo boxes**) are very similar to list boxes, except that only the current selection is displayed, not all valid choices at once. Clicking a combo box with the cursor, opens a vertical list of all choices.

The command syntax is:

```
*p x_origin y_origin length num_options text_1 ... text_N symbol
num_lines_to_display
```

or, instead of listing all the options in the command, use a text file containing all the options:

```
*pf x_origin y_origin length height filename symbol symbol_line
num_lines_to_display
```

x_origin and *y_origin* set the left position of this box. *length* sets the width of the box. *num_options* is the total number of options. . *height* is height of the list box when expanded. *text_1 ... text_N* are the option lines, each enclosed by single quotation marks. *Filename* is the name of the text file that contains the options. *symbol* is the symbol that receives the sequential number of the selected option: 0 for the first line, 1 for the second and so on. *symbol_line* is the symbol that receives the character string of the selected option. *num_lines_to_display* is an optional value that sets the number of the first options to be displayed. By default all options are displayed. If *num_lines_to_display* is smaller than *num_options*, only the the *num_lines_to_display* options are displayed. Sometimes this is useful when the number of options to display has to be determined in runtime.

File-open combo

File-open combo is a special combination of combo box, list and edit boxes that makes it easier for you to browse and select a folder and / or a file.

> **Note:** Only one file-open combo is allowed in a control panel. Extra ones are ignored

The command syntax is:

```
*o x_origin y_origin width height #_lines extension dir filename
```

x_origin and *y_origin* set the position of this tool in the control panel. The *width* and *height* set the width and height of the tool. *#_lines* is an obsolete parameter and is ignored. The height determines the number of files to display inside the list box contained within the file-open tool. *extension* sets the default type of files to be searched for in the directory. *dir* is the directory to be searched, and *filename* is the name of the file that is returned when you select a file from the list.

> **Note:** In the previous versions of FELIX, file-open combo included **OK**, **Cancel** and **Filter** buttons. FELIX 2002 in contrast assumes that you will add them explicitly (using the *b commands). This allows you more controls over the layout of the control panel. However, if a file-open combo exists without **OK** and **Cancel** buttons explicitly defined, FELIX 2002 will automatically add them at the bottom of the control panel.

Slider combo

A **slider combo** comes with a horizontal (or vertical) adjustable "value bar" plus three edit boxes that displays the minimum, maximum and current values. A slider combo is usually used in a modeless control panel for real-time display of a spectrum while interactively adjusting the values of one or more symbols.

A slider combo is created using the following command:

```
*a x_origin y_origin length minimum maximum decimal orientation drag_mode
symbol label edit_width
```

x_origin and *y_origin* set the upper left position of the slider in character units, and *length* defines the length of the slider in character units. *minimum* and *maximum* define the minimum and maximum values of the slider. The absolute value of *decimal* tells FELIX how many decimal places to display for the slider position. If *decimal* > 0, a normal float number is displayed. If decimal < 0, scientific notation is used to display the number. *orientation* determines the orientation of the slider. When the *orientation* value is **1**, the slider is drawn horizontally; otherwise it is drawn vertically. (For FELIX 2002 only horizontal slider can be displayed). *drag_mode* tells the program when to call the call back macro if provided (see following section for details about callback macros): **0** means to call the callback macro only when the mouse button is released, otherwise the callback macro is called whenever the value of the slider changes. *symbol* is

the symbol that receives the slider value. The *label* is the text that appears above the slider.

edit_width is an optional parameter that defines the width of the three edit boxes in character units. The default value is six. You can change the minimum, maximum or current value by directly typing in the corresponding edit box followed by <Enter>, and the slider shifts to reflect your changes.

Control-panel dependencies

All the components in a control panel can be deactivated (grayed out) depending on a situation defined by you. For example, the selectability of any entry box in a control panel can be made dependent on the value of another symbol. By default, an entry box is always selectable. However, you can define a dependency for the box, so that it is selectable and editable only when some symbol value meets a specific criterion. For example, the entry box for the name of an annotation file can be made dependent on the symbol that controls the drawing of annotations, so that the user can change the annotation filename only when annotation-drawing is turned on.

An example helps to illustrate the usage and syntax. The following two lines come from the FELIX 2002 macro that specifies the 1D PLOT GENERAL PARAMETERS control panel. They control automatic drawing of annotations in the current plot:

```
*t 20 9 12 2 'No' 'Yes' pltann

*f 20 10 12 c annfil (de:pltann.eq.1)
```

The first line defines a yes-or-no toggle to control the symbol **pltann**, the reserved symbol for drawing annotations on plots. The second line defines an entry box where the user can enter the name of the annotation file into the symbol **annfil**. The last portion of the second line contains the dependency formula, which states that this box is selectable only when **pltann** is **1**; that is, when the toggle is turned on.

Another example from the same control-panel macro controls the selectability of the entry box for the 1D peak entity:

```
*f 20 6 12 c picent (de:drwpks.gt.0)
```

This dependency states that the 1D peak entity name is selectable only when **drwpks** is greater than zero.

The formal syntax for specifying entry-box dependency requires the following:

- The parentheses and the leading **de:** are required and define this parameter to be a dependency descriptor.
- The rest of the descriptor must be of the form *symbol.relation.value*. This specifies the name of one symbol, one of the arithmetic relations (for example, eq, ne, lt, gt, le, or ge), and one numeric value.
- The entry box is selectable only when the stated relation is true.
- An entry box can have many dependency descriptors, which are separated by Boolean (and / or) relationships, for example:

```
(de:usedef.eq.0.or.method.gt.1)
```

Immediate exits

In addition to exiting a control panel via the **Cancel** or **OK** button, you may also trigger an immediate exit from a control panel when a pulldown is selected. This is useful for taking immediate action as soon as a pulldown is selected, without having to wait until the user clicks a button.

Another example from the same control-panel macro illustrates immediate exits via a pulldown:

```
   *p 20 3 12 6 'None' 'Points' 'Hertz' 'PPM' 'Sec-
onds' '1/cm' axtype (ex:2)
```

This exit descriptor states that the control panel should immediately exit with **button** set to **2** whenever the symbol **axtype** is changed by this pulldown.

The formal syntax for immediate exit requires that the parentheses and the leading **ex:** be present and define this parameter as an exit descriptor. The rest of the descriptor must be of the form: *value*. This specifies the numeric value to be given to the **button** symbol and tells the macro that invoked the control panel the precise exit method that was activated.

Callback macro

Sometimes you may want the change in a control to activate the execution of a certain macro immediately. This is especially useful when you display a modeless control panel containing a slider control, and you want to redraw the spectrum whenever you move the slider.

For example, the control panel for 1D real-time phasing has the following command that defines a slider for you to adjust the 0-order phase parameter:

```
   *a 1 2 40 rtpmin0 rtpmax0 &rtpdec0 0 1 rtp0 '0
Order' "exm rtphase 1"
```

The last parameter "exm rtphase 1" forces the macro *rtphase.mac* to be executed with a parameter "1" whenever the you move the slider.

Finding your way through menu interface files

As mentioned, the FELIX 2002 GUI is built from macros, with the assistance of menus and control panels. Every facet of the functioning of this interface is explicitly contained in these ASCII text files. This allows the GUI to be altered and bugs to be fixed, without the need for a new version of FELIX.

However, this feature necessarily involves a huge number of files. There are literally hundreds of macros, menus, and control-panel files. This section is designed to help you find your way through this maze of files.

FELIX 2002 has a reserved symbol that facilitates the generation of diagnostic information about every macro, menu, and control panel invoked while the program is being run. This reserved symbol (**verify**) has three acceptable values (0, 1, or 2), which specify the type of information to be provided.

When **verify** is zero (the default value), no verification takes place and FELIX runs normally. If you select the menu item **Preference/ Macro Debug**, and next check the option of **Show macro names**, **verify** is set to **1**. Then, every time a menu interface file (macro, menu, or control panel) is invoked, the full filename is shown in the

output window. This shows you exactly what files are being accessed for every operation that is selected from the interface.

If you select the menu item **Preference/Macro Debug**, and next check the option of **Show all details**, this sets **verify** to **2**. FELIX gives even more diagnostic information. In addition to the filename, each FCL command is printed in the output window as it is about to be executed. This allows advanced users to see the complete literal text of each command that FELIX performs and is useful for debugging new macros or altered menu-interface macros. In fact, if you choose the "parent text" frame configuration and redirect FELIX output to a file, you can capture the actual stream of FELIX commands that are executed to a file, and then use that file to construct a new macro. However, be aware that this activity can slow down command execution, especially if a macro containing long command lines is used.

6 The Database and Tables

The database (DBA) facility in FELIX 2002 is an information storage and retrieval subsystem used for organization of and access to information related to spectrum data. For 1D applications, the database might contain information pertaining to baseline points, integral segment definitions, integral volume measurements, resonance positions, and line widths. For 2D applications, the database might contain information related to cross-peak positions, cross-peak widths, volumes, resonance assignments, and tile limits. The database is flexible enough that it can be used with any type of information, whether or not it relates to NMR data.

All data are contained in a single binary file called a database file. Since this file is binary, you cannot examine it directly. There are, several tools available for accessing the database information from within FELIX and for generating useful ASCII output. Database-access commands also allow writing of any entity in the database to an ASCII file and reading of the ASCII files back into the binary database.

Structure of the database

Each set of information is stored in the database as an individual *entity*. An entity can be thought of as a folder or container that holds zero or more units of a particular type of data. An *item* is one unit of data within an entity. All items in an entity have a single format that is defined in an auxiliary file called a *schema*. For example, the set of information that corresponds to one particular cross peak is stored in the database as one item, within an entity that may itself contain many cross peaks. Likewise, assignment information for one specific assignment might be stored in the database as a single item within an entity of multiple assignments.

An item itself is composed of several data *elements*. Each individual datum within an item is an element. Thus, one cross-peak item con-

tains center and width elements for all dimensions, and one assignment item contains atom-name and chemical-shift elements. The element is the smallest unit of information in the database. An element is always one integer, one floating-point value, or one ASCII string. Element names and types within an item are defined by the corresponding schema file when an entity is built.

Entities are called *Tables* throughout the FELIX 2002 graphical user interface (GUI). The two terms may be used interchangeably for the most part. See also "Spreadsheet interface."

The database schema

The **schema** file is the format descriptor for an entity. It tells how many elements that an item of this entity must have, as well as the internal and external format of each element. Schema are stored in the **schema** subdirectory and are prefixed with the **schpfx** symbol.

Schema files are format sensitive. If you want to build your own schema file, the order of the format descriptions must be the same as in the schema files provided. Building your own schema files provides a mechanism to store and output database information in any format you desire.

An example schema for storing 1D peaks (the line numbers are for explanation purposes only) is:

```
1   c**asg.sch

2   asg    5

3   item   i k 01 06 (1x,i5)

4   cenpnt r k 01 10 (f10.5)

5   wid     r   01 10 (f10.5)

6   cenppm r k 01 10 (f10.5)

7   name   c   08 10 (2x,a8)
```

where:

Line 1:

A filename record, of the form **c****, ***filename*** (a3,a)

Line 2:

A descriptor record, of the form *schema_name, number_of_elements* (a8,i2). The number of elements is calculated for one occurrence of each element.

Line 3:

The first element. It must be **item**, as this is the primary access key for database items. Always set the key flag for the **item** element by including the letter "k" after the type identifier. Subsequent elements can be named at your discretion. The formatting scheme for the element names and all subsequent parameters must be:

```
(a8,1x,a1,1x,a1,i3,i3,1x,a32)
```

The definition for each element in the schema is:

```
element type key internal_size external_size I/O_format
```

where:

element	(a8) element name
type	(a1) **r** = real **i** = integer **c** = ASCII
key	(a1) **k** = key on this element for sorting
internal size	(i3) binary data size (words)
external size	(i3) size of ASCII text of data
I/O format	(a32) FORTRAN format of element

In this example, all items within this entity would contain:

```
item, cenpnt, wid, cenppm, name
```

with a format of the form:

```
(1x, i5, f10.5, f10.5, f10.5, 2x, a8)
```

An example of the schema used for picking and storing multi-dimensional data is:

```
1   c**xpk.sch
2   xpk    6
3   item   i k 01 06 (1x,i5)
4   #begin
5   cen#   r k 01 10 (f10.3)
6   wid#   r   01 10 (f10.3)
7   ptr#   r   01 06 (1x,i5)
8   asg#   r   16 18 (2x,a16)
9   #end
10  cc      r   01 06 (1x,f5.2)
```

In this instance, the number of elements stored in the database depends on the dimensionality of the matrix. For 2D matrices, two center position elements are created, named *cen1* and *cen2*. This applies to all the elements that fall between the #begin and #end statements. Similarly, for 4D data, positions for four assignment names (*asg1*, *asg2*, *asg3*, *asg4*) are created.

When building a new user-defined entity, the value of *occur* in the build string is used to determine the number of occurrences in a receptive entity (see below). You encounter this option when you build an entity from scratch:

```
dba entity build entity_name schema occur
```

The **dba entity build** command creates one new empty entity called *entity_name*. This new entity stores information in the format defined by the *schema* file and is repeated the number of times set by *occur*. More information on this procedure is found below.

Creating database information

Certain program commands extract spectrum information and it directly into a database file. These commands look at the current values of specific symbols to determine which files are current for that application. For this reason, opening a database file must precede execution of these commands. For example, the **drx**, **pic**, and **vol** commands need to access peak information before drawing peak boxes, picking more peaks, and generating volumes. Since this information is stored in the database file, this file must be open and current. By default, the commands that store spectrum information in the database automatically build their own entities. For example, the **pic** command uses the value of the reserved symbol **pksent** to determine where to put the subsequently picked peaks. Likewise, the **vol** command uses the value of the reserved symbol **volent** to store *N*D volume information in the entity.

Database command structure

Since the database facility is fairly sophisticated, we have developed a database command mode separate from the FELIX command language. To enter database mode, enter:

```
dba
```

Entering this command changes the command prompt to **DBA>**, indicating that only database subcommands may be entered. The subcommands and their required parameters may be entered on a single line (128 characters maximum).

After executing the command statement, the **DBA>** prompt reappears, waiting for the next subcommand. To return to the FELIX prompt, enter one of the database subcommands: **bye**, **exit**, or **quit**. Alternatively, you can place an entire database command statement (starting with **dba**) on one line. After execution of the **dba** command, the program returns to the FELIX prompt.

Basic dba subcommands

In summary, the basic **dba** subcommands are:

dba

Activate database command mode.

dba bye

Exit database command mode.

dba exit

Exit database command mode.

dba quit

Exit database command mode.

Building database files

To build a new database file from scratch, you must issue the appropriate command for constructing a new file:

```
dba file build file_name confirm
```

The *file_name* parameter names the file with the default .dba extension. This file is placed in the directory defined by the **dbapfx** reserved symbol.

The *confirm* parameter is used to overwrite a database file that already exists. A "y" overwrites a database file with the same name automatically, and an "n" does not overwrite the database file (default).

Once a database file is built, you must explicitly open it:

```
dba file open file_name
```

Once the database file is opened, you can subsequently pick peaks, integrate volumes, make assignments, or otherwise access spectrum information.

To save the database file contents you must explicitly close the file:

```
dba file close confirm file_name
```

The *confirm* parameter requires the values **s** or **q**. The **s** option saves the current database file contents to disk with all changes; **q** discards all the changes you've made to the database since the database file was first opened. To change between databases, you must first close the current database file before opening or building a new one. The *file_name* parameter is used for saving the DBA to another filename and is optional.

To list the directory (contents) of the current database file, use the **show** command:

```
dba file show
```

To determine whether the contents of a database have been modified, use the **mod** command:

```
dba file mod symbol
```

If the database has been modified since it was last saved, the resultant value of *symbol* is **1**. If the database has not been modified, the value of *symbol* is **0**.

To write out all database entities into a set of ASCII files, use the command:

```
dba ascii dir_name
```

This command writes out all entities into the *dir_name* directory.

From this directory you can restore a database using the command:

```
dba restore dir_name
```

This destroys all entities in the current database, so it should be used only with a clean database.

dba file subcommands

In review, the **dba file** subcommands are:

dba file open *file_name*

Open database file

dba file close *confirm file_name*

Close database file

dba file delete *confirm*

Delete database file

dba file build *file_name confirm*

Build database file

dba file show

Database directory

dba file mod *symbol*

Database modified query

dba file ascii *dir_name*

Write ASCII files

dba file restore *dir_name*

Restore database

Database entities

As mentioned above, a *schema* file, which is created when an entity is built, defines the format for all items in an entity. The schema thus functions as the item template for an entity: it contains the element names, data types, and ASCII text format that precisely define the contents and layout of every item in that entity. Schema are explained in more detail in "The database schema."

Building an entity

You can build an entity for storing spectrum information using the command:

```
dba entity build entity_name schema occur
```

The **dba entity build** command creates one new empty entity called *entity_name*. This new entity stores information in the format defined by the *schema* file and repeats it the number of times set by *occur*. For example, if you want to build an entity for storing NOE buildup intensities at 6 mixing times, the format for a single intensity could be defined in the schema, and the intensity field would be repeated *occur* (i.e., six) times.

Entities can be deleted from the database using the command:

```
dba entity delete entity_name
```

To list the contents of an entity on the screen, use the **show** command:

```
dba entity show entity_name
```

This prints the contents of the specified entity on the screen in the format defined by the schema for that entity.

To generate an ASCII file of an entity for output, editing, or analysis, use the command:

```
dba entity write entity file_name
```

The first line of this file is a copy of the filename. The next line is the schema name and number of occurrences. The other lines in the file are the data contents of the entity written in the defined schema format.

To read the ASCII file of an entity that you have modified back into the program, use the command:

```
dba entity read entity file_name
```

This command reads a file where the first line is a filename. The next line after the filename must be the schema name and number of occurrences. The remaining lines of the ASCII file are read, as items to be stored in this new entity, using that schema format. This feature is most useful for editing existing entity information rather than for building new information from scratch. Therefore, we suggest that you use the **dba entity read** command in conjunction with the **dba entity write** command.

In some instances, you may want to read items of one entity while directly altering another entity. The command:

```
dba entity current entity name
```

addresses this need by presetting the item buffer to one entity without specifically loading an item from that entity. See below for more information about the item buffer.

Entities can be copied to a new entity based on the same schema with the command:

```
dba entity copy old_entity new_entity renumber
```

When *renumber* is zero, all item numbers remain the same. When *renumber* is non-zero, all items are renumbered so that the first item is number *renumber*, and all other item numbers increment by one.

Converting an entity

An entity can be converted to a new entity based on a slightly different schema with the command:

```
dba entity filter old_entity new_entity new_schema constants
```

For every item in the entity, all elements that match name for name between the old schema and the new schema are copied. Any elements of the new schema that are not present in the old schema may be filled in with a constant value by including *constant* parameters on the command line in the form ***element=value***.

Selected ranges of elements for every item can be put into a file with the command:

```
dba entity put entity file_name first_elm last_elm
```

For every item in the entity, a line is written to the file containing the values of the elements in the range from *first_elm* to *last_elm*, inclusive. Both *first_elm* and *last_elm* are integer element numbers. Optionally, if the *file_name* is entered as

"#", then the lines are appended to the current FELIX output file. See the descriptions of the **opn** and **put** commands in Appendix A, *Command Reference*, for more information about output files.

The schema file that an entity is built from is available with the command:

```
dba entity schema entity symbol
```

This sets the value of *symbol* to be the name of the schema file, without the prefix or .sch extension.

The spreadsheet interface to the database is described below.

Integrity testing an entity

The database also provides a command for integrity testing an entity. All entities have special indexing trees for all keyed elements (elements that have a "k" in the schema file). These index trees provide fast search of and access to items in the entity. The entity can be tested for integrity with the command:

```
dba entity test entity
```

An entity must pass this test. There is no way to recover an entity that fails. However, if an entity does fail, it is usually only on some of the keyed elements and not on all elements. In most cases, you can still write the entity to a file (**dba entity write**) and then read the file back into another entity (**dba entity read**) and the entity integrity will be restored. This command is for debugging purposes only.

dba entity subcommands

In review, the database entity subcommands are:

dba entity current *entity_name*

Activate entity

dba entity read *entity file_name*

Read ASCII file

dba entity write *entity_name*

Write ASCII file

dba entity delete *entity_name*

Delete entity

dba entity build *entity schema occur*

Build new entity

dba entity show

Show entity contents

dba entity copy *old_entity new_entity renumber*

Copy entity

dba entity filter *old_entity new_entity new_schema constants*

Filter entity

dba entity put *entity file_name first_elm last_elm*

Put entity to file

dba entity *schema entity symbol*

Get schema file name

dba entity test *entity*

Test entity integrity

Database items and elements

An **item** is defined as one unit of data within an entity. An item itself is composed of several data **elements**. The individual datum within an item is an *element*.

The element is the smallest unit of information in the database. An element is always one integer, one floating-point value, or one ASCII string. Element names and types within an item are defined by the schema file when an entity is built. The only restriction is that the first element within an item always be an integer with an element name of **item**.

item buffer

Since items can contain a number of elements (up to 20 total) of several types (integer, floating point, ASCII), each with its own unique element name, a simple and universal item interface is essential.

To this end, the database command interface provides one generic buffer and a collection of item and element subcommands for accessing this buffer. These subcommands allow you to move data values into and out of any element of any item of any entity. The item subcommands move items from an entity into the item buffer and store items back in their place in the entity. The element subcommands move individual elements from the item buffer into symbols, then store new values back into elements in the item buffer.

To load an item into the item buffer, use the command:

```
dba item load entity.item#
```

The *item#* is the identifying marker for that item and is simply the item element.

To load any element from the item buffer into a symbol, use the command:

```
dba element load element symbol
```

This loads the value of the specified *element* into the *symbol* you've named.

To store information back into the database file, you must first store the value of an element in the item buffer, which must then be stored in an item in the entity. For example:

```
> dba element store element value
```

stores the current *value* in the specified *element* in the item buffer.

To store the item buffer back in your database file, the above command would be followed by:

```
> dba item store entity.item#
```

Database items can also be deleted from an entity using the command:

```
dba item delete entity.item#
```

Only the specified item is affected. No renumbering of other items occurs. This command is useful for permanently removing spectrum information from the database file.

To show the current contents of an item, use:

```
dba item show entity.item#
```

The **dba item show** command prints the contents of one item of one entity. To show the contents of the item buffer itself, you can use **dba item show 0** (the item buffer is item number 0 by default). Likewise, the command:

```
dba element show entity.item#.element
```

shows the contents of the specified element to the screen.

In conclusion, to use the **dba item** and **dba element** commands to change the current value of an element within an item, you follow these steps:

1. Load the desired item from the entity to the item buffer.
2. Load the desired element into a symbol.
3. Store new values in an element.
4. Store the item back in the database.

Alternatively, to create new items you would:

1. Build a new entity from a schema.
2. Make the new entity current.
3. Store values in all elements.
4. Store the item buffer to any one item.

Element storing shortcuts

The FELIX 2002 database also has some shortcut methods for loading and storing elements. One method allows you to bypass the use of the item buffer and get values directly from the entity. This is most useful if you need only one element value from an item. The command:

```
dba element load entity.item#.element symbol
```

loads one element directly from the entity into a symbol. This is equivalent to the two commands:

```
dba item load entity.item#
```

```
dba element load element symbol
```

Likewise, to store a value to a single element in an item, the command:

```
dba element store entity.item#.element value
```

is equivalent to the three commands:

```
dba item load entity.item#
```

```
dba element store element value

dba item store entity.item#
```

The second shortcut method allows loading or storing adjacent elements in an item with a single command. The elements ***must be*** adjacent. With the **dba element load** command, if the command line has multiple *symbol* parameters, then multiple elements are loaded to those symbols. Likewise, with the **dba element store** command, if the command line has multiple *value* parameters, the multiple values are stored to those elements. In the item buffer method, the command format is:

```
dba element load element_1 symbol_1 ... symbol_N

dba element store element_1 value_1 ... value_N
```

while for the entity direct method, the command format is:

```
dba element load entity.item#.element_1 symbol_1 ... symbol_
N

dba element store entity.item#.element_1 value_1 ... value_N
```

These different access methods provide the fastest possible runtime access to the database contents. The basic guidelines for speediest access are:

1. To load multiple elements and then store one or more elements, use the item buffer.
2. To load just one element, use the entity direct method.
3. To load or store multiple adjacent elements, use the multiple parameters method.

dba item and element subcommands

In review, the **dba item** and **dba element** subcommands are:

dba item load *entity.item*

Load an item to the item buffer

dba item store *entity.item*

Store item buffer to an item in the entity

dba item show *entity.item*

Print contents of an item

dba item delete *entity.item*

Delete an item from an entity

dba element load *element symbol*

Load an element from the item buffer

dba element load *entity.item.element symbol*

Load an element from an entity to a symbol

dba element store *element value*

Store a value into an element of the item buffer

dba element store *entity.item.element value*

Store a value into an element of an entity

Database item lists

The FELIX 2002 database also has utilities for fast read-only perusal of entities. In addition to accessing explicit items one by one via the item buffer, you can also generate and manipulate complete lists of item numbers.

Item lists

Because these lists occupy the 1D buffers, the buffers may require configuring to contain large lists. These lists know only how to hold *pointers* to items in an entity and do not actually store data-items themselves. In using these pointers to access spectrum information, you must still specify specific items and elements using the commands described above.

The purpose of item lists is to help maximize the speed of database queries. Instead of explicitly loading every item one at a time and testing various elements for a condition to see if that item meets the criterion, the item list commands generate the set of items that meet a criterion in a single step. You can then work with just those items in the list, knowing that they satisfy the criterion. Because you can avoid looking at every item explicitly, you save considerable time.

As an example, locate a range of items within a database. This is accomplished using the following **dba list** command sequence:

```
dba list range list # entity element low high symbol
```

More specifically, the command statement:

```
dba list range 1 xpk:peaks cen1 200 500 pknum
```

puts the item numbers of all of the cross-peak footprints from the entity xpk:peaks, whose centers in D1 (cen1) are between 200 and 500 point units, into list "1". The number of items that meet this criterion is returned in the symbol **pknum**. You can then use **dba list show *list_#*** to view the resulting list. For example, if seven items meet the above criterion, list 1 might appear as:

```
slot #=      1      item#=4
slot #=      2      item#=309
slot #=      3      item#=17
slot #=      4      item#=90
slot #=      5      item#=8
slot #=      6      item#=5
slot #=      7      item#=99
```

In this list, each slot number contains an item number. For use within a macro, the following sequence prints the D1 centers on the screen, one per line:

```
1   dba list range 1 xpk:peaks cen1 200 500 pknum
```

```
2    for loop 1 &pknum
3            dba list load 1 &loop tstitm
4            dba element load xpk:peaks.&tstitm.cen1 cenval
5            ty peak=&tstitm D1 center=&cenval
6    next
```

The first line creates a list of cross peaks, with centers between points 200 and 500 in the first dimension. The second line sets up a loop to move through each item in the list. Because **pknum** number of peaks were found using the **dba list range** command, the loop executes **pknum** cycles.The third line loads each item from the list, saving the item number as the symbol **tstitm**. The fourth line loads the value of an element within the selected item to the symbol **cenval**. Line five prints the value of the symbol **cenval**, and line six exits the **for** loop. Using this logic, values for any element in the database, or even database items themselves, can be modified.

dba list subcommands

The dba list subcommands are given below, with brief descriptions of their functions.

dba list add1 list value

This command appends one value (item #) to the end of a list. If the value is already in the list, the item # is not added again.

dba list count list symbol

This command loads the count (number of item #'s) of a list into a FELIX symbol.

dba list del1 list value

This command deletes one value (item #) from a list. If the value is not found in the list, no deletion occurs. Note that delete looks at the value in a slot, not at slot numbers.

dba list difference list1 list2 list3

This command builds a new list that is the set difference of two lists. The resulting list number may be one of the input lists. The result *list3* contains only those item numbers from *list1* that are not present in *list2*.

dba list entity list symbol

This command extracts the entity ID number from the given list. The ID number is the same value as returned by the **db$entity()** function.

dba list find list value symbol

This command finds one value (item #) in a list and puts the result into a symbol. If the value is not found in the list, the result is zero. When the value (item #) is found, the result is the slot number containing that item #.

dba list intersect list1 list2 list3

This command builds a new list that is the set intersection of two lists. The result *list3* may be one of the input lists. The result *list3* only contains those item numbers that were in both *list1* and *list2*.

dba list load list slot symbol

This command loads the current contents of one slot of one list (an item#) into a FELIX symbol.

dba list match list entity element match_string symbol

This command builds a list of all items where the given element matches a wildcard string. This command is only applicable for character-string elements. The normal rules for using wildcards in the match string apply. The number of items placed in the list is returned in *symbol*.

dba list exact list entity element match_string symbol

This command builds a list of all items where the given element matches a string taken literally, that is, the * is not interpreted as a wildcard but as a letter. This command is only applicable for character-string elements. The number of items placed in the list is returned in *symbol*.

dba list move list1 list2

This command copies *list1* to *list2*.

dba list range list entity element low high symbol

This command builds a list of item numbers in a range based on the values for a single element. You specify an entity name to search, an element name to select, and the desired range values. The result is a list, built to contain all the item #'s having that element within the requested range. The item numbers are ordered such that the element values are in increasing order. The number of items placed in the list is returned in *symbol*.

dba list show list

This command shows the contents of one list.

dba list sort list key up/down

This command sorts the contents of one list. The basic sort is limited to elements containing numeric fields. The list may be sorted in ascending or descending order. Item numbers are rearranged in the list so that they occur in the sorted order.

You can sort on a key other than the item number in two ways. First, you can sort a list of item #'s, keying on any numeric element in that item. In this method, the item numbers are rearranged so that the values of the key element occur in the sorted order.

The second way to sort a database list involves the 1D workspace WORK. For a list with *N* item numbers, load the first *N* points of WORK with real values, one for each item# (that is, WORK(i) = function (item# at LIST(i))). Instead of

an element number or name for the sort key, use the name "WORK" as the key. The item numbers (and the contents of WORK) are rearranged so that the values in WORK occur in sorted order. This lets you order the item numbers in a database list based on any criterion that you can quantify.

dba list union list1 list2 list3

This command builds a list that is the set union of two lists. The resulting *list3* may be one of the input lists. *list3* contains the item numbers that were in either *list1* or *list2*.

dba list zero list entity

This command zeros one list by setting the list count to zero and pre-setting the entity owner of this list. A list must be zeroed before you can build a new list from scratch using the **dba list add1** command. The new list needs an entity owner to keep track of which dataset the items of this list belong to.

dba list parameters

The **dba list** parameters are:

element	name of one element in this entity
entity	name of one entity (owner of this list)
high	high value for setting range limits
key	element number or name, as the key to sort on
list	number of one list [range 1 ... n_buffers]
low	low value for setting range limits
match_string	wildcard ASCII string to match against
slot	number of one list slot [range 1 ... list size]
symbol	symbol loaded from contents of slot in list
up/down	sort order: **0**=ascending, **1**=descending
valu	value to add to or delete from a list

Creating lists of information

The database list subcommands discussed above are specifically designed to let you manipulate arrays of spectrum information. In addition, the database list subcommands let you build lists from scratch using the **dba list add1** and **dba list range** commands.

For certain applications, however, it is more efficient to provide specific commands for building lists automatically. The **xpl** command generates a database list of item numbers from cross-peak entities based on the options listed below. These all function as described below and are equivalent to a series of **dba list** commands, but they operate much more quickly.

Please refer to the **xpl** command in Appendix A, *Command Reference*, for more information.

xpl subcommands

xpl box *list entity d1low d1hi d2low d2hi ... symbol*

This command makes a list of all cross peaks inside an *N*D box. All peaks having center-point values inside the box limits are put in the list. The box may reference outside the current plot limits. The box limits are in points. When *d1low* is negative, no other box limits are needed. Instead, box limits are defined by using the spot cursor to draw a rectangle around a region of the current *N*D plot. The value placed in *symbol* represents the number of items placed in the list.

xpl touch_box *list entity d1low d1hi d2low d2hi ... symbol*

This command makes a list of all cross peaks inside an *N*D box. All peaks having at least part of the peakbox touching the box limits are put in the list. The box may reference outside the current plot limits. The box limits are in points. When *d1low* is negative, no other box limits are needed. Instead, box limits are defined by using the spot cursor to draw a rectangle around a region of the current *N*D plot. The value placed in *symbol* represents the number of items placed in the list.

xpl inside_box *list entity d1low d1hi d2low d2hi ... symbol*

This command makes a list of all cross peaks inside an *N*D box. All peaks having their full peakbox inside the box limits are put in the list. The box may reference outside the current plot limits. The box limits are in points. When *d1low* is negative, no other box limits are needed. Instead, box limits are defined by using the spot cursor to draw a rectangle around a region of the current *N*D plot. The value placed in *symbol* represents the number of items placed in the list.

xpl line *list entity dimension point symbol*

This makes a list of cross peaks touched by a line in one dimension. All cross peaks in the entity are candidates because plot limits have no effect. The value placed in *symbol* represents the number of items placed in the list.

xpl point *list entity d1point d2point ... symbol*

This command finds all cross peaks touching an *N*D point. All peaks with their footprints (center plus width) encompassing the given *N*D point are put in the list. The point may reference outside the current plot limits. The *N*D point is given in points. When *d1point* is negative, no other dimensional points are given, instead the point is selected using the cross-hair pointer. The value placed in *symbol* represents the number of items placed in the list.

xpl range *list entity dimension lowpt hipt symbol*

This command makes a list of all peaks within a range in the specified dimension. All cross peaks in the entity are candidates, and currently displayed plot

limits have no effect. The value placed in *symbol* represents the number of items placed in the list.

xpl name *list entity dimension match string symbol*

This command makes a list of all peaks, with an assignment name in that dimension that matches the given character string. The value placed in *symbol* represents the number of items placed in the list.

xpl frequency *list entity dimension freq_list# resolution symbol*

This command makes a list of all peaks that align with a frequency list in that dimension within the given *resolution*. The value placed in *symbol* represents the number of items placed in the list.

xpl pattern *list entity dimension pattern_id resolution spectrum_id symbol*

This command makes a list of all peaks, with position in that dimension, that align within the *resolution* with the given pattern's (*pattern_id*) frequencies from the **Assign** database. The generic or spectrum-specific shifts of the frequencies are used, depending on the *spectrum_id* parameter. The value placed in *symbol* represents the number of items placed in the list.

xpl clipboard *list entity dimension resolution symbol*

This command makes a list of all peaks, with position in that dimension, that align within the *resolution* with the frequencies in the clipboard from the **Assign** database. The value placed in *symbol* represents the number of items placed in the list.

xpl proto *list entity dimension proto_# resolution symbol*

This command makes a list of all peaks, with position in that dimension, that align within the *resolution* with the given protopattern's (*proto_#*) frequencies from the **Assign** database. The value placed in *symbol* represents the number of items placed in the list.

xpl pt *list entity dimension assignment_ptr symbol*

This command makes a list of all peaks, with an assignment pointer in that dimension, that matches the given assignment pointer. The value placed in *symbol* represents the number of items placed in the list.

Spreadsheet interface

The database can be viewed and modified through a visual spreadsheet (table) interface. This section describes the commands of this interface.

Similar to a spectral frame, a spreadsheet is a child window (called table frame) inside the FELIX main window. A table frame usually has its own menu bar and toolbar. FELIX displays only the menu bar and toolbar for the currently active table or spectral frame. Therefore, you must highlight a frame to access its menu items or toolbar icons.

There is a set of commands for you to compose and operate on tables. In all these commands, the name of the table is used to refer to the table frame.

Table composition commands

The **tbl com(pose)** commands define a table from one or multiple entities in the database and then display the table. The options are:

```
tbl com(pose) cre(ate) table_name #_of_rows #_of_cols
```

This creates an empty table with *#_of_rows* rows and *#_of_cols* columns with the *table_name* name.

```
tbl com(pose) dis(play) table_name
```

This finishes the table definition and displays it. You fill in the table using the **tbl set** commands.

You can create tables for whole entities using the following variation of the **compose** command:

```
tbl com(pose) ent table_name entity_name
```

There are some specific database entities, which may be a combination of other entities (for example, spin systems) or which may have a stored entity that is different from the table view (for example, peaks or prototype patterns). These entities require specific commands to view them:

tbl com(pose) xpk *table_name entity_name*
creates a peak table

tbl com(pose) pat *table_name entity_name*
creates a spin-system or pattern table

tbl com(pose) pro *table_name entity_name*
creates a prototype pattern table

Table menu command

The **tbl pul(ldown)** command allows you to to specify the menu bar using a .mnu file:

```
tbl pul table_name menu_file
```

The format of such a .mnu file is described in Appendix , *Menus and Control Panels*.

Table icon command

The tbl ico command defines the tool bar using a .mot.file :

```
tbl ico table_name icon_file
```

The format of such a .mnu file is described in Appendix , *Menus and Control Panels.*

Table frame layout commands

The following commands are used to manage the table frames from a macro:

```
tbl zero
```

closes all table frames.

```
tbl resize tblname x0 y0 xsize ysize
```

resizes a table frame to desired location and size. All are in pixels. *x0* and *y0* are the *x* and *y* coordinates of the low-left corner of the frame and *xsize* and *ysize* are the width and height, respectively.

```
tbl move tblname x0 y0
```

moves a table frame to desired location. All are in pixels. *x0* and *y0* are the *x* and *y* coordinates of the low-left corner of the frame, respectively.

```
tbl front tblname
```

activates a specified table frame.

```
tbl exist tblname symbol
```

checks if the specified table frame exists or not. If it exists, *symbol* has value of 1. Otherwise, 0.

Table navigation

The **tbl jum(p)** command allows you to move to a specific location in a table. This moves the active cell or row to that position, too. The options are:

```
tbl jum cel table_name row_number column_id
tbl jum row table_name row_number
tbl jum col table_name column_id
```

Table selection

The **tbl sel(ect)** command allows you to select or highlight a cell, row, column, or the full table. The options are:

tbl sel cel *table_name row_number column_id*
selects (highlights) a cell of a specific *row_number* row and *column_id* column.

tbl sel row *table_name row_number*
selects a full row.

tbl sel col *table_name column_id*
selects a full column.

tbl sel all *table_name*
selects the full table.

Table unselection

The **tbl uns(elect)** command allows you to deselect or unhighlight a cell, row, column, or the full table. The options are:

```
tbl uns cel table_name row_number column_id

tbl uns row table_name row_number

tbl uns col table_name column_id

tbl uns all table_name
```

Set a table to an entity

The **tbl ent(ity)** command sets a table type to a specific entity. This command does not fill in the table with the entity, in contrast to the **tbl com(ment)** command.

```
tbl ent table_name entity
```

Querying selection status of a table

The **tbl whi(ch)** command returns a list of rows or columns that are selected or highlighted:

```
tbl whi table_name list# number_of_items type_of_selection
```

The *number_of_items* variable is **0** if none is selected and **-1** if all are selected. The selected row or column numbers are returned in the *list#* list, and the *type_of_selection* symbol is set to **1** if rows are selected, to **2** if columns are selected, or to **0** if neither rows nor columns are selected.

Querying an item

The **tbl get** command returns a string of a contents of a cell or row:

```
tbl get cel table_name row_number colum_id string
```

returns the contents of a cell into *string* symbol, and:

```
tbl get row table_name row_number string
```

returns the contents of a whole row into the *string* symbol.

Deleting a row

The **tbl del(ete)** command deletes a row from the *table_name* table:

```
tbl del table_name item_number
```

Sorting a table by a column

The **tbl sor(t)** command sorts a table by a selected column and stores the sorted row numbers in a list:

```
tbl sor table_name column_id order list#
```

where *order* is descending (**0**) or ascending (**1**).

> **Note:** This kind of sorting affects only the spreadsheet view of the entity, not the entity itself.

Finding in a table

The **tbl fin(d)** command finds all instances of items that match a certain criterion and puts them into a list. The command works in a column. The resulting list contains the item numbers. There are two forms -- one for finding strings:

```
tbl fin table_name column_id match_type match_value list#
number_of_matches
```

where *match_type* can be exact matching (**ex**) where * are taken literally, or wildcard matching (**match**), e.g.: a string in a form quot;*abc*".

The other form is for numeric comparisons:

```
tbl fin table_name column_id match_type match_value (match_
value2) list# number_of_matches
```

where *match_type* can be one of the following types:

eq

Equal to *match_value*

ne

Not equal to *match_value*

lt

Less than *match_value*

le

Less or equal to *match_value*

gt

Greater than *match_value*

ge

Greater or equal to *match_value*

wi

Within *match_value* and *match_value2*

ou

Less than *match_value* or greater than *match_value2*

Adding a row

The **tbl add** command add a new row to the end of a table:

```
tbl add table_name rownumber
```

The row can then be filled with the **tbl set** command. The *rownumber* symbol is returned by the command.

Querying the number of tables active

The **tbl who** command returns the number of tables active in FELIX in the *num_tables* variable or the name of the table in the *table_name* variable, if a specific table is defined in the *table#* variable:

```
tbl who num_tables

tbl who table table_name
```

Closing a table

The **tbl clo(se)** command closes a table, but does not destroy the database entity:

```
tbl clo table_name
```

Querying the number of rows

The **tbl row** command returns the number of rows in a table:

```
tbl row table_name number_of_rows
```

Querying the number of columns

The **tbl col(umn)** command returns the number of columns in a table:

```
tbl col table_name number_of_columns
```

Setting a value

The **tbl set** commands allow you to set the value of a row, column, or cell or to set the item number to a particular value:

```
tbl set cell table_name row_id column_id value
```

This sets the (*row_id*,*coumn_id*) cell to *value*. To set cells in a full row, use this command:

```
tbl set row table_name row_id value value2 ...
```

The column command sets the title of the column:

```
tbl set column table_name column_id value
```

Double-click action

The **tbl dbc** command allows you to connect a macro to a row double-click, so that double-clicking a row with the cursor executes the macro and passes the row number to the macro:

```
tbl dbc table_name macro_name
```

Column width

The **tbl wid(th)** command allows you to set the width of a column in characters:

```
tbl wid table_name column_id width_in_characters
```

Updating the table

The **tbl upd(ate)** command allows you to update a table after an entity has been changed:

```
tbl upd table_name
```

This updates the full table.

```
tbl upd table_name item_number
```

This updates a particular item in the row or adds a new row with that item from the entity. The entity update from the table happens automatically.

Querying the item

The **tbl ite(m)** command allows you to query the row number where a specific database item is shown in the table:

```
tbl ite table_name item_number row_number
```

The returned value of *row_number* is greater than zero if it is found or zero in case of error.

Table callback

When you select a menu item or click a toolbar icon, a callback macro associated the command is executed immediately. While a command from the main menu and main toolbar (associated with a spectral frame) always calls the callback macro exactly once, a command from a table has a slightly different way of callback control.

As described in the section, “Changing the menubar interface”in Appendix , *Menus and Control Panels*, a menu item has the following format:

```
item LABEL HOTKEY ACCELKEY CALLBACK CBCONTROL NULL DEPEND
```

If it is a command for a table, the method to call the CALLBACK command depends on the current selections in the table and the value of CBCONTROL:

If CBCONTROL = **NULL** or **0**, the CALLBACK command is executed N+1 times, where *N* is the number of cells / rows / colums selected. For each cell / row / column selected, the CALLBACK command is executed with some appended parameters:

```
CALLBACK table_name item_number row_number column_number
```

And after the last one, the terminating call:

```
CALLBACK table_name -1 -1 -1
```

If a full row is selected, the *column_number* is **0**. If a full column is selected, the *row_number* is **0** and the *item_number* is **0**.

If CBCONTROL = **1**, the CALLBACK command is executed twice, first for the first cell / row / column selected, with the appended parameters:

```
CALLBACK  table_name item_number row_number column_number
```

And after the last one, the terminating call:

```
CALLBACK  table_name -1 -1 -1
```

If CBCONTROL = **2**, the CALLBACK command is executed only once with appended parameters:

```
CALLBACK  table_name -1 -1 -1
```

For example, if a menu item defined as the following:

```
item "Select" S NULL NULL "ex select 1" NULL NULL
```

is selected from a table **xpk-peaks**, while two rows (#1 and #2) are selected, the following commands will be executed:

```
ex select 1 xpk:peaks 1 1 0

ex select 1 xpk:peaks 2 2 0

ex select 1 xpk:peaks -1 -1 -1
```

abl — Automatic baseline flattening

abl *noise points line points'*

noise points	number of points for noise convolution
line points	number of points for widest line

The **abl** command first determines whether each point in the work space is baseline or not. abl then goes through all points, convolving baseline points with a moving average and applying a straight-line correction to non-baseline intervals. This algorithm was developed by W. Massefski.

abl is very robust and requires as input only estimates of the point-wise half width of the widest spectral line and the desired convolution width for noise. This algorithm has the unusual characteristic of not only flattening the baseline, but convolution of baseline intervals actually reduces the noise level. **abl** does an excellent job on sparsely populated spectra such as slices of multidimensional data that contain mostly baseline. For reliable operation, *line points* should be set much larger than *noise points*, otherwise **abl** may interpret the tails of peaks as noise and bite chunks out of them.

Since **abl** selects unique baseline points for each spectrum, it is not necessary to define a list of baseline points, nor are the points used saved in the baseline entity.

See also

flf — FaceLift baseline correction

abp — Automatic baseline point selection using FLATT algorithm

abp *baseline_width chi tau stride*

baseline-width	number of points in linear fit
chi	minimum chi square
tau	cutoff factor
stride	set interval for baseline point testing

abp uses the FLATT algorithm (Güntert, 1992) for selecting baseline points in a spectrum. The resulting points are stored in the entity whose name is stored in the *basent* symbol. Before running the **abp** command, you may decide to run the **chi** command: the value returned by the **chi** command is stored in the *chi* symbol and is read automatically by **abp**.

abp begins by calculating the chi-square value for a fit of (2**baseline_width* + 1) data values to a straight line at points (1 + *baseline_width*) to (npoint - *baseline_width*).

The command works by comparing the minimum chi-square value in an interval of (*baseline_width* * 2 / 3) that is centered on point (*baseline_width* / 3 + 1). If the minimum chi-square value is less than (*tau* * *chi*), the central point is stored in the baseline point entity. The comparison is repeated every *stride* points.

Symbol dependence

chi Minimum chi-square value

See also

chi — Calculate minimum chi-square value

flt — FLATT baseline flattening

flf — FaceLift baseline correction

abq — Automatic selection of baseline points

abq *interval size standard deviation*

interval size	interval size used for selection of base points
standard deviation	maximum deviation allowed for baseline points

abq selects baseline points of the data in the work space automatically, and places the resulting points in a database entity defined by the symbol basent.

If no parameters are entered, **abq** uses defaults that work well for most spectra. First, **abq** divides the data in the work space into segments of interval size in points. For each segment, it calculates the mean data value of the points and the deviation of each point from the mean.

Next, it collects the largest deviation value, which is called the segment deviation, from each segment and orders them from smallest to largest.

Finally, any segment with a segment deviation that is less than a cutoff value is to determined to be baseline. The cutoff value is calculated as the product of standard deviation and the smallest segment deviation value. The bas command can be used to add or delete baseline points in the baseline entity.

Following **abq**, a polynomial or cubic spline baseline correction may be executed using either polynomial baseline correction (pol) or cubic spline baseline correction (csp).

Symbol dependence:

basent Defines Baseline Points Entity

See also

csp — Cubic spline baseline correction

flf — FaceLift baseline correction

pol — Polynomial baseline correction

bas — Baseline points manipulation

abs — Absolute value replacement of work

abs

abs replaces each point in the workspace with its absolute value, i.e.:

value = | value |

abs should not be confused with ms, which calculates the absolute magnitude of complex points.

Symbol dependence

datsiz Number of Data Points

See also

ms — Magnitude spectrum

ps — Power spectrum

adb — Add work to buffer

adb *buffer number*

buffer number number of buffer to which to add the data in the work space

adb adds the data in the work space to the specified buffer. This command is very useful for saving intermediate results or for generating projections of multidimensional spectra.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

ldb — Load buffer into work space

mwb — Multiply work by buffer

stb — Store work space to buffer

add — Add number to work

add *number*

number number to be added to work

add adds the specified number value to each data point in the work space.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

mul — Multiply the work space by a number

aln — Antilogarithm (exponential) of work

aln

aln replaces each data value of the workspace with e(work), its natural (base e) antilogarithm or exponential. The **aln** command is the inverse of the logarithm of workspace (**log**) command.

Symbol dependence

datsiz Number of Data Points

See also

log — Natural logarithm of work space

alt — Alternating real/imaginary

alt

alt changes data consisting of separate real and imaginary parts to alternating real and imaginary parts. All complex data within FELIX is processed in the alternating mode. The **alt** command is thus useful for restoring imaginary parts of *N*-dimensional hyper-complex spectra (separated using **sep**) for phasing after transformation. **alt** defines the data type to be complex (datype=**1**) and sets the data size (**datsiz**) to half the original number of real points.

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol Changed

datype Data Type

See also

sep — Separate real and imaginary

ann — Annotate plot

ann

ann annotates the current plot according to the contents of the file designated by the symbol annfil. The annotation file contains annotation commands. To annotate hardcopy plots, define the symbol **pltann** to **1** before issuing the hcp command.

Symbol dependence

annfil Annotation File

annpfx Annotation Prefix

See also

arr — Arrow annotation

lin — Line annotation

tex — Text annotation

gre — Greek text annotation

rec — Rectangle annotation

aph — Autophase spectrum

aph

aph zero

aph exclude *dim minpt maxpt*

aph exclude

aph calculate *opt peak_width min_phase1 max_phase1*

aph provides functions for automatic phase correction for 1D and ND spectra. For details see Chapter 1, "Theory", in the FELIX *User Guide*.

Used without subcommands, **aph** calculates the phase error and corrects the spectrum displayed in the workspace. It is sensitive to several factors, including bad baselines. Autophasing a spectrum that has a large amount of baseline roll does not yield a well phased spectrum. **aph** by itself is recommended for proton spectra. It does not use the excluded areas as it does when subcommands are included.

aph zero removes all defined excluded areas.

aph exclude adds an excluded area that is ignored while computing the phase parameters using the aph cal subcommand. Any datapoints falling between minpt and maxpt along dimension dim are ignored. An excluded area is usually a solvent region. Up to 10 excluded areas can be defined. If dim, minpt, and maxpt are not specified, aph exclude displays the currently defined excluded areas.

aph calculate calculates the zero and first-order phase parameters for a 1D spectrum in the workspace or for a certain dimension of the current 2D or 3D spectrum. It does not change the spectrum.

opt	opt = dim * 10 + method. If dim = 0, the 1D spectrum in the workspace is used. Otherwise, the dimension Ddim (D1, D2, or D3) of an ND spectrum is used. If method = 0, the PAMPAS method is used for phase error detection. If method = 1, the APSL method is used for phase error detection. If method = 2, the integration method is used for calculation of phase parameters (in 1D spectra only.)
peak_width	Minimum peak width at the half height for a sample peak. Sample peaks are used to fit phase parameters. Recommended value 3-6. Default = 3.
min_phase1	Lower limit of phase1, the first-order phase. Default = -720.
max_phase1	Upper limit of phase1. Default = +720.

Symbols changed

phase0 Zero order phase correction

phase1 First order phase correction

See also

ph — Phase correction

rph — Real-time phase

arr — Arrow annotation

arr *X0 Y0 (Z0 (A0)) X1 Y1 (Z1 (A1))*

X0	x-coordinate for starting position of arrow
Y0	y-coordinate for starting position of arrow
Z0	optional z-coordinate for starting position of arrow if matrix is more than 2D
A0	optional a-coordinate for starting position of arrow if matrix is more than 3D
X1	x-coordinate for ending position of arrow
Y1	y-coordinate for ending position of arrow
Z1	optional z-coordinate for ending position of arrow if matrix is more than 2D
A1	optional a-coordinate for ending position of arrow if matrix is more than 3D

arr draws an arrow on the current plot with its tail at the (X0,Y0) point and its head at the (X1,Y1) point. The optional coordinates are useful if the matrix is a 3D or 4D and the current plot is a strip plot. The arrow should be drawn starting in one strip and ending in another strip, where the strips are taken from different slices of the ND matrix. All arrow coordinates are interpreted based on the symbol annunt according to the following table:

annunt value	coordinate interpretation
0	normalized in plot
1	normalized in frame
2	axis units
3	points
4	ppm

The color of the line is determined by the symbol anncol, the style of the line (solid or dashed) is defined by annlst and size of the arrowhead is determined by the symbol annasz.

Symbol dependence

anncol Annotation Color

annasz Arrowhead Size

annlst Annotation Line Style

annunt Annotation Units

See also

ann — Annotate plot

lin — Line annotation

bas — Baseline points manipulation

bas *op parameters*

bas provides the ability to assemble and display the array of points that are required for baseline correction. The entire array of points may be selected interactively, or alternatively, baseline points may be selected automatically using **abq**, and additional points may be added manually.

The *x-axis* point value specifies the data point to add to the baseline points database entity (by default, **bas**:**baseline**). A parameter value of -1 for add enables a crosshair cursor for point selection. The entire array of baseline points may be discarded by invoking **bas zero**.

The baseline points manipulation operators and their parameters are below:

bas add x-axis point value

bas show

bas zero

Symbol dependence

basent Baseline Entity

See also

abq — Automatic selection of baseline points

csp — Cubic spline baseline correction

flf — FaceLift baseline correction

pol — Polynomial baseline correction

bc — Baseline correct

bc *fraction*

fraction fraction of data used to compute baseline offset; default = 0.25 (last 1/4 of FID)

bc removes a DC offset of the FID baseline (DC offset creates a spike at the carrier frequency) by subtracting the offset from work. bc will work properly only if the data at the end of the FID is a baseline. Adequate DC offset correction is usually obtained by using the default fraction value of 0.25, although, for FTIR, fraction is set to 1.0, and bc is computed on the entire contents of the work space, as there is no intensity at zero frequency.

Symbol dependence

datsiz Number of Data Points

bck — Back-calculate NOE intensities

bck *spin shift bckxpk bckvol mix rleak*

bck calculates expected NOE cross peak intensities from the structure in the entity xyz:atoms. Chemical shifts and line widths are extracted from the shift entity, and a cross peak entity bckxpk and corresponding volume entity bckvol are built. The volumes are calculated for the specified mixing time mix in seconds.

The *rleak* parameter specifies the leakage rate of Z-magnetization over time and is in sec-1 units. The cross-relaxation rate is determined by the symbol taucee in ns units.

The symbol *bckrad* allows you to save time by ignoring spin pairs further apart than bckrad Å. This algorithm generates NOE intensity by using matrix doubling at a spectrometer frequency spcfrq in MHz units.

The symbol *minzee* is used to filter out interactions below some sensible threshold that would not be observed experimentally. There is an upper limit of 2048 spins for the simulation.

There is another form of this command which can be used in conjunction with the **Assign** module:

bck *spectrum_id bckxpk bckvol mix rleak*

Once you have a *spectrum_id* spectrum defined in the **Assign** database, FELIX 2002 calculates the theoretical spectrum using all the information in the database (e.g., assigned patterns, assigned peaks, mixing times, transfer types, and spectrometer frequencies). That also means that if the spectrum specified is a 3D NOE-NOE, then FELIX will back-calculate a 3D NOE-NOE theoretical spectrum.

The theory behind **bck** is described in Chapter 2, "Theory", in the FELIX *User Guide*. **bck** will efficiently generate expected spectra, and is very useful when generating structures from NOE data.

Symbol dependence

taucee Correlation Time (ns)

bckrad Back-Calculation Radius (Å)

minzee Minimum Z-Magnetization

spcfrq Spectrometer Frequency (MHz)

bft — Bruker-Fourier transform

bft

bft transforms a Bruker FID in the workspace into a frequency-domain spectrum. Many Bruker spectrometers acquire real and imaginary data at alternate points in time rather than simultaneously, resulting in rolling baselines and phase errors. Modern Bruker spectrometers are capable of true quadrature, and we recommend that you use this acquisition mode if possible.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

ft — Fast Fourier transform

rft — Real Fourier transform

ift — Inverse Fourier transform

bir — Read database from Insight II

This set of commands allows you to read the NMR Refine database files from the disk.

bir pks *file*

file Insight II peak file (.pks)

bir pks reads an Insight II peak file into the current peak entity (table).

bir asn *file*

file Insight II peak assignment file (.asn)

bir asn reads an Insight II peak assignment file into the current peak entity (table).

bir ppm *file*

file Insight II resonance assignment file (.ppm)

bir ppm reads an Insight II resonance assignment file into the current pattern entity (table).

> **Note:** This deletes all the existing patterns from the Assign project.

bir rstrnt *file*

file Insight II restraint file (.rstrnt)

bir rstrnt reads an Insight II restraint file into the current restraint entities (tables).

Symbol dependence

pksent Current peak table

volent Current volume table

rpaent Current pattern table

rreent Current resonance table

See also

ins — Insight II-FELIX inter-process communication

biw — Write database to Insight II

bit — Bit manipulation operators

bit clear *maskin bit symbol*

The **bit clear** command clears a bit in the mask to zero. If bit is less than one, all bits in the mask are zeroed. The new bit mask is returned in symbol.

bit set *maskin bit symbol*

The **bit set** command sets a bit in the mask to one. If bit is less than one, all bits in the mask are set to one. The new bit mask is returned in symbol.

bit test *maskin bit symbol*

The **bit test** command tests a bit in the mask. The value of that bit (zero or one) is returned in symbol.

bit or *mask1 mask2 symbol*

The **bit or** command combines two bit masks using the logical "or". The new bit mask is returned in symbol.

bit and *mask1 mask2 symbol*

The **bit and** command combines two bit masks using the logical "and". The new bit mask is returned in symbol.

bit xor *mask1 mask2 symbol*

The **bit xor** command combines two bit masks using the logical "xor". The new bit mask is returned in symbol.

bit not *maskin symbol*

The **bit not** command reverses all the bits in a mask. The new bit mask is returned in symbol.

The following parameters are used with the bit commands:

mask	an integer bit mask
bit	a bit number in the range 1 to 32
symbol	the symbol to receive the resultant bit mask

biw — Write database to Insight II

This set of commands allows you to write to the NMR Refine database files to the disk.

biw pks *file*

file Insight II peak file (.pks)

biw pks writes an Insight II peak file to the disk using the current peak entity (table).

biw asn *file*

file Insight II peak assignment file (.asn)

biw asn writes an Insight II peak assignment file to the disk using the assignments in the current peak entity (table).

biw ppm *file*

file Insight II resonance assignment file (.ppm)

biw ppm writes an Insight II resonance assignment file to the file using the current pattern entity (table).

biw rstrnt *file (type)*

file Insight II restraint file (.rstrnt)

type optional type descriptor, if omitted or the *type*=0 the restraints are written in Discover format to the file *file*.rstrnt, if *type*=1 the restraints are written in X-PLOR format into files *file*.xdcn, *file*.xdih and *file*.xint

biw rstrnt writes restraint file to the disk using the current restraint entities (tables).

Symbol Dependencies

pksent Current peak table (needed for biw asn and biw pks)

volent Current volume table (needed for biw asn and biw pks)

rpaent Current pattern table (needed for biw ppm)

rreent Current resonance table (needed for biw ppm)

noerst Current NOE-distance restraint table

noeors Current NOE-overlapped distance or ambigous distance restraints

dihrst Current dihedral restraint table

mixrst Current mixing time table

volrst Current NOE-volume restraint table

volors Current NOE-volume overlapped or ambiguous intensity restraints

rchrst Current remote-chiral restraint table

chirst Current chiral restraint table

j3drst Current 3J-dihedral restraint table

ndirst Current NMR dihedral restraint table

disrst Current distance restraint table

See also

ins — Insight II-FELIX inter-process communication

biw — Write database to Insight IIe

bld — Build a matrix file

bld *file_name dimensions size1 size2 ... sizeN type*

file_name	name of the matrix to be created-Matrix files are given the file extension .mat and prefix **matpfx**
dimensions	number of dimensions of the matrix — Up to 6-dimensional matrices can be created
size1	size of the first dimension in points
size2	size of the second dimension in points
sizeN	size of the N-th dimension in points
type	data type: **0**=real (default), **1**=complex
overwrite flag	y if overwrite needed in case if a file exist with the same name

overwrite flag

In the following example, a three-dimensional matrix called t*est.mat*, 512 ¥ 256 ¥ 32 points, is created.

```
bld test 3 512 256 32
```

bld creates a file, or a series of files, to contain an N-dimensional matrix. The size of the matrix is restricted to powers of two in each dimension, with a minimum size of 4 points. If a matrix is defined with a size that is not a power of two, **bld** will use the next highest power of two. FELIX matrices exist as direct-access files on disk, and may exist as multiple files if desired. The maximum file size of the matrix is defined using the matrix frame size symbol (mframe). When the actual size of a matrix exceeds mframe megabytes, multiple files will be created. Once a matrix has been built using bld, you open it using the command mat to access vectors, planes, and other subspaces.

Symbol dependence

matpfx Matrix Prefix

mframe Matrix Frame Size

See also

mat — Open matrix

bml — Get molecule name

bml *moleculename*

moleculename Insight II molecule name

bml clears up the molecule buffer and sets up the molecule name. This is needed if the molecule is displayed in Insight II.

Symbol changed

bmlname Molecule Name

See also

ins — Insight II-FELIX inter-process communication

bir — Read database from Insight II

biw — Write database to Insight II

bun — Set bundle mode

bun *dimension*

dimension dimension for bundle mode

bun defines a matrix dimension for bundle mode operation. **bun** also defines the value of the reserved symbol vector as the total number of vectors along the specified dimension in the matrix. For example, for a 512 x 256 x 32 matrix, you have to perform 512 x 256 operations when transforming the third dimension, therefore vector is calculated to be 131072 (= 512 times 256). In this example, with **bun 3**, successive vectors are loaded into the work space from the matrix using the load work space from bundle command (**lwb**), operated on (apodized, Fourier transformed, and phased) and stored back to the matrix using the store work space to bundle command (**swb**). Following the last **lwb** or **swb** access, the **bun 0** command is issued to terminate bundle mode access and to return to discrete access mode.

Bundle mode is very useful for processing all vectors along a single dimension of a matrix in exactly the same way when the order of processing does not matter. For example, when processing the third dimension of a 3D experiment, it does not matter which D3 vector is transformed first, only that it is transformed once during the processing. A matrix transformation performed in bundle mode is many times faster than the same transformation performed in discrete vector mode. To exit bundle mode and enter discrete vector mode, enter **bun 0**.

At any one time, a matrix must be either in bundle mode or in discrete vector mode (the default). When in bundle mode, the load command (**loa**) and store command (**sto**) cannot be used to access discrete vectors in the matrix. Likewise, when in discrete vector mode, the bundle mode commands **lwb** and **swb** cannot be used.

Symbol changed

vector Number of Vectors in the Entire Bundle

See also

lwb — Load work space from bundle

swb — Store work space to bundle

by, bye — Exit FELIX

bye

bye ends the session with FELIX and returns you to the operating system. Any open matrices are closed. If a database is in use, you will be prompted to save or discard changes for this session.

cal — Macro call

cal *$macroname*

cal *label*

cal performs a subroutine call within a macro. The called macro is read into the macro work space and given a label of *$macroname*. Control is transferred to the first line of the called macro, and the macro executes until complete. Control then returns to the line following the cal command. The called macro remains in the macro work space until another macro is executed, so repeated calls will be efficient. **cal** is only valid in a macro.

> **Note:** The **cal** command is obsolete and can usually be replaced by the **exr** command. It is retained in FELIX solely for compatibility with existing user macros.

See also

exr — Execute a macro and return

rf — Read FELIX for Windows file

cd — Convolution difference window

cd *line broadening*

line broadening line broadening in Hz

cd is an apodization function, which is really a special case of convolution difference, namely, the difference between no line broadening and lbroad. The **cd** command uses the global symbol *lbroad* if no line broadening parameter is entered. **cd** multiplies each point in the work space by the function:

1 - e - (line broadening t)

cdf — Conditional define

cdf *symbol value*

cdf lets you define a symbol to have a value, only on the condition that the symbol is not yet defined.This is a good way to guarantee that a symbol has a value without changing the value if it already exists. Since it is an error to use an undefined symbol, this command can simplify writing robust macros.

cfg — Configure memory

cfg *size count*

size maximum size for work or buffers (in complex points)
count maximum number of work + buffers

cfg allows you to reserve memory for 1D work spaces. When configuring memory, a good rule of thumb is to set the 1D work space size to the maximum size data you plan to work with, and then set the count parameter to the maximum number of buffers you need plus one. If you need access to more than one buffer, simply set count to a larger value (the maximum value is 64). However, by setting count to a larger value (thus, allocating precious memory to 1D work space), you may be unable to open a multidimensional matrix (matrices also need memory). In this case, it may be necessary to reduce the size of the 1D work space and buffer memory with the **cfg** command.

You may notice that using the **cfg** command while a matrix is open will close the matrix before reallocating the 1D workspaces. After configuration, you must open the matrix again to access it.

Symbol changed

frsize Workspace and Buffer Size

nframe Number Of 1d Buffers

See also

mmp — Display memory map

cgd — Change values in the control panel

cgd

cgd updates symbols on a non-modal (mnu a) control panel.

chi — Calculate minimum chi-square value

chi *baseline_width*

baseline_width baseline segment width

For the spectrum in the work space, **chi** calculates a minimum chi-square value, which is stored in the symbol *chi* and used by the **flt** command. This minimum chi-square value is calculated by fitting the data values within a window of baseline_width length to a straight line, and calculating the chi-square value for the fit. The chi-square value is calculated for each full window of data values, as the window is moved point-by-point along the spectrum. The smallest chi-square value is stored in **chi**.

Symbol Changed

chi Minimum Chi-Square Value

cl — Close a data file

cl

cl closes the current data file being accessed by FELIX 2002. A subsequent read command (**re** or **rn**) reads the first record of the data file.

See also

re — Read a file (old format)

rn — Read file (new format)

wr — Write a file (old format)

wn — Write a file (new format)

clr — Clear frame

clr

clr causes the current graphics frame to be erased. The frame will no longer have any current graphics context or mapping.

Symbol changed:

disply Current Frame Display

cls — Close output file

cls

cls closes the current output file. Unless an output file is open, the put record command (**put**) will take no action.

See also

put — Put record

opn — Open output file

cmb — Change symbol on the user interface

cmb *symbol value*

symbol symbol to change in the user interface (for example, command name)

value the new text for the symbol

This command changes the text in the FELIX graphical user interface (GUI), where an item was defined via the symbol.

cmd — List commands

cmd *match string*

match string optional string, which may contain a wild card (*)

This command causes FELIX to list all commands. If a match string is entered, only commands that match the string will be listed.

cmx — Close matrix file(s)

cmx

cmx closes all open matrix files. The matrix buffer is also de-allocated.

Symbol changed

matfil Current Matrix File

dimen Number of Matrix Dimensions

See also

bld — Build a matrix file

mat — Open matrix

cnj — Complex conjugate

cnj

cnj negates the imaginary part of the data in the workspace. This command will reflect the spectrum about zero frequency if it is performed before the ft.

Symbol dependence:

datsiz Number of Data Points

datype Data Type

cnv — Time-domain convolution

cnv *window_type window_size extrapolation*

window_type	convolution function to convolve with the data: **0**=sinebell, **1**=gaussian
window_size	number of points used to define the convolution function
extrapolation	type of extrapolation used between point one and point *window_size* and between point (**datsiz** — *window_size*) and point **datsiz**: **0**=average slope at tails (original method) **1**=linear prediction of tails (new method)

cnv is used to eliminate huge solvent lines and the effect of the tails of these lines on less intense signals. cnv convolves the FID with the selected window of the specified width, then subtracts the result from the original FID. Lines are removed only within a small range of zero frequency, and the effective width of the range is dependent on the window size used, that is, the wider the win-

dow, the narrower the width of the range. Convolution of finite length data sets necessarily begins at the point window_size and ends at the point (**datsiz** — *window_size*). Therefore, another method for estimating the solvent signal must be used to for the first and last *window_size* points. The *extrapolation* parameter allows you to select either a linear extrapolation, which is fast and usually sufficient, or a linear prediction estimation, which is slow but often more accurate. The linear prediction can produce undesirable results under certain conditions; for example, when the *window_size* is narrow, the convolved data possesses higher frequency components that the linear prediction subsequently attempts to extrapolate. For *extrapolation* of one, the linear prediction parameters are *points* = 32, *coefficients* = 16, and *peaks* = 8. This technique was developed by Dominique Marion.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

lpx — General linear prediction

lpf — Linear predict first points

lpl — Linear predict last points

com — Execute FELIX commands in macros

com

com puts you in touch with the command interpreter of FELIX while a macro is running. This allows you to input a FELIX command while a macro is running. The command is then executed within the macro. The string must be a valid FELIX command line. **com** is only valid in a macro.

cp — Contour plot

cp

cp draws a contour plot of the current plot region, defined by the matrix limits command (**lim**) on the current graphics device. The lowest contour level is determined by the product of the reserved symbols level and mscale.

The contouring algorithm can perform spline interpolation (contyp=1) between the real points, making the appearance of the plot smoother. While this improves the appearance, the speed of the plotting decreases.

The default rendering mode erases the screen before each display; this action may be disabled by setting the reserved symbol erase to 1. Video buffering may be enabled by setting the reserved symbol animat to 1. The size and graphics attributes of the region plotted by the **cp** command are affected by a number of other reserved symbols.

Symbol dependence:

animat Specifies Video-Buffering

clmode Selects Linear or Geometric Contour Spacing

conmod Modifies Contour Level

contyp Interpolation Type

cycle Sets Color Cycle

drwbox Draws Box Around Plot

drwpks Draws Peaks Switch

erase Disables(0)/Enables(1) Automatic Screen Erasing

grid Specifies Superimposed Grid Lines

level First Contour Level

mscale Matrix Scale Factor

nlevel Number of Contour Levels

posneg Enables Plotting of Negative Contour Levels

projct Selects Dimensionality of Display

pennum First Color

rowinc Point Skipping Factor for 3D Displays

xpklbl Label Peaks Switch

Symbol changed

disply Current Display Type

See also

ip — Intensity plot

np — Null plot

sp — Stack plot

ovc — Overlay contour plot

pla — Redisplay 3D object

rmx — Reference matrix

cpl — Real to complex

cpl

cpl turns a real vector into a complex vector with an imaginary part of zero. This command works only on data stored in the current workspace.

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol changed

datype Data Type

See also

red — Reduce complex to real

csh — Circular signed shift

csh *n1 n2 scale factor*

n1	first point value
n2	second point value
scale	scaling factor for shift

csh shifts the data in the work space left or right the number of points specified by the value [(n1-n2)*scale]. Negative shifts move to the left, positive shifts move to the right. Points shifted off the leading edge of the spectrum are "circular shifted" to the trailing edge. The **csh** command is used for tilting spectra where the scale is usually set to the digital resolution ratio of the 1 and 2 dimensions.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

shl — Shift left

ssh — Signed shift

shr — Shift right

csr — Circular shift right

csl — Circular shift left

csl — Circular shift left

csl *points*

points	number of points to shift data in work space

csl shifts the data in the work space left by the number of points specified. Points shifted off the left edge of the spectrum are circular shifted back onto the right edge of the spectrum.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

shl — Shift left

ssh — Signed shift

shr — Shift right

csr — Circular shift right

csl — Circular shift left

csp — Cubic spline baseline correction

csp

csp performs a cubic spline baseline correction on the contents of the workspace according to the baseline points defined in the baseline entity *basent*. The baseline points may be set automatically using the automatic selection of baseline points command (**abq**). **csp** uses the value of the interval width symbol (*iwidth*) to minimize the effects of noise on the correction by averaging the points within the interval about each baseline point plus and minus the value of *iwidth.*

A cubic spline will correct each baseline point to exactly zero. This can present a problem if **csp** is used to correct the first dimension of a multidimensional transform. Since each defined baseline point will be corrected to zero, the transform along the next dimension will see all zeroes when the vector passing through the baseline point is loaded. The FFT of all zeroes is all zeroes, and if a DC offset is present in the data, this will appear as ridges or valleys in the transformed data. Accelrys recommends using **pol** for baseline correction during transforms.

Symbol dependence

datsiz Number of Data Points

datype Data Type

iwidth Interval Width

See also

abq — Automatic selection of baseline points

flf — FaceLift baseline correction

pol — Polynomial baseline correction

bas — Baseline points manipulation

csr — Circular shift right

csr *points*

points number of points to shift data in work space

csr shifts the data in the work space right by the number of points specified. Points shifted off the right edge of the spectrum are circular shifted back onto the left edge of the spectrum.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

shl — Shift left

ssh — Signed shift

shr — Shift right

csr — Circular shift right

csl — Circular shift left

cur — Cursor control

cur *wait_mode map behavior exit_style*

cur allows you to control the cursor location and appearance, wait for mouse or keyboard events, and obtain positional information from the graphics display.

The general outline of the behavior of the **cur** command is:

1. Change the cursor to a particular style, and optionally preposition it at a specified position on the graphics display.
2. Wait for zero, one, or two mouse button or keyboard events.
3. Return positional (x, y location) and event (key or mouse button) information from the cursor into reserved symbols for future use.
4. Change the cursor to a particular style on exit.

There are many variations of the **cur** command, with the specific behavior determined by the parameter values. General descriptions of each parameter are given below:

parameter	setting	description
wait_mode		The number of mouse button or keystroke events **cur** waits for before returning

0 Return immediately. This mode lets you set or get the current cursor position and change the cursor style without waiting for any event. The new cursor position is returned in **x0pnt** and **y0pnt**.

1 Wait for a single mouse button or keyboard key press event. This mode allows you to set the current cursor position, change the cursor style, and then wait for you to move the cursor to any position on the graphics display using the mouse. Mode **1** returns when a mouse button or keyboard key is pressed. The selected position of the cursor is returned in **x0pnt** and **y0pnt**, and the ASCII code of the key pressed is returned in

keyhit.

2 Wait for a mouse press-drag-release series of events. This mode facilitates operations that involve two different cursor positions. In general, the mouse button press selects the first cursor position and the release selects the second cursor position. When the cursor style is a box, these two positions represent the lower left and upper right corners. When the cursor style is a line segment, the two positions are the endpoints of the line segment. The two cursor positions are returned in **x0pnt**, **y0pnt**, **x1pnt**, and **y1pnt**

map Selects the units of measure in which positions on the graphics display are to be specified. Any form of the **cur** command will return positional information in the units chosen. Likewise, you can set the cursor position in any units. Depending on the application and the nature of the graphics display, one type of units in the following list will be most appropriate.

0 Screen pixels. These are defined relative to the lower left corner of the graphics window occupied by FELIX, and have no relation to the current frame display.

1 Data points. These are defined based on the plot in the current frame. For 1D plots, x will be data points and y will be between zero and one corresponding to the smallest and biggest intensity, respectively. For 2D plots, x and y will both be data point units.

2 Inches. Defined relative to the lower left corner of the current frame.

3 Normalized coordinates. x and y positions are both returned in a range from zero to one. The reserved symbol **ndctyp** specifies the interpretation: **ndctyp**=**0**, defined relative to plot in current frame; **ndctyp**=**1**, defined relative to current frame.

	4	Axis units. Defined relative to the plot in the current frame. This mapping mode will return x and y position in units corresponding to the plot in the current frame. For a 1D plot, the x units correspond to the x-axis units and the y units correspond to the actual data values plotted. For a 2D plot referenced in ppm, both x and y units will be in ppm units
behavior		Varies depending on each mode. See the table below.
exit_style		Selects the style in which to leave the cursor when the **cur** command completes. Independent of *wait_mode* and *behavior*, you can make the cursor appear in one of several styles. The cursor will remain in the selected style until the next cursor operation occurs.
	0	turn cursor off (invisible)
	1	enable default system cursor (usually an arrow)
	2,3	large crosshair cursor
	4	busy cursor (an hourglass or clock face)
	5	small crosshair cursor
	7	vertical half crosshair
	8	horizontal half crosshair

Cursor behaviors:

Mode	*behavior*	action
0	0	get current position
	1	place cursor at position (**x0pnt**, **y0pnt**)
	5*	clear stationary crosshair at (**x0pnt**, **y0pnt**)
	6*	draw stationary crosshair at (**x0pnt**, **y0pnt**)
1	0	enable system cursor
	1	pre-positioned system cursor
	2	crosshair cursor
	3	pre-positioned crosshair cursor
	4**	multiple crosshairs
	5**	pre-positioned multiple cross-hairs
	7	vertical half crosshair
	8	horizontal half crosshair
2	0	full rubber box
	2	pre-positioned box, moves with fixed size
	3	pre-positioned box, resize fixed upper left
	4	rubber line
	5	pre-positioned box, resize with fixed center

* *exit_style* is not used with stationary cursor ** multiple cursors always return axis units	

Symbol dependence

ndctyp Normalized Coordinate Type

x0pnt First X Position

y0pnt First Y Position

x1pnt Second X Position

y1pnt Second Y Position

Symbol changed

keyhit Cursor Key Event Code

x0pnt First X Position

y0pnt First Y Position

x1pnt Second X Position

y1pnt Second Y Position

See also

ena — Enable multiple cursors

dba — Database facility

dba *object operation data*

dba provides access to a database that can be used to store spectral features and relational information. The database is a central part of FELIX, and Chapter 6, *The Database and Tables*, contains detailed information on all aspects of its use.

dbc — Oversampled baseline correction

dbc *decimation_factor fraction*

decimation factor oversampled decimation factor
fraction fraction of data used to compute baseline offset; default = 0.25 (last 1/4 of FID)

dbc removes a DC offset of a digitally oversampled Bruker FID baseline (DC offset creates a spike at the carrier frequency) by subtracting the offset from work. The **dbc** command works properly only if the data at the end of the FID is a baseline. Adequate DC offset correction is usually obtained by using the default fraction value of 0.25.

Symbol dependence

datsiz Number of Data Points

dbl — Double data size

dbl

dbl doubles the size of the data in the workspace by performing a linear interpolation between the existing data points.

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol changed

datsiz Number of Data Points

See also

hav — Halve data size

def — Define a symbol

def *name value*

name name for the defined symbol
value value associated with the symbol name

def is used to explicitly define values for specified symbols. These defined symbols can then be used in macros in the form of *&symbol*. When the *&symbol* notation is encountered within a macro, the symbol's value replaces the symbol's name before command execution.

Here is an example:

```
def count 10
```

In this example, the symbol count is given an explicit number value of 10. Subsequently, if you run a macro containing the following command:

```
for rows 1 &count
```

the explicit number value of count replaces the symbol. In this example, the for loop increments 10 times as defined by the value of the symbol count.

There are two classes of symbols within FELIX: *reserved* symbols and *user* symbols. Reserved symbols have pre-defined meanings in FELIX, whereas user symbols have no pre-defined names or meanings. For a complete list of the reserved symbols and their meanings, see Appendix B, *Symbol Reference*.

Symbol changed

Only the symbol named in the def command

See also

lis — List symbol table

get — Get a symbol value

eva — Evaluate expression and assign to symbol

pur — Purge symbol table

der — Derivative

der

der takes the derivative of the data in the workspace and pushes it onto the current buffer stack. The workspace is left unchanged.

Symbol changed

stack Stack Depth

See also

int — Integrall

dft — Fast Fourier transform of digitally oversampled data

dft *(decimation_factor version)*

dft performs a complex fast Fourier transform on a digitally oversampled Bruker FID in the work. It uses the optional *decimation_factor* and *version* variables to use the correct algorithm. Alternatively, you can omit these two variables but you must set the decim and dspfvs symbols. The symbols correspond to the BRUKER parameters DECIM and DSPFVS, respectively.

The currently supported versions (dspfvs) are 10, 11, and 12. If there will be a newer version then you can enter the 21 phase parameters into FELIX through an ASCII file, which is then read by FELIX when it executes the dft command. The file is located in the

Accelrys/Felix 2002/macros/mac

directory and should be called according to the version number:

```
bphase.dspfvs
```

The **dft** command executes more quickly if the size of the data in the workspace is a power of two, but it will transform data of any size.

If the symbol **gibbs** is set to 1, the first point of the workspace is divided by two before transformation to properly weight the time period this sample represent. If **gibbs** is set to zero, the division is not performed.

Symbol dependence

datsiz Number of Data Points

decim Decimation factor (from BRUKER DECIM variable in acqus)

dspfvs Oversampling version (from BRUKER DSPFVS variable in *acqus*)

gibbs Gibbs Filter Switch

See also

rft — Real Fourier transform

bft — Bruker-Fourier transform

ift — Inverse Fourier transform

dir — Current working directory

dir get *curdir*

dir set *curdir*

dir get returns the current working directory to symbol curdir. If *curdir* is omitted, it displays the current working directory on the status bar.

dir set sets the current working directory as a *curdir* (a valid folder name expected, otherwise it is ignored). Once the current working directory has been changed, any paths relative to ".\" is changed.

dr — Draw work space and stack

dr

dr draws the contents of the workspace (default) as well as the number of buffers indicated by the stack depth symbol stack.

Symbol dependence

absint Absolute Intensity

animat Enable Double Buffering

axtype Axis Type

center Center Zero Switch

cycle Color Cycle

drwbox Draw Box Around Plot

drwclv Draw Contour Levels

drwpks Show Picked Peaks

dspmod Display Mode

first First Point

last Last Point

linpts Line / Points Switch

ovrlap Plot Overlap

pennum Starting Color

pltann Plot Annotations

scale Scale Factor

segint Show Integral

stack Stack Depth

Symbol changed

smalpt Smallest Data Value

bigpt Largest Data Value

disply Current Display Type

See also

exp — Expanded display

ful — Full display

ip — Intensity plot

drb — Display brother cross peaks

drb *peaks color*

peaks cross peak entity
color color to draw brother connectivities

drb draws lines that connect cross peaks having the same parent. **drb** uses the parent pointer element of cross peaks to determine whether two cross peaks

are brothers. The *peaks* parameter can specify either a DBA entity name or a DBA list number.

See also

drx — Display cross peaks

xpl — Make a list of peaks

drx — Display cross peaks

drx *peaks color (matfil) (overlaycolor)*

peaks	cross peak entity
matfil	current matrix file
color	color to draw cross peaks

drx displays cross peaks in the database that are in the current plot region on the current display. If the display is 2D, 2D cross peaks will be displayed. In 3D mode, cross peaks will be displayed as 3D objects. The peak entity can specify either a DBA list number or DBA entity name.

If the color is defined as -1, then the peaks in a 2D plane of a 3D or 4D matrix are colored depending on the relative positions to the current plane. Defining color as -2 colors peaks based on assignment states, and -3 colors the peak boxes based on whether or not they belong to prototype patterns within **Assign**. If color is defined as -4, then the peak set of a different matrix (matfil) will be displayed on the current matrix with color = overlaycolor.

Example:

```
drx &pksent 4
```
(using entity name)

```
drx L2 6
```
(using DBA list number)

Example:

```
drx pksent -4 matfil color
```
(Overlays *pksent* referenced by *matfil* on current matrix using color color)

The relative dimensions (i.e., what dimension of the peak set should be displayed on what dimension of the spectrum) is defined with the repek1, repek2, ... user symbols.

Example:

To display an HSQC peak set (D1 = HN, D2 = N-15) on a HSQC-NOESY spectrum (D1 = HN, D2 = H-1, D3 = N-15) set the following variables:

```
repek1 = 1
repek2 = 3
repek3 = 2
```

Symbol dependence

xpksym Peak Symbol Switch

dst, exd, don, dof — Distributed processing commands

A macro can be distributed through several machines by creating a top level macro which contains a command(s) directing FELIX to use the given machines for distributed processing, preferably at the very beginning of the macro.For example:

```
dst compname number_of_processors first_bundle number_of_
bundles tmp_dir
```

compname	computer (host) name on the network, preferably full name
number_of_processors	number of processors to run the job on — currently only valid for multiprocessor SGI computers
first_bundle	first bundle to process on the given processor
number_of_bundles	number of bundles to process on the given processor
tmp_dir	directory where the temporary files get written — needs to be seen from the *compname* computer and from the current computer

Note that the matrix to be processed should be seen from the computers where the distributed processing will be happening (through NFS).

This command can be run on multiprocessor SGI's in distributed parallel mode; that is, each processor runs its own copy of FELIX.This top level command calls the macro to distribute using the command:

```
exd macro arg1 ... argN
```

This executes a macro in distributed mode and then returns to the current macro, continuing with the line immediately following this exd command. The called macro is deleted once it completes, and the current calling macro is not disturbed.

The macro for distributed processing should contain a command from which to start the distributed processing:

```
don
```

The macro after the don command should contain processing instructions using the bundle mode.

At the very end of the macro before the return statement (ret and end) there should be a command to finish the distributed processing:

```
dof
```

Executing this macro results in distributing the bundles along that dimension through the machines defined by the dst command, while the original FELIX process returns to the calling macro or prompt. Certainly you can set the current machine as one of the target machines or even as the only target machine

(which is effectively the background processing), but that is a separate FELIX process.

Caveats:

- The original process should not alter the matrix until the distributed processing is done. Thus, the macro should be carefully written not to start for example, a second dimension processing, until the first is finished.
- There should be no **ex** (**exr**, **cal**) command within the distributed macro.
- The temporary directories should be accessible from the computer where the original FELIX runs (for example, *ls* should work).
- As with all distributed programs, higher processing / disk I / O ratio equals better efficiency; do not use this command for only Fourier transforming a small data set, but rather use it for a big 3D or 4D processing with LP.
- This processing can be used only after the data in the FELIX matrix format; that is, generally it is not applicable to the first dimension processing, unless it is imported into a FELIX matrix format before starting the transformation.
- A file is written to the temporary directory at the end of the processing called *felix_out*.

eif — Macro end of block if

eif defines the end of a block-form macro if statement. See the if documentation for a complete description of the if / else syntax.

els — Macro else block

els defines the beginning of the else portion of an if / else statement. See the if documentation for a complete description of the if / else syntax.

em — Exponential multiply

em *line broadening*

line broadening line broadening in Hz (optional)

em multiplies the data in the work space by an exponential window. This apodization function is used to reduce noise at the expense of spectral resolution. The parameter line broadening may be entered with this command or, if no line broadening parameter is specified, the current value of the line broadening symbol (*lbroad*) is used. The *swidth* parameter must be set correctly; otherwise, the window function will be improper.

Symbol dependence:

datsiz Number of Data Points

datype Data Type

lbroad Line Broadening

swidth Spectral Width in Hz

ena — Enable multiple cursors

ena *frame on/off*

frame graphics frame number
on/off cursor enable state for specified frame: **0**=disabled, **1**=enabled

ena enables a correlated cursor in a single frame, based on the current graphics context from the current plot. If multiple frames have correlated cursors enabled, the **cur** command can be used to enable correlated cursors that appear in all enabled frames in corresponding axis unit positions. Any new graphics display in a frame with a correlated cursor enabled will automatically disable the correlated cursor since the display context changed.

See also

cur — Cursor controll

end — Macro end statement

end terminates macro execution and returns to command mode. **end** is only valid in macros.

env — Get a system environment variable

env *env_var symbol*

env_var the name of one environment variable
symbol symbol to receive current value of environment variable

env allows the user to read the system environment variables from inside FELIX. If *env_var* exists in the current process's environment, then symbol receives the value of that environment variable. If the environment variable does not exist, the symbol is left unchanged.

err — Macro branch on error condition

err *label macro_label*

err *macro macro_file_name*

macro_label	target label in macro
macro_file_name	name of macro to be executed

err defines an error trap response for a macro. Once an **err** command has been executed by a macro, any subsequent error condition will cause a branch to the specified macro label, or execution of the specified macro. The **err** destination is valid for the duration of the macro. **err** is only valid in macros.

esc — Test for escape key event

esc *symbol*

symbol	symbol to receive escape event status: **0**=escape key has not been pressed, **1**=escape key has been pressed

The **esc** command lets you watch for an interrupt request inside a macro. When placed in a macro for loop **esc** can be used as a way to exit the processing macro before normal completion.

> **Caution:** Be warned that there are several commands that also intercept escape key events, and may deal with the escape event before the **esc** command is encountered during macro execution. These commands include **cp**, **ip**, **cur**, **obj**, and other commands that enable a cursor. These commands will update the symbol keyhit to the value <Esc> when they trap an escape event.

eva — Evaluate expression and assign to symbol

eva *symbol_name expression num_of_characters*

symbol_name	target symbol for expression value
expression	arithmetic expression in parentheses
num_of_characters	number of characters used to define the symbol name

eva evaluates an arithmetic expression and places the formatted result into the value field of the symbol. The parameter expression must be enclosed in parentheses, for example:

```
eva count (&row*5)
```

In this example, the symbol count is given a value that is equal to five times the current value of the symbol row. The capabilities of the **eva** command and the syntax of expressions are described explicitly in Appendix B, *Symbol Reference*.

Note: There are no spaces allowed between the parentheses.

See also

def — Define a symbol

get — Get a symbol value

cdf — Conditional define

ex — Execute a macro

ex *macro arg1 ... argN*

file_name macro to execute (optional — executes current macro)

ex causes control of FELIX to change from interactive to macro execution mode. During execution of a macro, FELIX no longer accepts input from the user keyboard. Upon termination of macro execution, FELIX returns to the user and interactive control is returned to the keyboard (the **>** prompt appears on the screen and signifies the end of macro execution). Macros may also be executed with the *file_name* parameter.

Within a macro, the **ex** command causes another macro to be executed. All traces of the first macro executed are removed from memory. To call a second (or third) macro as a subroutine and return to the current macro, the macro call command (**cal** *$macro_name*) or execute and return command (**exr** *macro*) must be used within the parent macro. For a full discussion of macros, see Chapter 4, *Macros*.

Arguments may be passed to the macro. See Passing arguments to macros for more information.

Symbol changed

macfil Current Macro File

See also

cal — Macro call

exm — Execute Multiple Macros

exr — Execute a macro and return

go — Macro unconditional branch

exc — Exchange real and imaginary

exc

exc causes the real and the imaginary parts of the work space to be exchanged. This command is most often used when you must merge the real component

of a real T1 FID with the real component of an imaginary T1 FID (i.e., hypercomplex data acquisition. See States 1983).

Symbol dependence

datsiz Number of Data Points

datype Data Type

exm — Execute Multiple Macros

exm *macro arg1 ... argN*

exm executes a macro while preserving the existing mscro memory and context. **exm** may be invoked from anywhere in FELIX : the command line, the menu interface, or a macro. The executed macro is deleted after it completes the job.

Arguments may be passed to the macro. See Passing arguments to macros for more information.

See also

exr — Execute a macro and return

ex — Execute a macro

exp — Expanded display

exp

exp generates an expanded display of the workspace using the current 1D plot limits (first, last).

Symbol dependence

first First Data Point

last Last Data Point

See also

ful — Full display

dr — Draw work space and stack

ip — Intensity plot

exr — Execute a macro and return

exr *macro arg1 ... argN*

Execute a macro and then return to the current macro, continuing with the line immediately following this **exr** command. The called macro is deleted after it completes, and the current calling macro is not disturbed. Contrast this with **ex** and **cal**.

Arguments may be passed to the macro. See “Passing arguments to macros” in Chapter 4, *Macros* for more information.

See also

ex — Execute a macro

cal — Macro call

ret — Macro subroutine return

fit — Fit 1D peaks

fit

fit invokes a line fitting subsystem of FELIX for optimizing peak parameters to yield a least squares fit to a spectrum. Peaks may be pre-picked using **pic** or created within **fit**. Single peaks may be added, removed, or manually edited.

Three independent optimization algorithms are provided within **fit**. These are simplex, quasi-Newton, and simulated annealing. You can select an algorithm and determine the set of peak parameters to be optimized. If you save the fitted peaks, the 1D peaks entity is updated with the new fitted values.

fit *peak_entity volume_entity slot_number optimize_center optimize_widths optimize_heights*

This command fits ND peaks in the temporary entities *peak_entity* and *volume_entity* using quasi-Newton minimization of the requested parameters (*optimize_center= 1*, *optimize_widths = 1*, and / or *optimize_heights = 1*).

Symbol dependence:

datsiz Number of Data Points

picent 1D Peaks Entity

See also

pic — Peak pick and label

ssp — Synthesize spectrum from peak list

flf — FaceLift baseline correction

flf baseline *filter_width num_std_dev smooth_pnts buff_no*

flf correct *filter_width num_std_dev smooth_pnts*

flf smooth *filter_width buff_no*

flf provides functions for baseline correction of the data in the work space, using the FaceLift algorithm published by Chylla, R. A. & Markley, J. L. (J. Mag. Reson. Series B 102, 148-154, 1993).

flf baseline performs FaceLift baseline modeling on the data in the workspace. It identifies the baseline points and generates a model baseline. The generated baseline either overwrites the workspace (if buff_no is 0) or is stored in a buffer while the workspace remains unchanged (if buff_no is greater than 0).

flf correct performs FaceLift baseline correction on the data in the workspace. It is very similar to subcommanded flf bas, except that the baseline is directly subtracted from the original spectrum in the workspace.

flf smooth performs a (2 * filter_width + 1)-point binomial filtering on the data in the workspace. The smoothed data are stored in buffer buff_no.

filter_width	Half-width of the smoothing data window over which data points are sampled. Recommended range is 32-64 data points (powers of 2 are not necessary).
num_std_dev	Used to determine a threshold standard deviation, above which any point is considered to be a signal point. Recommended range is 2.5-3.0.
smooth_pnts	Half-width of the smoothing data window over the resulting baseline. Normally the same as *filter_width*, but can be changed.
buff_no	Number of the work buffer that stores the generated baseline. Default or 0 means to overwrite the workspace.

See also

abl — Automatic baseline flattening

abq — Automatic selection of baseline points

bas — Baseline points manipulation

csp — Cubic spline baseline correction

flt — FLATT baseline flattening

smo — Binomial smooth

fli — Frequency list manipulation

The **fli** command manipulates frequency lists. These lists provide a succinct method of storing chemical shift values that are believed to arise from a single

residue. Chemical shift values may be added, deleted, filtered, or displayed from the frequency lists by using these commands. After examining a particular residue, the frequency lists associated with that residue may be written to a database frequency list entity for future use. Conversely, one can read chemical shift values into a frequency list from a database frequency list entity. FELIX provides eight frequency lists and allows storage of up to 64 chemical shift values in each list.

fli clear *list*

The **clear** subcommand removes all values from the specified frequency list.

fli add *list freq freq ...*

The **add** subcommand appends chemical shift values to the frequency list. Any number of chemical shift values may be specified on the command line although each frequency list stores a maximum of 64 values.

fli list *list*

The **list** subcommand lists (on the screen) the chemical shift values stored in the specified frequency list.

fli delete *list lo_freq hi_freq*

The **delete** subcommand removes chemical shift values that are greater or equal to *lo_freq*, and less than or equal to *hi_freq*, from the specified frequency list.

fli union *list1 list2 list3*

The **union** subcommand combines frequency lists one and two into frequency list three. Unique chemical shift values are transferred to frequency list three from either of the source lists. In the case of two equal frequencies in each source list, **union** adds just one instance of the chemical shift value to frequency list three.

fli draw *D1list D1color D2list D2color obj#*

The **draw** and **object** subcommands store graphical representations of frequency lists one and two in the specified object. The chemical shift values in frequency list one appear as vertical lines of the requested color and the values in list two appear as horizontal lines of the requested color. The *obj#* parameter determines the destination for the drawn frequency lists. When *obj#* is positive, the frequency lines are placed into a graphical object for later display. When *obj#* is zero, the frequency lines are drawn immediately on the current 1D or 2D plot. If the D1color or D2color is 0 then it erases the displayed frequencies. If the *D1color* or *D2color* is -1 then each frequency in the list will have a different color.

fli object *D1list D1color D2list D2color obj#*

See **draw** (dr — Draw work space and stack).

fli tile *D1list D2list tile_width tile_entity*

The **tile** subcommand uses the chemical shift values in frequency lists one and two, as well as the *tile_width* factor, to create a tile entity with the name *tile_entity*.

Symbol dependence

The **fli draw** *D1list D1color D2list D2color obj#* command affects the following user symbols:

flid1 Current frequency list number on D1, set to the value of D1list.

flid2 Current frequency list number on D2, set to the value of D2list.

flicol1 Color to display frequencies on D1, set to D1color.

flicol2 Color to display frequencies on D2, set to D2color.

See also

cp — Contour plot

ip — Intensity plot

til — Tile plot

fli read *list freq_entity*

The **read** subcommand copies the chemical shift values in the database frequency list entity *freq_entity* to the specified frequency list.

fli write *list freq_entity*

The **write** subcommand copies the chemical shift values in the specified frequency list to the database frequency entity *freq_entity*.

fli move *list1 list2*

The **move** subcommand copies the chemical shift values in *list1* to *list2*.

fli equiv *list resolution*

The **equiv** subcommand removes nearly equal chemical shift values from a specified frequency list and replaces the two values with their average. Two chemical shift values are considered to be nearly equivalent when the difference between their values is less than *resolution*.

fli xpks *list peaks dimen resolution*

The **xpeaks** subcommand creates a frequency list from a list or entity of cross peaks. A frequency is made at each center along the given dimension, then all frequencies closer than *resolution* are combined.

fli peak *list peaks resolution*

The **peak** subcommand creates a frequency list from a list or entity of 1D peaks. A frequency is made at each center, then all frequencies closer than *resolution* are combined.

fli an *list freq name*

The **an** subcommand appends chemical shift value and names to the frequency list one by one. Each frequency list stores a maximum of 64 values and names.

fli rc *list clipboard_entity*

The **rc** subcommand copies the chemical shift values and names in the assign database frequency clipboard entity *clipboard_entity* to the specified frequency list.

fli rn *list pattern_entity pattern_# spectrum_id*

The **rn** subcommand copies the chemical shift values and names in the assign database pattern entity *pattern_entity* to the specified frequency list. You have to specify which pattern's (*pattern_#*) chemical shifts to copy and whether to use the generic shifts (*spectrum_id* = 0) or spectrum specific shifts (in that case the *spectrum_id* should reflect that experiment's number).

fli rp *list protopattern_entity proto_#*

The **rp** subcommand copies the chemical shift values and names in the assign database protopattern entity *protopattern_entity* to the specified frequency list. You have to specify which protopattern to copy (*proto_#*).

fli collect *list pattern_entity spectrum_id centrum delta*

The **collect** subcommand collects frequencies from the assign database pattern entity into the specified frequency list. All frequencies will be copied which generic (*spectrum_id* = 0) or spectrum specific shifts (*spectrum_id* =n) are within *delta* ppm from the specified *centrum* position.

fli sort *list order*

The **sort** subcommand will sort the content of the specified frequency list in descending (*order*=0) or ascending order (*order*=1) by the chemical shifts.

flp — Low-point fold of work space

flp

flp performs a low-point fold on the data in the workspace by saving the lower value of the two symmetrical points (for data that has a size of 1024 points, point 1 is compared with point 1024; point 2 with point 1023; point 3 with point 1022; etc.). The command **flp** is similar to the fold data command (**fol**) and can be used if you know that the workspace contains symmetric data. By performing a low-point fold on the workspace, the size is decreased by a factor of two. This command is convenient for non-diagonal symmetrization of multidimensional spectra.

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol changed

datsiz Number of Data Points

See also

fol — Fold work space in half

unf — Unfold work

flt — FLATT baseline flattening

flt *baseline_width chimin number_of_points tau*

baseline_width	baseline segment width
chimin	minimum chi-square value
number_of_points	number of points to correct
tau	baseline cutoff factor

This command performs FLATT baseline correction on the data in the work space. **flt** automatically identifies the segments of data in the work space that constitute the baseline. The command fits a Fourier synthesized curve to the baseline points using a linear-least-squares fit, based on a singular value decomposition, and subtracts the synthesized curve from the data in the work space. The FLATT baseline correction was introduced by Güntert and Wüthrich in 1992 (please see the FELIX *User Guide* for the full reference).

flt uses the *baseline_width*, *chimin*, and *tau* parameters to find baseline segments in the spectrum. The *baseline_width* parameter determines the width of a sliding window that **flt** uses to identify baseline segments. **flt** moves the window point-by-point along the length of the spectrum, fitting the spectral data points within the window to a straight line, calculating the chi square value of the fit, and storing the chi-square value in a vector. Next, a narrower window of width 2/3*basline_width* is moved point-by-point along the vector of chi-square values. If the smallest chi-square value within the window is less than the product of the parameters *chimin* and *tau*, **flt** considers the point in the center of the window to be a baseline point. The *number_of _points* parameter determines the number of terms used in the Fourier synthesis.

Symbol dependence

chi Minimum Chi-square Value

See also

chi — Calculate minimum chi-square value

flf — FaceLift baseline correction

fol — Fold work space in half

fol

fol performs a linear symmetrization of the 1D workspace by co-adding the first and last points together; second and next-to-last points; etc., until a "new" symmetrized, 1D spectrum is created. By performing a co-addition fold on the workspace, the size of it is decreased by a factor of two. This command is convenient for non-diagonal symmetrizations of multidimensional spectra.

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol changed

datsiz Number of Data Points

See also

flp — Low-point fold of work space

unf — Unfold work

for — Loop for macros

for *symbol start end step*

symbol	name of the **for** loop counter
start	beginning value of the **for** loop counter
end	ending value of the **for** loop counter
step	increment **for** loop counter (may be negative)

for is the FELIX macro loop operator, which acts in a manner very similar to the BASIC computer language **FOR** command. The **for** command may only be used in macros. Within a macro, the **for** command allows all of the commands between itself and the macro command **nex** to be executed, incrementing the value of *symbol* by *step* each cycle through the loop. The **for** command is mainly used to loop through rows and columns of matrices while performing multi-dimensional transforms. Like the BASIC **FOR** command, FELIX **for** loops may be nested within each other as long as each **for** loop has a **nex** command defining the end of the loop.

fpo — Pop FELIX window

fpo

With this command, you can pop the FELIX window on top of any window residing in the same position on the screen. This command is useful if you use

FELIX with Insight II, and wish to bring FELIX into the foreground from within a macro.

fpu — Push FELIX window

fpu

With this command you can push the FELIX window to the background, i.e., behind any window residing in the same position on the screen. This is useful if you are using FELIX with Insight II, and want to push FELIX into the background from within a macro.

fra — Manipulate graphics frames

fra *op parameters*

fra allows you to define a number of graphics frames in the FELIX window and to direct graphics to these frames. Each frame maintains its own complete graphics context, which includes the 1D workspaces, all user and reserved symbols, and matrix. Only one frame is current at a time. For example, displaying the workspace in the current frame modifies only the graphics context for the current frame. All other frames will maintain their original graphics context.

The frame operators and their functions are as follows:

fra open *X_origin Y_origin X_size Y_size*

fra open *-1*

fra open opens a new frame. Origins and sizes are given in pixels. A newly opened frame becomes the current frame and inherits the graphics context of the previously current frame. Specifying the *x_origin* as -1 opens a default size and position frame.

fra zero

fra zero closes all frames and re-initializes the frame manager.

fra front *frame#*

fra front pops any frame to the front of all other frames. The specified frame also becomes the current frame. Specifying a *frame#* of -1 allows the desired frame to be identified by clicking anywhere within the frame.

fra close

fra close closes the current frame. The graphics context of the closed frame is discarded. The next-most-current frame then becomes the current frame.

fra move *X_origin Y_origin*

fra move moves the current frame to a new position with its origin at the position specified by the pixel parameters *x_origin* and *y_origin*. Specifying an *x_origin* of -1 allows you to move the frame using the mouse.

fra resize *X_origin Y_origin X_size Y_size*

fra resize allows the current frame to be re-sized and re-positioned anywhere within the FELIX window. The origin and sizes are in pixel units. Specifying the *x_origin* as -1 allows manual re-sizing of the current frame using the mouse.

fra verify *frame# symbol*

fra verify allows you to verify or inquire about the existence of a frame by number. If the specified frame exists, the value 1 is stored in *symbol*. A value of 0 is stored in *symbol* if the frame does not exist.

fra header *frame# header text*

fra header allows you to label the frame header with a text string. The text should be enclosed within single quotes if you wish to preserve upper case characters and blanks.

fra who *symbol symbol2 symbol3*

fra who allows you to inquire which frame is current. The frame number of the current frame is stored into *symbol*. A zero will be stored in *symbol* if no frames exist. The optional *symbol symbol2* amd *symbol3*, if present, receive the count of opened frames and the maximum ID of them.

fra 3D_reset

fra 3D_reset resets the 3D viewing parameters. By default, the 3D display interface does not reset the rotation, translation, scaling, and clipping parameters. This allows you to maintain the exact view of a 3D object between plots. This frame operator resets all the 3D display parameters to initial values, just like the **set** button in the 3D display interface.

fra xpand

fra xpand expands the current frame to cover all other frames. This enhances the pixel resolution to show more detail. The current plot is automatically redrawn.

fra unexpand

fra unexpand restores an expanded frame to its previous size and location. The current plot is automatically redrawn.

fra j

fra j iconifies the current frame. The current plot is automatically redrawn.

fra k

fra k restores an iconified frame to its previous size and location. The current plot is automatically redrawn.

fra n *x0 y0 x1 y1*

fra n returns the current coordinates of the active frame.

All the above **fra** operations act upon the single current frame. Recall that each frame has its own complete graphics context including graphics, symbol values and data. For example, when you change to a different current frame, the 1D workspace is overwritten with the new frame's copy of the workspace contents.

The following **fra** operators facilitate the transfer and sharing of context between pairs of frames:

fra export *frame#*

fra export exports the entire context of the current frame to another frame. The current frame does not change and its context is not affected.

fra import *frame#*

fra import imports the entire context from another frame into the current frame. The graphics context, symbols and data are changed in the current frame, which becomes an exact copy of the specified frame's context.

fra symbol *frame# mode symbol_name local_symbol*

mode **0**=transfer from another frame, **1**=transfer to another frame

fra symbol is used to pass a single symbol value between the current frame and another frame. This operation is frequently used when working with correlated displays of corresponding regions of different spectra.

The *mode* parameter allows transfer of context symbols either into or out of the current frame. When *mode* is zero, the value of *symbol_name* is transferred from the other frame *frame#* to the symbols *local_symbol* in the current frame. When *mode* is one, the value of *symbol_name* is transferred to the symbol of the same name in the other non-current frame *frame#*.

The following **fra** operators facilitate the connection of frames-using this command you can assure that navigating in one frame will update the plot limits in other frames:

fra q

fra q cleans up the frame connections. It is necessary to call this if you need to start a new frame connection setup.

fra add *frame#*

fra add adds a frame to the frame connection list, subsequently the **fra l(ink)** command should be called to set up the dimension connections.

fra link *frame_n dimension_a frame_m dimension_b*

Link *dimension_a* of *frame_n* to *dimension_b* of *frame_m* after both *frame_n* and *frame_m* were defined as linked by the **fra a(dd)** command. If *dimension_a* or *dimension_b* are less than or equal to zero, the connection along that dimension gets broken. If both are less than or equal to zero, the link gets cleaned up.

fra y *frame_n dimension_a frame_m dimension_b*

Define a jump from *dimension_a* of *frame_n* to *dimension_b* of *frame_m* after both *frame_n* and *frame_m* were defined to be linked by the **fra a(dd)** command.

If later the **fra y** command without parameters is called from frame *frame_n* then you can click a position in *frame_n* and that will define a new plane position in *frame_m* along *dimension_b*.

fra y

Execute the jump in the current frame if previously the current frame was put on the connection list via **fra a** command and the jump was defined using the **fra y** command.

fra 0

Temporarily disables frame connection

fra 1

Restores frame connection if it was defined before.

ft — Fast Fourier transform

ft

ft performs a complex fast Fourier transform on the contents of work. The **ft** command runs faster if the size of the data in the workspace is a power of two, but it will transform any size of data.

If the symbol **gibbs** is set to **1**, the first and last point of the workspace are divided by two before transformation to properly weight the time period these samples represent. If gibbs is set to **0**, the division is not performed.

Symbol dependence

datsiz Number of Data Points

gibbs Gibbs Filter Switch

See also

dft — Fast Fourier transform of digitally oversampled data

rft — Real Fourier transform

bft — Bruker-Fourier transform

ift — Inverse Fourier transform

hft — Hilbert transform

ful — Full display

ful

ful draws the entire 1D spectrum in the current frame. **ful** is the opposite of the **exp** command.

Symbol changed

First First Data Point

last Last Data Point

See also

exp — Expanded display

ip — Intensity plot

fxp — Filter cross peaks

fxp *peaks diagonal tolerance confirm*

fxp *peaks doublets tolerance*

fxp *peaks asymmetric tolerance confirm*

fxp *peaks width min_width max_width confirm*

fxp provides operators for various types of cross peak filtering. **fxp** is used after picking peaks to remove unwanted peaks and eliminate redundant peaks.

Symbol dependence

hafwid Cross Peak Half Width Factor

See also

pic — Peak pick and label

fxp *peaks diagonal tolerance*

peaks	cross peak entity to filter
tolerance	tolerance in data points
confirm	wait for user confirm (1) or not (0)
units	optional unit parameter — points (0), Hz(1) or ppm (2)
type	diagonal type — body (0), D1D2 (1), D1D3 (2), D2D3 (3), D1D4 (4), D2D4 (5), D3D4 (6)

fxp *confirm (units type)*

This operation removes all diagonal peaks from a square matrix or from a non-square matrix if the *units* parameter is set to ppm and on of the *type* parameters is set. If the peak center along each dimension is within *tolerance* points of centers for that peak in all other dimensions, the cross peak is considered to be on the diagonal and will be deleted.

fxp *peaks doublets tolerance*

peaks	cross peak entity to filter
tolerance	tolerance in data points

This operation combines multiplet peaks into single peaks. If any two peak centers are within *tolerance* points of each other along all dimensions, the two peaks are combined into one larger peak with a footprint that covers all the multiplet footprints.

fxp *peaks asymmetric tolerance confirm (units type)*

peaks	cross peak entity to filter
tolerance	tolerance in data points
confirm	wait for user confirm (1) or not (0)
units	optional unit parameter — points (0), Hz(1) or ppm (2)
type	symmetrization type — D1D2 (0), D1D3 (1), D2D3 (2), D1D4 (3), D2D4 (4), D3D4 (5)

This operation removes asymmetric peaks from a 2D square matrix. If the optional *units* and *type* parameter is specified the matrix can be a non-symmetric 2D, 3D or 4D. For every peak in the *peaks* entity, a search occurs for a peak located within *tolerance* points of the symmetrical position. If no peaks are found within this distance, that cross peak has no symmetric partner and is deleted from the *peaks* entity.

fxp *peaks width min_width max_width confirm*

peaks	cross peak entity to filter
min_width	minimum width in data points
max_width	maximum width in data points
confirm	wait for user to confirm (**1**) or not (**0**)

This operation removes peaks from the database if their width is less than the minimum width (*min_width*) parameter or greater than the maximum width (*max_width*) parameter.

get — Get a symbol value

get *prompt_string symbol_name*

prompt_string	prompt requesting for user input
symbol_name	name of symbol that will receive the input string

get is used to prompt for data from within a macro. The command **get** issues the specified prompt (entered as the *prompt_string*) when executed in the

macro, then waits for you to enter a value for the defined *symbol_name*. The value of the specified symbol is set to the data value entered. Enclose the *prompt_string* in single quotes if you wish to preserve uppercase letters and spaces.

See also

def — Define a symbol

lis — List symbol table

gf — Generate FID

gf *amplitude frequency tau*

gf allows you to generate an FID containing as many lines as you wish. The *amplitude* is in arbitrary units. The *frequency* is in Hz, and the *tau* is the time constant of the decay in seconds, or (1/linewidth in Hz). If you let the program prompt you for input, you can continue adding lines until you are done, then terminate **gf** with a zero amplitude.

Symbol dependence

datsiz Number of Data Points

datype Data Type

See also

gsp — Generate spectrum

gif — Macro arithmetic goto

gif *expression label1 label2 label3*

gif branches to a label depending on the numeric value of *expression*. If *expression* is negative, zero or positive, **gif** branches to *label1*, *label2*, or *label3* respectively. This command is valid only in macros.

gm — Gaussian/Lorentzian window

gm *broadening coefficient*

broadening	line broadening in Hz
coefficient	Gaussian coefficient

gm acts in the same manner as the **gm** command found on Bruker spectrometers. The command **gm** is generally used to enhance resolution at the expense of sensitivity. If the *broadening* and *coefficient* parameters are not entered, the reserved symbols for line broadening (**lbroad**) and Gaussian parameter

(**gbroad**) are used. Typically, a negative *broadening* parameter is used with the *coefficient* parameter of 0.2 to give a Gaussian component to the line shape.

Symbol dependence:

swidth Spectral Width in Hz

datsiz Data Size

datype Data Type

gmh — Gaussian multiply in Hz

gmh *broadening*

broadening Gaussian broadening in Hz (optional)

gmh multiplies data in the work space by a Gaussian window. Gaussian apodization, rather than Lorentzian multiplication (**em**), is useful in processing data whose inherent line shapes are Gaussian as is the case for most solid state NMR spectra. This command does not introduce apodization errors that will adversely affect the data in later quantization, fitting, or deconvolution, because it does not change the line shape from Gaussian to a Gaussian-Lorentzian mixture (Gladden 1986). The **gmh** command may also be used in constructing customized apodization function for two-dimensional processing.

> **Note:** Do not confuse the **gmh** command with the **gm** command, which applies to a separate function.

go — Macro unconditional branch

go *macro label*

macro label label telling the **go** command where to branch to

go is used within macros to perform an unconditional branch to the specified label.

gre — Greek text annotation

gre *x0 y0 (z0 a0) (fix) (anchor) number text*

x0	x-coordinate for text placement
y0	y-coordinate for text placement
z0	optional z-coordinate for text placement
a0	optional a-coordinate for text placement
fix	scaling of text — fixed (0), according to Y size of plot (1), according to X size of plot (2), according to both sizes (3
anchor	centering — left justify (0), center point (1), right justify (2)
number	number of characters of text

text text to be made in Greek characters

gre draws text with its origin at (*x0*, *y0*) (optionally *z0* and *a0* if strip plot of a 3D or 4D matrix). The coordinates are interpreted based on the symbol **annunt**. All 24 Greek characters can be drawn, in both upper and lower case. The text is translated from the Roman to Greek alphabets as shown below:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Α Β Χ Δ Ε Φ Γ Η Ι ϑ Κ Λ Μ Ν Ο Π Θ Ρ Σ Τ Υ ς Ω Ξ Ψ Ζ

a b c d e f g h i j k l m n o p q r s t u v w x y z

α β χ δ ε ϕ γ η ι φ κ λ μ ν ο π θ ρ σ τ υ ϖ ω ξ ψ ζ

Symbol dependence

anncol Annotation Color

annunt Annotation Units

annang Annotation Angle

annsiz Annotation Text Size

slant Annotation Text Slant

thick Text Thickness

See also

tex — Text annotation

ann — Annotate plot

gto — Macro case goto

gto *expression label0 label1 ... labeln*

gto branches to the n^{th} label based on the value of the expression in the range 0:n. If the expression value is less than zero or greater than n, no branch will occur. This command is valid only in macros.

gsp — Generate spectrum

gsp *intensity frequency Lorentzian Gaussian*

intensity	intensity of FID to generate
frequency	frequency of generated spectrum
Lorentzian	Lorentzian line width
Gaussian	Gaussian line width

gsp generates an FID from the input parameters. The **gsp** command prompts you for the parameters of a set of lines and then generates a spectrum with these lines. Parameters include intensity, frequency, Gaussian, and Lorentzian line widths. If both Gaussian and Lorentzian line widths are specified, a Voigt

line shape is computed. **gsp** adds the lines to the data in the work space, so the work space should be set to zero before you use this command. **gsp** generates only real data, therefore if you need a complex spectrum, a Hilbert transform (**hft**) must be performed on the FID.

gv — Get value

gv *point real symbol_name imaginary symbol_name*

point	data point number
real symbol_name	name of symbol to receive real part
imaginary symbol_name	name of symbol to receive imaginary part (optional)

gv gets the value of the specified data point and assigns the value to the defined symbol(s). If the point number is outside the range of 1 to **frsize** (maximum frame size), no action is taken. The symbol for the imaginary part is optional, and will be ignored in the case of real data.

Symbol dependence

datype Data Type

See also

pv — Put value

hav — Halve data size

hav

hav halves the size of the workspace by averaging successive pairs of points. The **hav** command works in a manner opposite the double data size command (**dbl**)

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbol changed

datsiz Number of Data Points

See also

dbl — Double data size

hcp — Hard copy plot

hcp

hcp preview

hcp setup

The printing of FELIX 2002 is WYSIWYG, so the parameters that affects spectral display on the screen also affects the printout. The parameters used in the previous versions that only controlled hardcopy, such as **hardmo**, **harddv, papwid**, **paphgt**, **hardx0**, **hardy0**, **hardxs**, **hardys**, **pltorg**, and **orient**, are all obsolete.

hcp setup displays the standard **Print Setup** dialog for you to select printers, set page properties, etc.

hcp preview displays the standard **Print Preview**.

hcp displays the standard Print dialog box and submit the print job if you click **OK**.

> **Note:** Depending on which frame is the active, hcp applies to both spectral display and spreadsheet display.

See also

dr — Draw work space and stack

ip — Intensity plot

cp — Contour plot

sp — Stack plot

sca — Scale factor for dimension

hft — Hilbert transform

hft *mode*

hft performs a Hilbert transform on real data in the workspace by converting a real vector into a complex vector. It is important that the number of points in the workspace be power of two. The Hilbert transform generates the imaginary (dispersive) part of the spectrum from the real vector and is useful for phase correcting frequency-domain data that has no imaginary part. Once **hft** generates the imaginary data, you may use the inverse FFT (**ift**) to transform the data into time-domain data. Many useful processing tricks can be performed only in the time domain, thus the Hilbert transform is useful for converting real data to time-domain form.

If the *mode* parameter is set to **1** then a different and more precise algorithm is used (Ernst 1969, Zolnai 1990, Rance private communication).

Symbol dependence

datsiz number of data points

See also

ft — Fast Fourier transform

bft — Bruker-Fourier transform

ift — Inverse Fourier transform

rft — Real Fourier transform

idf — Is defined

idf *test_symbol ans_symbol*

test_symbol	symbol name to test for definition
ans_symbol	symbol name to receive the result of test

The **idf** command tests to see if a symbol is defined. If *test_symbol* is defined, then *ans_symbol* is set to **1**. Otherwise, the value is **0**.

See also

cdf — Conditional define

def — Define a symbol

pur — Purge symbol tables

if — Macro if conditional branch

if *expression1 operator expression2 macro_label*

expression1	first expression for content comparison
operator	relational operator defining how expressions relate
expression2	second expression for content comparison
macro_label	branch point target when the if statement is true

if compares *expression1* and *expression2* according to the relational operator, defined by *operator*. Two classes of relational operators are provided; one compares expressions as a string of characters, while the other first converts the expressions to real numbers. If the condition is true, control transfers to the specified macro label; if it is false, no branch occurs. There are 12 legal relational operators:

Numeric Operator	String Operator	Description
eq	eqs	equal to
ne	nes	not equal to
lt	lts	less than
le	les	less than or equal to

gt	gts	greater that
ge	ges	greater than or equal to

Advanced if conditionals

1. **Sub-string Specification**: For string comparisons, there is a syntax for specifying a substring on which to base the comparison. To specify a substring, append (***first_char*:*last_char***) to the end of *expression*, with no intervening spaces. This means to consider only the characters beginning with *first_char* and ending with *last_char* in the comparison.
2. **Compound Conditionals**: All forms of the **if** command also support compound conditionals. You can combine two or more conditionals in one **if** command by separating them with the keywords **and** or **or**.

Example:

if *expression op expression* and *expression op expression* ...

if *expression op expression* or *expression op expression* ...

Macro if conditional block execution

if *exp op exp* then

{block of commands}

eif

The block **if** evaluates the condition as described above. If the condition is true, all commands in the block up to the **eif** are executed. If the condition is false, execution skips to the command following the **eif**.

if *exp op exp* then

{true block of commands}

els

{false block of commands}

eif

The **if/then/els** evaluates the condition as described above. If the condition is true, the block of statements between **then** and **els** is executed, then execution skips to the command following the **eif**. If the condition is false, the block of commands between the **els** and **eif** commands is executed.

Due to block **if**s and compound conditionals, the reserved keywords **then**, **and**, and **or** may not be used as labels in an **if** command.

ift — Inverse Fourier transform

ift

ift performs an inverse Fourier transform; it transforms a complex frequency-domain spectrum into a complex time-domain FID.

Symbol dependence

datsiz Number of Data Points

gibbs Gibbs Filter Switch

datype Data Type

See also

dft — Fast Fourier transform of digitally oversampled data

ft — Fast Fourier transform

rft — Real Fourier transform

bft — Bruker-Fourier transform

inq — Inquire If file exists

inq *ext file_name symbol*

ext	extension of *file_name*; defines prefix
file_name	file name to look for, without extension
symbol	symbol to set to exist status

inq is used to determine if a specified file already exists or not. **inq** sets *symbol* to zero if the file does not exist and to one if the file does exist. When *ext* is a known file extension (**dat**, **mat**, **mac**, **dba**, **ann**, **mnu**), the corresponding prefix and extension will be used. When the extension is null, no prefix or extension will be used.

Symbol changed

The symbol specified as a parameter is set to zero or one.

ins — Insight II-FELIX inter-process communication

ins *keyword rest_of_command_line*

ins initialize

The **ins initialize** command initializes the FELIX server.

ins connect

The **ins connect** command connects FELIX to Insight II.

ins check

The **ins check** command checks to see if the RPC connection is currently up

ins command *rest_of_command_for_Insight*

ins command passes the command string to Insight II, substituting the atom specification from the NMR spec into an Insight II spec.

ins command2

ins command2 passes the string to Insight II as is, without converting the data format.

ins getdir

The **ins getdir** command gets the current directory from Insight II.

The **ins** command has operators to facilitate communication between FELIX and Insight II. This allows the two programs to interact whenever they are being run simultaneously.

All the **ins** commands set the symbol **instat** to show the resulting status of that **ins** call. A status of zero (0) denotes success, while a non-zero status indicates an error.

The keywords are:

init initialize the FELIX server

connect connect FELIX to Insight II

check check if RPC connection is currently up

command send a **no reply wanted** command to Insight II

getdir get the current directory from Insight II

int — Integral

int

int generates the integral of the data in the workspace and pushes it onto the display stack. For optimal results using the **int** command, a level baseline is necessary.

Symbol changed

stack Stack Depth

See also

der Derivative

inv — Inverse of workspace

inv

inv replaces the data in the workspace vector by its inverse. The **inv** command takes the reciprocal of each point in the workspace (replaces work by 1 / work)

and stores the new value back in the workspace. If there are any zero points in the workspace, they are skipped to avoid a divide by zero error. **inv** can be used to create novel and useful window functions for apodization.

ip — Intensity plot

ip

ip draws an intensity plot of the current plot region on the graphics device. The default rendering mode erases the screen before each display; this action may be defeated by setting the reserved symbol **erase** to **1**.Video buffering may be enabled by setting the reserved symbol **animat** to **1**. The behavior of the command **ip** is affected by a number of other reserved symbols.

Symbol dependence

animat Specify Video-Buffering

clmode Selects Linear or Geometric Contour Spacing

conmod Contour Level Modifier

cycle Sets Color Cycle

drwbox Draw Box Around Plot

level First Contour Level

nlevel Number of Contour Levels

posneg Enables Plotting of Negative Contour Levels

projct Select Dimensionality of Display

pennum First Color

rowinc Point Skipping Factor For 3D Displays

Symbol changed

disply Current Display Type

See also

cp — Contour plot

np — Null plot

sp — Stack plot

pla — Redisplay 3D object

rmx — Reference matrix

jcp — Calculate J-coupling constant

jcp *peaks item who j1val j2val j1sig j2sig*

peaks	cross peak entity defining footprints in COSY matrix
item	item of selected peak (**-1** enables crosshair to select) (**-2** enables crosshair to select E-COSY peak)
who	symbol to receive item number selected if *item* = **-1**
jNval	symbol to receive J coupling value (Hz) along dimension *N*
jNsig	symbol to receive sigma of J coupling value along dimension *N*

jcp calculates J coupling constants by fitting multiplet footprints. This command destroys the contents of the work space and yields coupling constants for both dimensions along with standard error. Be aware that apodization functions can alter the observed line shapes of antiphase multiplets and affect the values calculated by **jcp**. This should be considered a somewhat crude measurement.

If *item* is set to **-2** then the program allows you to drag two subpeaks of an E-COSY multiplet, and the optimized J coupling values are returned to *jNval* variables.

The optimization method used in determining the coupling constant can be either quasi-Newton, simplex, or simulated annealing depending on the **optmth** symbol. The default is quasi-Newton minimization.

Symbol dependence

optmth optimization method for DQF-COSY or E-COSY J coupling extraction: Quasi-Newton (0), simplex (1) or simulated annealing (2).

kw — Kaiser window

kw *size parameter*

size	window size in points
parameter	window parameter; try 8

kw is a window function described in Hamming's book *Digital Filters*. This window is useful for apodizing truncated data.

ld — List data

ld *first last*

first	first point for listing data values (optional)
last	last point for listing data values (optional)

ld lists the values of the data in the work space. If no parameters are entered, data values for all of the points in the work space are listed.

See also

gv — Get value

ldb — Load buffer into work space

ldb *buffer*

buffer buffer number

ldb loads the contents of the specified buffer into the work space.

Symbol dependence

datsiz Number of Data Points

datype Data Type

nframe Number of Buffers

See also

adb — Add work to buffer

swb — Store work space to bundle

sp — Stack plot

lim — Matrix limits

lim *dimension first_point last_point*

dimension	matrix dimension for these limits
first_point	first point along dimension
last_point	last point along dimension

lim is used to select regions of multidimensional matrices for display on graphics devices. Matrix subsets are defined by specifying the first and last data points defining the boundaries of the subset for each matrix dimension using the **lim** command. (For example, by typing **lim 1 1 50**, **lim 2 1 50**, **lim 3 1 50**, you would define the limits of a 3D subset, in this case a cube.) Projections (or slices) of a matrix may also be defined by setting the first and last points of one of the limits to the same number for one or more dimensions, thus decreasing the dimensionality of the data. (For example, **lim** 1 25 25, **lim** 2 1 50, **lim** 3 1 50 would define a 2D slice of a 3D matrix taken through point 25, along dimension 1). To list the current matrix limits, the **lim 0** command may be entered. Other forms of the **lim** command include:

lim -1

Sets limits with the rubber band box cursor.

l**im -2**

Resets limits to their previous designation; like **old** for **1D**.

lim -3

Sets full **ND** limits, like **ful** for **1D**.

Symbol changed

lolimn Low Limit For Dimension N

hilimn High Limit For Dimension N

See also

bld — Build a matrix file

mat — Open matrix

cp — Contour plot

ip — Intensity plot

ord — Matrix dimension order

lin — Line annotation

lin *x0 y0 (z0 a0) x1 y1 (z1 a1)*

x0	x-coordinate starting point for line
y0	y-coordinate starting point for line
z0	optional z-coordinate for starting position for line if matrix is more than 2D
a0	optional a-coordinate for starting position for line if matrix is more than 3D
x1	x-coordinate for ending position for line
y1	y-coordinate for ending position for line
z1	optional z-coordinate for ending position for line if matrix is more than 2D
a1	optional a-coordinate for ending position for line if matrix is more than 3D

lin draws a line (solid or dashed depending on the symbol **annlst**) from (*x0*, *y0*) to (*x1*, *y1*). The optional coordinates are useful if the matrix is a 3D or 4D and the current plot is a strip plot. The line should be drawn starting in one strip and ending in another strip, where the strips are taken from different slices of the ND matrix. The color of the line is determined by the symbol **anncol**. The units of all coordinates are interpreted based on the symbol **annunt** and may be specified in a variety of units.

Symbol dependence

anncol Annotation Color

annlst Annotation Line Style

annunt Annotation Units

See also

arr — Arrow annotation

rec — Rectangle annotation

ann — Annotate plot

lis — List symbol table

lis *optional_specifier*

lis displays current symbols and their values. The optional specifier may contain a ***** wildcard to list only specified symbols.

See also

def — Define a symbol

lm — List macro

lm *file_name*

file_name macro file name (optional; default is current macro)

lm lists the current or specified macro.

loa — Load vector from matrix

loa *dim1 dim2 dim3 ...*

dim1 vector coordinate to be loaded in D1 dimension
dim2 vector coordinate to be loaded in D2 dimension
dim3 vector coordinate to be loaded in D3 dimensio

loa loads the specified vector from a matrix into the work space. For example, **loa 0 1** would load the vector along D1 that passes through point 1 of D2; and **loa 1 0** would load the vector along D2 that passes through point 1 of D1. **loa** must be given exactly one parameter that is zero. You can think of the zero as specifying the dimension along which the vector is loaded, and the other parameters locating the vector position in all other dimensions.

Symbol changed

dnvect Vector Coordinates Along Dimension N

datsiz Number of Data Points

datype Data Type

sfreq Observe Frequency

swidth Spectral Width

refsh Reference Shift

refpt Reference Point

first First Point

last Last Point

disply Current Display

See also

sto — Store vector to matrix

lwb — Load work space from bundle

log — Natural logarithm of work space

log

log replaces each data point in the workspace with its natural (base e) logarithm. The **log** command may be used to compute some novel windows for apodization. If a data point in the workspace is less than or equal to zero, it is set to zero to avoid a mathematical error.

Symbol dependence

datsiz Number of Data Points

See also

aln — Antilogarithm (exponential) of work

lmd — Load theoretical vector

lmd *dim1 dim2 dim3 ...*

dim1	vector coordinate to be loaded in D1 dimension
dim2	vector coordinate to be loaded in D2 dimension
dim3	vector coordinate to be loaded in D3 dimension

lmd loads the specified vector from a theoretical matrix into the work space similarly to the **loa** command. Therefore you must first issue the data modeling command (**md**) to display the theoretical matrix. The loaded vector contains data from the peak and volume entities. For example, **lmd 0 1** would load the vector along D1 that passes through point 1 of D2; and **lmd 1 0** would load the vector along D2 that passes through point 1 of D1. **lmd** must be given exactly one parameter that is zero. You can think of the zero as specifying the dimension along which the vector is loaded, and the other parameters locating the vector position in all other dimensions.

Symbol changed

dnvect Vector Coordinates Along Dimension N

datsiz Number of Data Points

datype Data Type

sfreq Observe Frequency

swidth Spectral Width

refsh Reference Shift

refpt Reference Point

first First Point

last Last Point

disply Current Display

See also

md — Model data

lpf — Linear predict first points

lpf *points coefficients peaks first*

points	number of data points used to calculate the linear prediction coefficients
coefficients	number of linear prediction coefficients used for linear predicting points
peaks	number of exponentially damped signals in the FID (not currently used)
first	first data point to linear predict

lpf uses linear prediction to estimate the values of incorrectly acquired first points using subsequent data points. **lpf** determines *coefficients*-LP coefficients based on data points numbered (1 + *first*) through *points*. New data values are generated backwards from point *first* to point one.

Suggestions for choosing the parameters include setting *points* to the number of data points in the FID and setting *coefficients* to one-quarter to one-third the value of *points*. However, if there are more than several hundred data points in the FID, **lpf** may require more than tens of seconds to perform its work. If this occurs, choose smaller values for these two parameters by selecting a value for *coefficients* that is greater than the number of signals in the FID and setting *points* to three or four times the value of *coefficients*.

The *peaks* parameter is left in for compatibility with older macros, but its value is not used in the calculation.

Symbol dependence:

datsiz Data Size

datype Data Type

See also

lpl — Linear predict last points

lpl — Linear predict last points

lpl *points coefficients peaks first last reflect*

points	number of data points used to calculate the linear prediction coefficients
coefficients	number of linear prediction coefficients used for linear predicting points
peaks	number of exponentially damped signals in the FID (not currently used)
first	first data point to linear predict
last	last data point to linear predict
reflect	root reflection flag

lpl uses linear prediction to extrapolate additional points from existing time-domain data. This command can be used as an alternative to zero-filling or apodization when dealing with truncated data. **lpl** determines *coefficients*-LP coefficients-based on data points **1** through *points*. New data values are generated for points *first* through *last*. If *last* is greater than the number of data points, the **datsiz** symbol is set to *last*. The root reflection flag enables reflection of roots that fall outside the unit circle, which ensures that the linear prediction coefficients represent decaying signals. Set *reflect* to **1** to enable root reflection, and to **0** to disable it.

Suggestions for choosing the parameters include setting *points* to the number of data points in the FID and setting *coefficients* to one-quarter or on-third the value of *points*; however, if there are more than several hundred data points in the FID, **lpl** may required more than tens of seconds to perform its work. If this occurs, choose smaller values for these two parameters by selecting a value for *coefficients* that is greater than the number of signals in the FID and setting *points* to three or four times the value of *coefficients*.

The *peaks* parameter is left in for compatibility with older macros, but its value is not used in the calculation.

Symbol dependence

datsiz Data Size

datype Data Type

See also

lpf — Linear predict first points

lps — Solvent suppression using linear prediction

lps *points peaks threshold*

points	number of points used to calculate linear prediction coefficients
peaks	number of signals to subtract from data
threshold	minimum ratio of adjacent singular values

lps subtracts the largest signal(s) from the time domain data, which accomplishes solvent signal suppression. The algorithm uses singular value decomposition to determine the singular values of the data matrix. These singular values and the signal amplitudes are correlated such that the highest amplitude signals have the largest singular values. The command uses this fact and additional information from the singular value decomposition (SVD) to generate a set of data points that are nearly identical to the highest amplitude signal(s) in the data set, and which is subsequently subtracted from the data. *points* sets the number of points used to create the data matrix for the singular value decomposition. The number of peaks you wish to subtract from the signal is set with *peaks*. When transforming data "blindly", as in macros, it may be useful to prevent signal subtraction when the *peaks* signal is not significantly larger than the next largest signal. The *threshold* parameter accomplishes this by referring to the ratio between the *peaks* singular value and the (*peaks* + 1) singular value. If the ratio falls below *threshold*, the subtraction will not occur.

See also

cnv — Time-domain convolution

lpx — General linear prediction

lpx *use_first use_last predict_first predict_last coefficients method reflect mode*

use_first	first point used to calculate the linear prediction coefficients
use_last	last point used to calculate the linear prediction coefficients
predict_first	first data point to linear predict
predict_last	last data point to linear predict
coefficients	number of linear prediction coefficients used for linear predicting points
method	method to use
reflect	root reflection flag
mode	mirror-image mode

lpx uses linear prediction to extrapolate additional points from existing time-domain data. This command can be used as an alternative to zero-filling or apodization when dealing with truncated data. **lpx** determines *coefficients* — LP coefficients — based on data points *use_first* through *use_last*. New data values are generated for points *predict_first* through *predict_last*. If *predict_last* is greater than the number of data points, the **datsiz** symbol is set to *predict_last*. The root reflection flag enables reflection of roots that fall outside the unit circle, which ensures that the linear prediction coefficients represent decaying signals. Set *reflect* to **1** to enable root reflection, and to **0** to disable it.

The method can be either forward prediction (*method*=**0**) or backward (*method*=**1**). You can use forward-backward (Zhu and Bax 1992) prediction, which has been proven to be superior to the forward method, but is slower (*method*=**2**). If the FID does not decay, then you can also carry out mirror-image linear prediction (Zhu and Bax 1990) (*method*=**3**). In this latter case you have to

set the *mode* parameter to reflect whether your data was collected with half dwell time shifted (**0**) or not (**1**).

Suggestions for choosing the parameters include setting *use_first* and *use_last* to include as many good points as possible from the FID, and setting *coefficients* to one-third or one-fourth the value of points; however, if there are more than several hundred data points in the FID, **lpx** may require more than tens of seconds to perform its work. If this occurs, choose smaller values for these two parameters by selecting a value for *coefficients* that is greater than the number of signals in the FID.

Symbol dependence

datsiz Data Size

datype Data Type

See also

lpl — Linear predict last points

lpf — Linear predict first points

lrl — Find local extremum

The **lrl** commands finds local minimum, maximum or extremum in the current spectrum.

lrl max *lopt1 hipt1 ... loptn hiptn symbol_posN symbol_int*

loptN	low point along dimension *N*
hiptN	high point along dimension *N*
symbol_posN	symbols to recieve the position of the maximum
symbol_int	symbol to recieve the value at the maximum

lrl max finds the maximum intensity position within the defined box limits and stores the position into the *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

If (*lopt1*) is set to **-1** it finds the maximum intensity position defined by the consecutive rubber box cursor:

lrl max -1 *symbol_posN symbol_int*

If (*lopt1*) is set to **0** it finds the maximum intensity position within the current plot limits:

lrl max 0 *symbol_posN symbol_int*

lrl min *lopt1 hipt1 ... loptn hiptn symbol_posN symbol_int*

loptN	low point along dimension *N*
hiptN	high point along dimension *N*
symbol_posN	symbols to receive the position of the local minimum
symbol_int	symbol to receive the value at the local minimum

lrl min finds the minimum intensity position within the defined box limits and stores the position into the *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

If the first parameter (*lopt1*) is set to **-1** then it finds the minimum intensity position defined by the consecutive rubber box cursor:

lrl min -1 *symbol_posN symbol_int*

If (*lopt1*) is set to **0** then it finds the minimum intensity position within the current plot limits:

lrl min 0 *symbol_posN symbol_int*

lrl ext *lopt1 hipt1 ... loptn hiptn symbol_posN symbol_int*

loptN	low point along dimension *N*
hiptN	high point along dimension *N*
symbol_posN	symbols to receive the position of the local extremum
symbol_int	symbol to receive the value at the local extremum

lrl ext finds the extremum intensity position within the defined box limits and stores the position into *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

If (*lopt1*) is set to **-1** it finds the local extremum intensity position defined by the consecutive rubber box cursor:

lrl ext -1 *symbol_posN symbol_int*

If (*lopt1*) is set to **0** it finds the local extremum intensity position within the current plot limits:

lrl ext 0 *symbol_posN symbol_int*

lrl pma *loppm1 hippm1 ... loppmn hippmn symbol_posN symbol_int*

loppmN	low ppm along dimension *N*
hippmN	high ppm along dimension *N*
symbol_posN	symbols to receive the position in ppm of the maximum
symbol_int	symbol to receive the value at the maximum

lrl pma finds the maximum intensity position within the defined box limits (in ppm) and stores the position in ppm into *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

lrl pmi *loppm1 hippm1 ... loppmn hippmn symbol_posN symbol_int*

loppmN	low ppm along dimension *N*
hippmN	high ppm along dimension *N*
symbol_posN	symbols to receive the position in ppm of the local minimum
symbol_int	symbol to receive the value at the local minimum

lrl pmi finds the minimum intensity position within the defined box limits (in ppm) and stores the position in ppm into *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

lrl pex *loppm1 hippm1 ... loppmn hippmn symbol_posN symbol_int*

loppmN	low ppm along dimension N
hippmN	high ppm along dimension N
symbol_posN	symbols to receive the position in ppm of the local extremum
symbol_int	symbol to receive the value at the local extremum

lrl pex finds the extremum intensity position within the defined box limits in ppm and stores the position in ppm into *symbol_pos1*, *symbol_posn*; the intensity at that place is stored in *symbol_int*.

lvo — Load volume time course

lvo *volumes item# (start)*

volumes	DBA volume entity
item#	item number of cross peak
start	optional parameter — start the time course with zero (0) or not (1)

lvo loads the volume time course for a single cross peak at all non-zero mixing times into the work space. This generates a set of x, y pairs in which x is time and y is volume. Use **dr** to display the volume time course, and **xyp fit** to fit the time course to a selected function. Peaks must have been picked and volumes measured before meaningful time courses may be loaded.

The optional *start* parameter can insert a zero volume at zero time (default action if no parameter specified or *start*=**0**) or not (*start*=**1**).

Symbol changed

datsiz Number of Data Points

datype Data Type

See also

mgv — Matrix get data value

gv — Get value

vol — Integrate cross peak volumes

xyp — X,Y data pair manipulation

pic — Peak pick and label

lwb — Load work space from bundle

lwb

lwb is used to load the next bundle vector into the workspace. The vector may then be processed and returned to the bundle using **swb**.

Symbol changed

datsiz Number of Data Points

datype Data Type

sfreq Observe Frequency

swidth Spectral Width

refsh Reference Shift

refpt Reference Point

See also

bun — Set bundle mode

loa — Load vector from matrix

swb — Store work space to bundle

mat — Open matrix

mat *file_name access storage*

file_name	file name of matrix
access	matrix access (**r** = read only, **w** = write enable)
storage	matrix storage location (disk or memory)

mat opens an existing matrix; previously built with the build matrix command (**bld**). An open matrix may be accessed one vector at a time using the load vector from matrix command (**loa**). A new matrix may be filled with data using the **sto** command.

The *storage* parameter allows you to control whether the matrix resides on disk (default) or is read entirely into memory. If your workstation has enough RAM, storing the matrix in memory speeds up most processing and plotting functions. If you attempt to store the matrix to memory and there is not enough space available, the matrix will simply stay on the disk and no error will be flagged.

Symbol dependence

matpfx Matrix Prefix

Symbol changed

dimen Dimensionality of Matrix

d1size D1 Size

d2size D2 Size

d3size D3 Size

d4size D4 Size

b1size D1 Brick Size

b2size D2 Brick Size

b3size D3 Brick Size

b4size D4 Brick Size

matfil Matrix File

See also

cmx — Close matrix file(s)

md — Model data

md *real model peaks volumes slot*

real	multiplicative factor for combining real data
model	multiplicative factor for combining modeled data
peaks	DBA peaks entity
volumes	DBA volumes entity
slot	DBA slot in volumes entity

md allows you to display any linear combination of real and modeled N-dimensional data. To use **md**, first pick peaks using **pic** and measure volumes using **vol**. These two steps define the model cross peak shifts, line widths, and intensities. Next, use the **md** command to enable display of model data. The effects of **md** stay in effect until disabled by setting *model* to zero. Modeled data can be displayed using **cp**, **ip**, and **sp**.

For example:

md 1 0 *(Display real data only — disables model data)*

md 0 1 peaks volumes 1 *(Display model data only)*

md 1 -1 peaks volumes 1 *(Display difference (real minus model)*

mf — Matched filter

mf *rho lbroad*

rho	ratio of the final Lorentzian line width to the initial line width (default = 2.0)
lbroad	line broadening to be used if fit fails (optional)

mf calculates and applies a matched exponential window to the FID. For *rho*=2.0, this window doubles the line width, and is a traditional "match filter". *rho* values of 1.3 to 1.5 are often used as well. *rho* allows you to tailor the trade off between resolution and sensitivity in the transformed spectra.

Line broadening is calculated using an analytical least-squares fit to the FID. If the FID has extremely low signal to noise, the fit may fail; a message to that effect will appear on the screen and the value you have specified in *lbroad* will be used instead. Note that a large, narrow solvent resonance may dominate the

fit. After **mf**, the reserved symbol **lbroad** is set to the value of the line broadening applied by **mf**.

Symbol dependence:

datsiz Number of Data Points

Symbol changed:

lbroad Line Broadening

mgv — Matrix get data value

mpv-Matrix put data value

mgv *dim1 dim2 dim3 ... symbol*

mpv *dim1 dim2 dim3 ... value*

dim1	point coordinate in the D1 dimension
dim2	point coordinate in the D2 dimension
dim3	point coordinate in the D3 dimension
symbol	symbol_name to receive the value at that *N*D point
value	real value to store at that N-D point

mgv and **mpv** are similar to the commands **gv** and **pv**, except these two commands operate on the current matrix instead of the 1D work space.

mgv loads the value out of the matrix at one N-D point, and stores that value to a symbol. **mpv** stores the given value into the matrix at the specified N-D point. The matrix must be write-enabled for **mpv**.

See also

gv — Get value

pv — Put value

mmp — Display memory map

mmp

mmp allows you to display and modify the memory map. FELIX maintains a pool of memory that can be allocated for use. The command **mmp** displays the size and usage for each allocated block of memory. **mmp** is commonly used as a diagnostic aid to help figure out what happened if something does not work properly.

See also

cfg — Configure memory

mnu — Menu manager

mnu *op parameters*

op	action
b(ar)	insert bar menu
g(auge)	put up, update, and remove a meter bar gauge
r(emove)	remove menu
p(arams)	insert a control panel
o(pen)	insert a modal control panel
a	insert a non-modal control panel
h(eader)	changes the header text of the main window
s	insert a control panel on top of a table

The **mnu** command provides menuing capabilities for FELIX. The **mnu** operations allow macros to generate popup menus on the display device and interact with you in menu mode. Menu and control panel definitions are simple ASCII files, which you can easily customize. More detailed information on the menu manager is given in Chapter 5, *Menus and Control Panels*

mnu bar *motfile X_origin Y_origin*

mnu bar draws a menu of items on the display. A bar menu goes across the display, while an insert menu goes down the display. The contents and size of the menu are read from the file *mot_file*. The location of the menu is specified in character cell units, with the upper left corner of the display being cell (x=1, y=1). No action other than drawing of the menu is performed. See Chapter 5, *Menus and Control Panels*, for a description of the contents of the menu files.

mnu gauge *1 max_value text_message*

mnu gauge *2 cur_value*

mnu *gauge 3*

The **mnu gauge** command allows you to create a descriptive meter bar gauge to show the progress of any complex operation. The meter gauge visually shows any value between zero and *max_value* as a colored bar that grows or shrinks as *cur_value* changes.

mnu gauge modes

mode	Description
1	defines a new meter gauge, gives the uppermost value, puts the meter on the center of the FELIX display, and labels it with the text message
2	updates the gauge to show the percentage: "*cur_value*/*max_value* * 100%"
3	removes the gauge and restores the image underneath.

In typical use, mode **1** is executed once, mode **2** is executed many times (with a different *cur_value* each time), and mode **3** is executed once. Note that only one gauge can be displayed at any given time. In addition, the image or picture behind the gauge should not be updated or redrawn while the gauge is displayed.

The selected item is returned in two symbols, **menu** and **item**. The **menu** symbol is set to the name of the file *menu_file* selected, or set to null if the cursor was not located on any menu. The **item** symbol is set to an integer specifying which item in the menu was selected, or set to 0 if the cursor was not on a menu.

mnu remove *menufile*

mnu remove all

Remove one menu from the display, or remove all menus if **all** is specified.

mnu p *menufile X_origin Y_origin*

Draws an interactive control panel (dialog box) on the display, letting you see and change symbol values. This operator waits for you to select an exit button using the mouse button, and then the control panel is removed. Based on which exit button is selected, either no action is performed (button 0) or all symbols appearing in the control panel are updated (any non-zero button). The reserved symbol **button** is set to the button number selected. See Chapter 5, *Menus and Control Panels* for a description of the content of control panel files.

mnu a *menufile X_origin Y_origin*

Draws a non-modal control panel on the display, letting you see and change symbol values. This control panel continues the execution of the macro without waiting for you to select an exit button button (unless you explicitly hold the macro flow using **wai**).). See Chapter 5, *Menus and Control Panels* for a description of the content of control panel files.

mnu h *option (text)*

The **mnu h** command allows you to change the header text of the main frame. If the **mnu h 1** *text* command is issued then the text in the *text* symbol is put on the frame header. The consecutive **mnu h 2** command clears up and sets back the header to the default setting of **1**.

Symbol changed

button Exit button number

ms — Magnitude spectrum

ms

ms replaces the real part of the work space with the sqrt[(real)2 + (imag)2)] or the absolute magnitude of the data, and replaces the imaginary part of the

work space with the arctan(real / imag) or the phase array of the data, in the range -180 to +180 degrees.

See also

ps — Power spectrum

mul — Multiply the work space by a number

mul *real_multiplier imaginary_multiplier*

real multiplier number to multiply all real points by
imaginary multiplier number to multiply all imaginary points by

mul multiplies the data in the work space by the specified number. If the data in the work space is complex (**datype** = **1**) then the multiplier may be complex. Note that if both the workspace and the parameters for this command are complex, then a complex multiplication will be carried out:

result_real = work_real * real_multiplier - work_imag * imag_multiplier

result_imag = work_real * imag_multiplier + work_imag * real_multiplier

The **mul** command allows you to change both the magnitude and the phase of the data in the work space.

Symbol dependence

datype Data Type

See also

add — Add number to work

mwb — Multiply work by buffer

mwb *buffer*

buffer buffer number

mwb multiplies the data in the work space by the data in the specified buffer. This command is commonly used after an apodization window is stored in a buffer; **mwb** multiplies the data in the work space by the stored window during a transform. Performing apodization by buffer multiplication saves time during lengthy multidimensional transforms.

See also

stb — Store work space to buffer

adb — Add work to buffer

ldb — Load buffer into work space

nd2 — Neighbor detection in 2D NOESY spectrum

This command is used to find potential neighbor patterns for any or all patterns. This requires a peak picked 2D NOESY spectrum and its associated patterns. The **nd2** command reports neighbor probabilities.

nd2 *tol tolerance*

tolerance resonance collapse tolerance

This command string defines the frequency collapse tolerance, to judge whether a candidate frequency is a new one.

nd2 noe *noecon noepks*

noecon minimum number of pattern frequencies with which a candidate must have NOE contacts

noepks minimum number of peaks (must be at least equal to the number of contacts)

nd2 pat *loppm hippm maxfrq*

loppm hippm range of pattern frequencies to use in ppm. Only NOEs with frequencies in this range are considered. For proteins, this is the HA, HB region, possibly extended with an amide proton.

maxfrq maximum number of frequencies to use for each pattern

nd2 *lev outlev*

outlev level of output: **0** = silent, **1** = low, **2** = medium, **3** = high, **5** = detailed

nd2 norm *false/true*

This command utilizes the normalize option, which normalizes the scores for each pattern to one.

nd2 ran *loppm hippm*

loppm hippm range of candidate frequencies in ppm (the amide proton region for proteins)

nd2 nei *number*

number number of candidate neighbors to store

nd2 rto *tolerance*

tolerance root frequency tolerance-used in the algorithm to find peaks within this limit

nd2 exe *nosypk */patnum*

nosypk executes the neighbor detection. The peak entity for a 2D NOE spectrum should be specified

**/patnum* all patterns, or a specific pattern, for which the detection should be carried out

nd3 — Neighbor detection in 3D NOESY spectrum

This command is used to find potential neighbor patterns for any or all patterns. This requires a peak picked 3D NOESY spectrum such as 3D NOE-NOE or 3D 15N-1H-HSQC-NOESY and its associated patterns. The **nd3** command reports and stores *i* - *i* + 1 neighbor probabilities.

nd3 met *method*

method neighbor finding method — 3D NOE-NOE spectrum (hom) or 3D HSQC-NOE spectrum (het)

This command string defines the method for neighbor detection — namely using homonuclear or heteronuclear spectrum.

nd3 tol *tolerance*

tolerance resonance collapse tolerance

This command string defines the frequency collapse tolerance, to judge whether a candidate frequency is a new one.

nd3 noe *noecon noepks*

noecon minimum number of pattern frequencies with which a candidate must have NOE contacts

noepks minimum number of peaks (must be at least equal to the number of contacts)

nd3 pat *loppm hippm maxfrq*

loppm hippm range of pattern frequencies to use in ppm. Only NOE's with frequencies in this range are considered. For proteins, this is the HA, HB region, possibly extended with an amide proton.

maxfrq maximum number of frequencies to use for each pattern

nd3 *lev outlev*

outlev level of output: **0** = silent, **1** = low, **2** = medium, **3** = high, **5** = detailed

nd3 norm *false/true*

This command utilizes the normalize option, which normalizes the scores for each pattern to one.

nd3 ran *loppm hippm*

loppm hippm range of candidate frequencies in ppm (the amide proton region for proteins)

nd3 sto *number*

number number of candidate neighbors to store

nd3 rto *tolerance*

tolerance root frequency tolerance-used in the algorithm to find peaks within this limit

nd3 seq *loppm hippm*

loppm hippm range of sequential frequencies in ppm — this is used for 3D HSQC-NOE spectrum and this is the range of the 15N frequencies

This command string defines the frequency range for the 15N resonances in the pattern s in the case of heteronuclear neighbor detection.

nd3 *rev t/f*

t/f look for reverse NOE contacts (true or false)

This command string defines whether the program should look for reverse NOE contacts appearing in, e.g., 3D TOCSY-NOE spectra.

nd3 shi *loppm hippm*

loppm loppm resonance range for patterns when looking for reverse NOE contacts

nd3 nsc *score*

score neighbor score

This command string defines the neighbor score option (values between **0** and **3**), used when the reverse NOE-contact option is on.

nd3 nuc *nucleus1 nucleus2*

nucleus1 first nucleus
nucleus2 second nucleus

This command string defines the type of nuclei in the pattern to use if the method is heteronuclear. You must define the *nucleus1* in accordance with the range defined in command **nd3 ran**, and *nucleus2* in accordance with **nd3 seq**. For example in a 3D 15N-1H-HSQC-NOESY spectrum the *nucleus1* should be set to *H* and the range should be defined through

```
nd3 ran 5.5 12.0
```

command for a protein and. The *nucleus2* should be set to *N* and the range should be defined through

```
nd3 seq 90.0 130.0
```

If a homonuclear spectrum is being used for neighbor detection, both *nucleus1* and *nucleus2* should be set to **H**.

nd3 cdm *dimension*

dimension candidate dimension

nd3 stm *method*

method neighbor storage method, overwrite old probabilities (0) or add to the old ones (1)

nd3 exe *nosypk */patnum*

nosypk executes the neighbor detection. The peak entity for a 3D NOE spectrum should be specified

**/patnum* all patterns, or a specific pattern, for which the detection should be carried out

nex — End of a loop

nex

nex defines the end for a **for** loop in macros. This command is only valid in macros.

See also

for — Loop for macros \

no — Generate random noise

no *amplitude*

amplitude amplitude of noise

no adds random noise of specified *amplitude* to the data in the work space. The amplitude distribution of the generated noise is normal (Gaussian) and its frequency distribution is white.

Symbol dependence

datsiz Number of datapoints

nop — No operation

nop

nop performs no operation. A typical use of **nop** in macros involves execution of commands that are stored in symbols. For example, you could save the cur-

rent window command in a symbol named **window**. If no window was used, the **window** symbol could be set to **nop**, which would not modify data when executed.

nor — Normalize data

nor *data point new value*

data point data point to normalize on
new value new value for this datapoint

nor normalizes the contents of the data in the work space so a specified point defined by the *data point* parameter has a given value defined by the *new value* parameter. This command is useful for comparing integrals and relative peak heights in separate spectra.

np — Null plot

np

np generates a null plot containing axes (like **ip** and **cp**) for the current region, but does not display any contours for data. **np** is very fast because it does not read matrix data for display. The main use of **np** is to establish graphics context without taking the time to draw a full plot. **np** is most often used prior to picking a large matrix to establish current plot context.

See also

cp — Contour plot

ip — Intensity plot

old — Recall old limits

old

old recalls the previous plot limits of 1D plots. If no previous plots were performed, the **old** command displays the full size of the data. For *N*-dimensional data, use **lim -2**.

Symbols changed

first First datapoint

last Last datapoint

opn — Open output file

opn *extension file_name access_mode*

extension	file extension (determines file prefix)
file_name	output file name
access_mode	selects new file or appends to end of existing file: **0** = open new file for output, **1** = open existing file for appended output, **2** = open existing file for input

opn opens a file for output or input. The default extension is .mac, but other extensions are valid as well. After **opn** has been used to create an output file (*access_mode*=**0**), the **put** command may be used in macros to put records in this file, which may contain any information you want. If the *access_mode* parameter is set to **1**, any lines output by **put** will be appended to the end of an existing file. If *access_mode* is set to **2**, the **rea** command may be used to read an existing file sequentially.

See also

put — Put record

cls — Close output file

rea — Read record from ASCII file

opt — FELIX option license inquiry or license checkin and checkout

opt *number symbol*

number	FELIX option number: **1**=2D, **2**=ND, **3**=Assign, **4**=Model
symbol	symbol to receive status: **0**=Not licensed, **1**=Licensed

opt gets the license status of any of the FELIX options. This command sets the specified symbol symbol to one if the option is licensed, and to zero if the option is not licensed to you.

To enable options that are not currently licensed, you will need a new feature in your license file. To obtain this, contact Accelrys (see the FELIX Getting Started document).

Licenses can be released or rerequested using the negative value for the *number* variable. For example:

opt -3 isok

causes the **Assign** license to be released if one was used by the current program (allowing other processes / users to use that feature), or alternatively check out an **Assign** license if there is one available in the license file.

ord — Matrix dimension order

ord *dim1 dim2 ... dimN*

dim1 matrix dimension to plot along x
dim2 matrix dimension to plot along y

ord is used to define the graphic representation of a matrix subspace. The default values for this command simply display the matrix dimensions in the order D1=1 (x axis), D2=2 (y axis) and D3=3 (z axis). If you wish to display the data transposed (i.e., the D2 dimension along the x axis instead of the y axis), simply exchange the order of the indices using the *dim1* and *dim2* parameters. For example, to display a 2D matrix transposed, enter **ord 2 1**. To list the current matrix order, enter **lim 0**.

See also

lim — Matrix limits

ovc — Overlay contour plot

ovc *(options)*

With this command you can overlay several plots on top of each other in the **Assign** module. Please note that, after overlaying plots, the active plot (i.e., the one from which values can be extracted) is the last spectrum plotted. You can overlay only contour plots.

These specific options must be executed before an overlay can be carried out

ovc clear

The **clear** option clears the overlay memory buffer; therefore anytime a new combination of spectra is to be overlaid this command has to be executed.

ovc set *spectrum_id*

The **set** option specifies which spectrum from the database should be overlaid. You have to call this command at least two times.

ovc connect *spectrum_id dimension spectrum_id dimension*

The **connect** option allows you to set, for two spectra, which dimensions should be mutually connected after the **ovc set** command is executed. This is particularly useful if you want to overlay 2D planes of 3D spectra.

ovc

Finally, calling the **ovc** command, with no options, draws the overlay plot.

pd2 — Prototype pattern detection in 2D

After cross peaks are picked, **pd2** performs prototype pattern detection on COSY, TOCSY, and NOESY type spectra for macromolecules, or performs prototype pattern detection on COSY, TOCSY, HMQC, and HMBC type spectra for small molecules.

pd2 seed *d1high d1low d2high d2low*

This command is useful only for macromolecules.

d1high d1low d2high d2low the seed area in ppm

pd2 exp *d1high d1low d2high d2low*

This command is useful only for macromolecules.

d1high d1low d2high d2low the expansion area in ppm

pd2 tol *tol tol2*

tol resonance collapse tolerance in ppm
tol2 C-13 resonance collapse tolerance in ppm. This is for small molecules only.

pd2 res *rppmin rppmax lopmax resmax*

This command is useful only for macromolecules.

rppmin minimum number of frequencies to add per loop
rppmax maximum number of frequencies to add per loop
lopmax maximum number of expansion loops
resmax maximum number of frequencies in prototype pattern

pd2 rem *logical*

logical remove intra prototype pattern peaks: true or false

pd2 lev *outlev*

This command is useful only for macromolecules.

outlev output level (**0** = quiet, **1** = low, **3** = high)

pd2 con cos *contact2 contact3 contact4 contact5*

pd2 con cot *contact2 contact3 contact4 contact5*

pd2 con ctn *contact2 contact3 contact4 contact5*

These command strings specify minimum numbers of contacts for COSY (**cos**), COSY+TOCSY (**cot**), and COSY+TOCSY+NOESY (**ctn**) type calculations, required for a frequency to be considered as a candidate, if the number of frequencies in a prototype pattern is 2, 3, 4 or more. This command is useful only for macromolecules.

pd2 nfi *fltnum*

This command is useful only for macromolecules.

fltnum number of ppm filters

pd2 fil *number loppm# hippm# min# max#*

This command is useful only for macromolecules.

number	filter number to assign limits to
loppm#	low ppm limit of filter #
hippm#	high ppm limit of filter #
min#	minimum number of frequencies within this filter in prototype pattern
max#	maximum number of frequencies within this filter in prototype pattern

pd2 ent cos *cosy peak entity*

pd2 met cos

These command strings specify the COSY peak entity, and define the method to be based on COSY spectrum.

pd2 ent toc *tocsy peak entity*

pd2 met toc

These command strings specify the TOCSY peak entity, and define the method to be based on TOCSY spectrum.

pd2 ent toc *tocsy peak entity*

pd2 ent noe *noesy peak entity*

pd2 met tno

These command strings specify the TOCSY and NOESY peak entities, and define the method to be based on TOCSY and NOESY spectra. This method is only for macromolecules.

pd2 ent toc *tocsy peak entity*

pd2 ent cos *cosy peak entity*

pd2 ent noe *noesy peak entity*

pd2 met ctn

The above command strings specify the TOCSY, COSY, and NOESY peak entities, and define the method to be based on TOCSY, COSY and NOESY spectra. This method is only for macromolecules.

pd2 ent hmq *hmqc peak entity*

pd2 ent cos *cosy peak entity*

pd2 met qco

The above command strings specify the HMQC and COSY peak entities, and define the method to be based on HMQC and COSY spectra. This method is only for small molecules.

pd2 ent hmq *hmqc peak entity*

pd2 ent toc *tocsy peak entity*

pd2 met qto

The above command strings specify the HMQC and TOCSY peak entities, and define the method to be based on HMQC and TOCSY spectra. This method is only for small molecules.

pd2 ent hmq *hmqc peak entity*

pd2 ent cos *cosy peak entity*

pd2 ent toc *tocsy peak entity*

pd2 met qct

The above command strings specify the HMQC, COSY and TOCSY peak entities, and define the method to be based on the HMQC, COSY and TOCSY spectra. This method is only for small molecules.

pd2 ent hmq *hmqc peak entity*

pd2 ent cos *cosy peak entity*

pd2 ent hmb *hmbc peak entity*

pd2 met qcb

The above command strings specify the HMQC, COSY and HMBC peak entities, and define the method to be based on HMQC, COSY and HMBC spectra. This method is only for small molecules.

pd2 ent hmq *hmqc peak entity*

pd2 ent toc *tocsy peak entity*

pd2 ent hmb *hmbc peak entity*

pd2 met qtb

The above command strings specify the HMQC, TOCSY and HMBC peak entities, and define the method to be based on HMQC, TOCSY and HMBC spectra. This method is only for small molecules.

pd2 exe

This command string executes the **pd2** command for macromolecules.

pd2 ana

This command string executes the **pd2** command for small molecules.

Symbol dependence

rprent Prototype pattern entity

pd3 — Prototype pattern detection in 3D

pd3 performs prototype pattern detection on 3D homonuclear spectra (for example, 3D TOCSY-TOCSY or 3D TOCSY-NOESY) or on 3D heteronuclear spectra (3D 15N-1H HSQC-TOCSY, 2D 15N-1H-HSQC and 3D 15N-1H HSQC-TOCSY, 3D HCCH-TOCSY), after cross peaks are picked:

pd3 seed *d1low d1high d2low d2high d3low d3high*

d1high d1low d2high d2low d3high d3low the seed area in ppm

pd3 exp *d1low d1high d2low d2high d3low d3high*

d1high d1low d2high d2low d3high d3low the expansion area in ppm

pd3 tol *tol*

tol resonance collapse tolerance in ppm

pd3 res *rppmin rppmax lopmax resmax*

rppmin minimum number of frequencies to add per loop
rppmax maximum number of frequencies to add per loop
lopmax maximum number of expansion loops
resmax maximum number of frequencies in prototype pattern

pd3 rem *logical*

logical remove intra prototype pattern peaks: true or false

pd3 lev *outlev*

outlev output level (**0** = quiet, **1** = low, **3** = high)

pd3 jco *contact2 contact3 contact4 contact5*

pd3 jno *contact2 contact3 contact4 contact5*

These command strings specify minimum numbers of contacts for J coupled (**jco**), and J coupled+NOESY (**jno**) type calculations, required for a frequency to be considered as a candidate, if the number of frequencies in a prototype pattern is 2, 3, 4 or more.

pd3 nfi *fltnum*

fltnum number of ppm filters

pd3 fil *number loppm# hippm# min# max#*

number filter number to assign limits to
loppm# low ppm limit of filter #
hippm# high ppm limit of filter #
min# minimum number of frequencies within this filter in prototype pattern
max# maximum number of frequencies within this filter in prototype pattern

pd3 use *used1 used2 used3*

usedn which coordinate of the seed peak to use — *usedn* = 0 means not to use this coordinate as a frequency in the prototype pattern

For heteronuclear detection you usually need to use the HN and N frequencies of the seed peaks.

pd3 *uex used1 used2 used3*

usedn which coordinate of the expansion peak to use — *usedn* = 0 means not to use this coordinate as a frequency in the prototype pattern

For heteronuclear detection you usually need to use the aliphatic H frequencies of the expansion peaks.

pd3 rot *dimension*

dimension root frequency dimension (usually the HN dimension) necessary for heteronuclear detection methods: 3D 15N-1H HSQC-TOCSY, 2and 3D HCCH-TOCSY

pd3 met *method*

method 3D prototype pattern detection method — 3D homonuclear (hom), 3D 15N-1H HSQC-TOCSY (het), 2D 15N-1H HSQC + 3D 15N-1H HSQC-TOCSY (hsq), 3D HCCH-TOCSY (hch)

This command string specifies the method to base on the prototype pattern detection.

pd3 exe *peaktable (seed_peaks)*

peaktable peak entity name to use for the prototype detection
seed_peak seed peak entity needed if the method is hsq

This command string executes the **pd3** command.

Symbol dependence

rprent Prototype pattern entity

pen — Define a new colored pen

pen *pennum R G B color_table_size*

pen lets you add additional pens to FELIX. You give the RGB values that define the color and the pen number you want to refer to this new color. Using this command, you can build your own color ramps.

The *color_table_size* is an optional value that tells FELIX how big the color map for your computer actually is.

On SGI computers running the GL version of FELIX, double buffer mode limits the number of bitplanes to one half of the number you actually have. For example, an 8-bitplane machine can only support 16 colors, while a 24-bitplane machine can support 4096 colors.

pen read *pennum ired igreen iblue*

This returns the rgb values for the pen number in the **ired**, **igreen**, and **iblue** symbols

ph — Phase correction

ph

ph performs a phase correction on the workspace based on the current values of the reserved symbols for zero order phase correction (**phase0**) and first order phase correction (**phase1**). The workspace must contain complex data in order to perform phase correction.

Symbol dependence

datsiz Number of datapoints

datype Data type

phase0 Zero-order phase angle

phase1 First-order phase angle

See also

aph — Autophase spectrum

rph — Real-time phase

pic — Peak pick and label

pic

pic picks peaks in a one- or multidimensional spectrum, generates a peak list, and annotates the current display. The **pic** command behaves differently for 1D and 2D spectra.

For 1D spectra

pic *entity pick_mode symbol*

pic picks 1D peaks and stores them in the database entity.

entity	database entity where 1D peaks are stored
pick_mode	**0**=build new entity of all peaks in current display
	1=append to existing entity all peaks in current display
	2=append to existing entity using a crosshair cursor
	3=append to existing entity multiple within rubber box
symbol	number of picked peaks

The behavior of **pic** is controlled by *pick_mode* as follows:

pic can generate peak labels on a 1D spectrum. For this application, it is recommended that you set the scale factor to 0.7, then draw (**dr**) the spectrum to get some empty space at the top of your current drawing. **pic** will label the peaks in either ppm or Hz depending on the current axis definition (set with the **axtype** symbol), unless **pkunit** is set to a number other than zero. In addition, **pic** will not pick any peaks smaller than the threshold value (**thresh**). If **thresh** is zero, **pic** sets it to 0.25 times the biggest peak. If **posneg** is set to **1**, **pic** picks both positive and negative peaks.

Symbol dependence

axtype 1D axis type

pkunit Peak pick units

thresh Peak pick threshold

level Contour level

posneg Positive / negative peak switch

Symbols changed

picent Current peak pick entity

For *N*D Spectra

pic *peak_style entity pick_mode symbol*

pic *peak_style entity pick_mode symmetry symbol*

pic picks *N*D peaks and stores them in the database entity *entity*. The number of peaks picked is returned in *symbol*. The peak shapes to search for is controlled by *peak_style* as follows:

peak_style	**0**=positive peaks (NOESY and TOCSY)
	1=absolute value (COSY anti-phase peaks
	2=negative peaks
	3=positive and negative peaks

When *peak_style* is set to one, the picker uses the values of the symbols **absmg1** and **absmg2** to define the x and y half widths (respectively) of a rectangle used

to better find the centers of multiplet cross peaks. Picking is actually done on the convolution of the box with the absolute value of the spectrum, in effect smearing out the fine structure of multiplets into a single fat positive peak. As a rule of thumb, set **absmg1** and **absmg2** to about the half width of an entire multiplet cross peak in dimension 1 and dimension 2, respectively.

The mode of data storage is determined by *pick_mode* as follows:

pick_mode **0**=build new entity of all peaks in the display
1=append to existing entity all picked peaks in the display
2=append to existing entity a single peak with the cursor*
3=append to existing entity multiple peaks selected using a rubber band cursor*[1]
4=append to existing entity a single peak at the location defined by the point (**x0pnt**, **y0pnt**)
5=append to existing entity multiple peaks in the box defined by the two points (**x0pnt**, **y0pnt**) and (**x1pnt**, **y1pnt**)

[1] When *pick_mode* is **2** or **3**, the *symmetry* parameter controls picking of the symmetric cross-diagonal peak as well. (**0**=no symmetric pick**, 1**=do symmetric pick)

Symbol dependence

level Contour level

hfwid1 Minimum halfwidth of the peak in points along D1

hfwid2 Minimum halfwidth of the peak in points along D2

hfwid3 Minimum halfwidth of the peak in points along D3

hfwid4 Minimum halfwidth of the peak in points along D4

mxwid1 Maximum halfwidth of the peak in points along D1

mxwid2 Maximum halfwidth of the peak in points along D2

mxwid3 Maximum halfwidth of the peak in points along D3

mxwid4 Maximum halfwidth of the peak in points along D4

maxmet Maximum definition method — center of gravity (0) or using singular-value decomposition (1)

fixwd1 Default halfwidth along D1 if halfwidth search failed

fixwd2 Default halfwidth along D2 if halfwidth search failed

fixwd3 Default halfwidth along D3 if halfwidth search failed

fixwd4 Default halfwidth along D4 if halfwidth search failed

xpklbl Label Peaks Switch

Symbols changed

pksent Current cross peak entity

piv — Set the pivot for phase correction

piv *flag*

piv displays (if *flag* = **1**) a small red triangle under the 1D axis that shows the pivot point for real-time phase correction. If *flag* = **0**, it hides the pivot.

Symbol dependence

pivot The data point used as a pivot for phsae correction.

See also

rph — Real-time phase

pla — Redisplay 3D object

pla

pla redisplays a 3D object on a 3D display and allows you to manipulate it using the standard 3D user interface. There must be a current 3D object for **pla** to work.

pol — Polynomial baseline correction

pol *order*

order polynomial order (1-9)

pol corrects the baseline of the data in the work space using the current list of baseline points. **pol** uses a polynomial of the specified order for the correction. Baseline points may be defined automatically with the automatic baseline points command (**abq**) or defined using **bas**.

pol uses the value of the interval width symbol (**iwidth**) to minimize the effects of noise on the correction by averaging the points in the interval about each baseline point (+/- **iwidth** points).

Symbol dependence

datsiz Number of datapoints

datype Data type

iwidth Interval width

See also

abl — Automatic baseline flattening

abq — Automatic selection of baseline points

bas — Baseline points manipulation

csp — Cubic spline baseline correction

flt — FLATT baseline flattening

pop — Pop the display stack

pop

pop causes the buffer stack head to be moved to the workspace and decreases the value of the stack depth parameter (**stack**).

Symbol dependence

datsiz Number of Data Points

datype Data Type

Symbols changed

stack Stack depth

See also

ldb — Load buffer into work space

psh — Push the work space onto the buffer stack

stb — Store work space to buffer

xsh — Exchange stack head with work space

ppm — Convert Between points and PPM

ppm *dim mode value symbol*

dim	dimension number, zero for 1D
mode	**0**=points to ppm **1**=ppm to points **2**=points to Hz **3**=Hz to points
value	point or PPM value
symbol	symbol name for converted value

ppm allows conversion between data point and ppm or Hz units for 1D data or for any dimension of *N*-dimensional data. **ppm** uses the information entered using **ref** (1D) or **rmx** (ND) and gives easy access to shift information.

prb — Residue type probability scoring

prb *Cappm Cbppm* (*pattern#*)

Cappm	C chemical shift
Cbppm	C chemical shift
pattern#	pattern number to store the residue probabilities with, if none is defined the probabilities will just printed out to the output window

This command uses the Grzesiek-Bax method (Grzesiek and Bax 1993) to calculate residue type probabilities from the C and C chemical shifts. This command either stores the result with the pattern if the pattern number (*pattern*) is given, or prints out the result to the output window.

prf — Formatted print

prf *format symbol1 ... symboln result*

format	FORTRAN-like format specifier
symbolN	symbols to print
result	resulting symbol holding formatted text of previous variables

prf puts a format description (FORTRAN style) for printing variables (symbols) into a string (symbol). For example:

```
prf 'i1,1x,a5,1x,f7.3,1x, "This is the end"' &myint
'&mytext' &myreal result
```

prints an integer, then text, then a real number, and then the quoted text to the variable (symbol) *result*.

The allowed format directives are:

i integer

f float

e scientific notation float

a character string

x space

t tab

> **Note:** Each variable in the list should have a specific formatting statement. For example. you cannot use **2(f7.3)** in the format statement. You must instead use **f7.3,f7.3**.

ps — Power spectrum

ps

ps replaces the real part of the data in the workspace with [(real)2 + (imag)2], or the power spectrum, and sets the imaginary part to the phase angle.

psa — Suggest assignment for a set of patterns

This command can be used to find potential matchings of patterns assigned to residue types onto the sequence of an unbranched biomacromolecule. The neighbor detection commands (**nd2** or **nd3** or the supported macros) and the residue type identification command (**prb** or the supported macros) should be run before using the **psa** command.

psa asn *asspro neipro*

asspro minimum assignment probability that a pattern should consider

neipro minimum neighbor probability score of a pattern

psa lev *outlev*

outlev level of output (**1** = low, **3** = medium, **5** = high, **9** = detailed, **100** = very detailed)

psa sto *logical*

logical whether to store assignment (**true** = store, **false** = print without storing)

This command stores and sort assignments or prints them as they are determined.

psa res *firres lasres minlen maxnas*

firres first residue for which to generate assignments

lasres last residue for which to generate assignments

minlen minimum length of assigned stretches to report

maxnas maximum number of assignments to generate

psa pri *maxpas*

maxpas maximum number of assignments to print

psa rto *rottol*

rottol root frequency tolerance

psa exe *stretch*

This command string executes the **psa** command and stores the resulting stretches in the *stretch* entity.

psh — Push the work space onto the buffer stack

psh

psh causes the contents of the data in the workspace to be pushed onto the buffer stack. The stack depth symbol (**stack**) is incremented and the contents of the data in the workspace remain unchanged. This command is useful for saving some data temporarily, or for displaying more than one spectrum at a time using the draw command (**dr**).

Symbol dependence

datsiz Number of datapoints

datype Data type

Symbols changed

stack Stack depth

See also:

ldb — Load buffer into work spacee

pop — Pop the display stack

stb — Store work space to buffer

xsh — Exchange stack head with work space

pso — Polynomial-based solvent suppression

pso *window order*

window number of points for window

order polynomial order for fit

pso removes a solvent signal from the time domain data. The solvent signal is approximated by calculating the mean value of every *window* points and fitting a polynomial of order *order* to the mean values. The resulting function is subtracted from the time domain data in work. This command performs well when the solvent frequency is close to zero.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

cnv — Time-domain convolution

lps — Solvent suppression using linear prediction

puf — Formatted put

puf *format symbol1 ... symboln*

format FORTRAN-like format specifier
symbolN symbols to print

puf takes a format description (FORTRAN style) for printing variables (symbols) and stores it in a file which was opened previously by the **opn** command. This command is similar to the **put** command but it allows formatting. For example:

```
puf 'i1,1x,a5,1x,f7.3,1x, "This is the end"' &myint '&mytext' &myreal
```

prints an integer, then a text, then a real number, and then the quoted text to the previously opened file.

The allowed format directives are:

i integer

f float

e scientific notation float

a character string

x space

t tab

> **Note:** Each variable in the list should have a specific formatting statement. For example, you can not use **2(f7.3)** in the format statement. You must instead use **f7.3,f7.3**.

pur — Purge symbol table

pur *symbol_list*

symbol_list list of user defined symbols to be deleted

The **pur** command purges the symbol list, causing user-defined symbols to be deleted. The symbol can be wildcarded:

```
pur abc*
```

which purges all variables having the first three letters "abc".

put — Put record

put *text*

text text to be written to file

put writes one line of text, defined by the parameter *text*, to the current output file (opened using **opn**). **put** can be used only within macros, and is useful for building new macros within a running macro. The *text* string will be subjected to symbol value replacement. To prevent symbol substitution within the text, precede the ampersand (&) with an exclamation point (!). The exclamation point (!) will prevent symbol replacement due to the ampersand (&), and the exclamation point (!) is then deleted from the output. Using the special string %t within the text creates a tab within the file.

See also

opn — Open output file

cls — Close output file

pv — Put value

pv *point real_value imaginary_value*

point data point to be changed
real_value real number value for the defined point
imaginary_value imaginary number value for the defined point

pv puts the specified value into the specified data point in the work space. If the point number is outside the range between 1 and the frame size (**frsize**), no action is taken. If the work space is complex (**datype**=**1**), then *imaginary_value* is put into the imaginary part of the work space point.

See also

gv — Get value

pxp — Automated peak assignment

This command can be used to automatically assign peaks after the spin systems are assigned to specific atoms.

pxp mul *yesno* (*entity*)

yesno do multiple assignment — *yesno*=1 or not *yesno*=0
entity multiple assignment peak entity — necessary if *yesno*=1

pxp lev *outlev*

outlev level of output (**0** = quiet, **4**= low, **6**= high

pxp any *enforce skipful skipmul*

enforce whether or not to enforce distance cutoff in NOE assignment (**0** = enforce cutoff, **1**= assigns clean peak even if distance criteria was not met)

skipful skip previously singly assigned peaks (1) or reassign them, too (0)

skipmul skip previously multiply assigned peaks (1) or reassign them, too (0)

psa tau *tauc cutoff specfrq t1leak unamb_dis*

tauc correlation time in ns

cutoff cutoff distance in Å

specfrq spectrometer frequency in MHz

t1leak t1 leakage in 1/s

unamb_dis unambiguous distance cutoff in Å

pxp exe *entity*

This command string executes the **pxp** peak autoassignment command using the *entity* peak table.

qsb — Skewed sinebell window

qsb *size shift skew*

size window size

shift phase shift

skew skew parameter

qsb multiplies the data in the work space by a skewed (or weighted) sinebell window. The *skew*, if less than one, skews the window to the left. If the *skew* is greater than one, the window function is skewed to the right.

Symbol dependence

datsiz Number of datapoints

See also

sb — Sinebell window

ss — Sinebell squared window

qss — Skewed sinebell squared window

qss — Skewed sinebell squared window

qss *size shift skew*

size window size
shift phase shift
skew skew parameter

qss multiplies the data in the work space by a skewed (or weighted) sinebell squared window. The *skew*, if less than one, skews the window to the left. If *skew* is greater than one, the window function is skewed to the right.

See also

sb — Sinebell window

ss — Sinebell squared window

qss — Skewed sinebell squared window

ra — Read ASCII data

ra *file_name*

file_name name of ASCII file to be read (required)

ra reads ASCII 1D data files produced by the write ASCII (**wa**) command. This command provides you with a means of transferring data between unlike hardware or different programs.

See also

wa — Write an ASCII data file

rb — Read Bruker file

rb *file* (*process_#*)

file name of the datafile to be read into FELIX
process_# optional parameter — specifies which pdata/*process_#*/proc(n)s files to use. If omitted it will search in for pdata/1... pdata/9 directories and whichever is found first will use that.

rb reads an FID from file into the 1D work space. rb can read Bruker files written on models AMX and newer. It is also important but not essential that the acqus... and pdata/../procs... files are present together with the ser or fid file, since the header parameters are read and used in FELIX.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference point

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

re — Read a file (old format)

re *file_name*

file_name name of datafile to be read into FELIX

re reads a data record from the specified file. The default file extension for the read command is .dat, therefore the extension does not need to be entered unless it differs from the default extension.

FELIX uses the same file type for 1D and 2D data. However, 2D datafiles contain more than one record. The data will therefore be read sequentially by issuing successive read (**re**) commands. To re-examine a data record after it has been read, the close command (**cl**) must be used to close the current data file and to re-position the record pointer to the beginning of the file.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference point

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

rea — Read record from ASCII file

rea

rea reads the next record from the current file opened using **opn 2**. The text of the record will be placed in the meta-string "$str".

See also

opn — Open output file

cls — Close output file

sub — Sub-string extraction

rec — Rectangle annotation

rec *x0 y0 x1 y1*

x0 starting x coordinate for rectangle
y0 starting y coordinate for rectangle
x1 ending x coordinate for rectangle
y1 ending y coordinate for rectangle

rec draws a rectangle having opposite corners (x0,y0) and (x1,y1). The color of the line is set by defining the symbol **anncol**. The coordinates may be specified in a variety of units as specified by **annunt**.

Symbol dependence

anncol Annotation color

annunt Annotation units

See also

ann Annotate plot

lin Line annotation

red — Reduce complex to real

red

red reduces a complex spectrum to a real one by discarding the imaginary part of the data in the workspace. The reduce command is used to convert a complex spectrum to a real spectrum.

Symbol dependence

datsiz Number of datapoints

datype Data type

Symbols changed

datype Data type

See also

cpl — Real to complex

ref — Set shift reference

ref *point shift*

point point number to be referenced to
shift chemical shift of reference point

ref defines the shift reference for 1D spectra. You can also reference a spectrum using the menus. For multidimensional spectra, use the **rmx** command.

Symbol dependence

axtype Axis type

Symbols changed

refpt Reference point

refsh Reference shift

ret — Macro subroutine return

ret

ret returns control from a called macro to the calling macro.

See also

cal — Macro call

exr — Execute a macro and return

rev — Reverse

rev

rev causes the data in the workspace to be reversed by swapping the data point values of 1 and 1024; 2 and 1023; 3 and 1022; and so on. When using this command, it makes a difference whether the data in the workspace is real or complex since, for the complex case, reversal will be pairwise.

Symbol dependence

datsiz Number of datapoints

rf — Read FELIX for Windows file

rf *file_name*

file_name name of the FELIX for Windows datafile to be read into FELIX

rf reads an FID from the FELIX for Windows file_name into the 1D work space.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference point

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

rft — Real Fourier transform

rft

rft assumes the contents of the work space to be real, and performs an in-place real Fourier transform. **rft** turns a real vector of length **datsiz** into a complex vector of length (**datsiz** / 2).

Symbol dependence

datsiz Number of datapoints

datype Data type

Symbols changed

datsiz Number of datapoints

datype Data type

See also

ft — Fast Fourier transform

ift — Inverse Fourier transform

bft — Bruker-Fourier transform

hft — Hilbert transform

rj — Read JEOL file

rj *file_name*

file_name name of the JEOL datafile to be read into FELIX

rj reads an FID from the JEOL *file_name* into the 1D work space. **rj** can read JEOL files collected with Alpha or Lambda systems.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference point

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

rm — Read macro

rm *file_name*

file_name macro filename

rm reads the macro file defined by the *file_name* parameter into the macro work space. All macros must have the default .mac extension. Once the macro is in the macro work space, it can be executed (**ex**) or listed (**lm**).

Symbol dependence

macpfx Macro prefix

Symbols changed

macfil Current macro file

rmx — Reference matrix

rmx *dim freq width axis refpt refval text*

dim	matrix dimension
freq	observe frequency for this dimension
width	spectral width for this dimension
axis	axis type (same as **axtype**)
refpt	reference point
refval	reference shift for reference point
text	axis label text

rmx sets shift reference information for one dimension of a matrix. This is similar to the **ref** command used for 1D data. A matrix must be opened before referencing. The reference information is stored permanently in the matrix file.

rmx may also be used to extract reference information from a matrix by using a negative dimension. The reference information corresponding to the above usage of **rmx** is loaded into the user symbols *sym1* through *sym6*.

rmx *-dim sym1 sym2 sym3 sym4 sym5 sym6*

rn — Read file (new format)

rn *file_name*

file_name name of the datafile to be read into FELIX

rn reads a *file* into the 1D work space. **rn** can read new format files written on hardware with a different byte order and transferred over a network.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference point

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

rph — Real-time phase

rph

This command is obsolete in FELIX 2002. It has been replaced by macros that use modeless control panels to realize real-time phasing. See macros such as *rtphase.mac* and *rtphase.mnu*.

rpl — Real-time polynomial baseline correction

rpl *order*

This command is obsolete in FELIX 2002. has been replaced by macros that use modeless control panels. See macro files *rtbasepoly.mac* and *rtbasepoly.mnu*.

See also

pol — Polynomial baseline correction

rv — Read Varian file

rv *file_name*

file_name name of the Varian datafile to be read into FELIX

rv reads an FID from the Varian *file_name* into the 1D work space. **rv** can read Varian fid files together with the procpar file collected under VXNMR 5.0 or newer.

Symbol dependence

frsize Frame size

Symbols changed

datsiz Number of datapoints

datype Data type

status Error status

axtype Axis type

refsh Reference shift

refpt Reference ppoint

phase0 Zero-order phase angle

phase1 First-order phase angle

sfreq Spectrometer frequency

swidth Spectrum width

See also

cl — Close a data file

re — Read a file (old format)

rn — Read file (new format)

rf — Read FELIX for Windows file

rv — Read Varian file

rj — Read JEOL file

ra — Read ASCII data

sar — Autoscreen command

sar ini *opt*

sar exe *specID item symbol_score symbol_threshold*

sar dsp *matrix opt threshold*

sar provides subcommands related to the Autoscreen module for SAR by NMR. Execution of this set of commands requires the Autoscreen license.

ini		**Set up scoring parameters.**
	opt	If *opt* = 1, discard previous results in score matrix where scoring results are stored. Otherwise, retain the previous score matrix.
exe		Score a selected test spectrum against the control spectrum.
	specID	Experiment ID of the test spectrum.
	item	Item number of the test spectrum in the experiment entity in the database.
	symbol_score	Symbol that returns the value of the score.
	symbol_threshold	Symbol that returns the value of threshold used to pick peaks in the test spectrum.
dsp		Cluster the experiments based on the displaced peaks, to identify binding subsites. **sar dsp** reorders the experiments and their peaks in the score matrix, so that experiments with the same group of displacement peaks are grouped in the same cluster, which can be viewed by displaying the score matrix.
	matrix	Filename of score matrix.
	opt	*opt* = 4; reserved for future development.
	threshold	Minimum contribution of the peaks to be clustered.

Symbol dependence

sar ini uses the values of these symbols:

pksent Entity name of the cross peaks in the database.

mindis1, **maxdis1** Lower and upper limits of peak displacements in the D1 dimension.

mindis2, **maxdis2** Lower and upper limits of peak displacements in the D2 dimension.

h1fac, **n2hfac** Scale factors for ^{1}H and ^{15}N chemical shifts.

thrmet Options for threshold method.

sarntp Number of sample points used to determine threshold automatically.

thrval User-defined threshold.

xpktyp Selection mode for peak picking.

maxmet Local maximum method for peak picking.

widop Option on whether or not to use peak width when matching peaks.

hgtop Option on whether or not to use peak height when matching peaks.

sarmsh Minimum similarity in peak shape required to match peaks.

scomet Method for searching a best match between the test and control peaks.

sarcpu The maximum CPU time (in seconds) allowed for a depth-first search

ucpfit Option on whether or not to fit unmatched control peaks to the test spectrum.

sarnmw Penalty of an unmatched control peak.

utpopt Selection method for unmatched test peaks

utpsgm Number of standard deviations tolerated when filtering unmatched test peaks.

sarutp Penalty of an unmatched test peak.

scomat Filename of the score matrix.

scorent Entity name of the scores in the database.

sb — Sinebell window

sb *size shift*

size window size in complex points
shift phase shift in degrees

sb performs a sinebell multiplication of the data in the work space using a function specified by the *size* and *shift* parameters. If no parameters are entered, the global parameter size (**datsiz**) is substituted for the *size* parameter and a phase shift of zero degrees is used. The sinebell window function yields good results for absolute magnitude spectra.

Symbol dependence

datsiz Number of datapoints

See also

ss — Sinebell squared window

qsb — Skewed sinebell window

qss — Skewed sinebell squared window

sca — Scale factor for dimension

sca *dimension scale_factor*

sca is used to control the appearance of multidimensional data displays. For example, if **sca 2 3** is entered, all plots will display dimension 2 scaled three times larger than normal. The default scale factor for all dimensions is one.

If the *scale_factor* is set to -1 then the plot is dynamically scaled, which means that the aspect ratio is discarded, and the plot fills out the available frame.

sca *-dimension scale_factor_variable*

returns the current scaling in *scale_factor_variable* if the negative *dimension* is used for the *dimension*.

seg — Integral segments

seg *op arguments*

op	*arguments*	action
add	*lowpt highpt*	adds one segment
show		displays segments
zero		zero or delete segment entity
normal	*lowpt highpt value*	normalize segment value

seg controls the creation, scaling, and display of segmented integrals on 1D spectra. The symbol **segint** switches on and off the automatic display of segmented integrals on 1D plots. The segment definitions are stored in a DBA entity named by the symbol
segent. Specifying *lowpt* as -1 enables a rubber band box cursor for adding and normalizing segments.

Symbol dependence

segent Segments entity

sep — Separate real and imaginary

sep

sep converts data consisting of alternating real and imaginary parts into data containing separate real and imaginary parts. All data within FELIX is processed in alternating mode. **sep** is useful for recovering imaginary parts of hyper-complex spectra for phasing after initial transformation. **sep** defines the data type to real, (**datype**=**0**) and defines the data size to two times the number of complex points.

Symbols changed

datsiz Number of datapoints

datype Data type

See also

alt — Alternating real/imaginary

set — Set work space to a value

set *real imaginary*

real value of real data
imaginary value of imaginary data (optional)

set simply assigns the specified value to all points in the work space. This command is useful for looking at the shape of window functions. For example, **set 1** sets the data in the work space to one. The desired window commands can then be issued, performing a window multiplication. The **dr** command can then be issued to draw the applied function in the graphics display to see what it looks like. If the work space is defined as being complex (**datype**=**1**), the **set** command can be used to define a complex value by specifying both the real and imaginary parameters.

Symbol dependence

datsiz Number of datapoints

datype Data type

shl — Shift left

shl *points*

points number of points to shift data left

shl shifts the data in the work space left by the number of points specified by the *points* parameter. Data values of zero are added to all points that lie to the right of the shift.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

shr — Shift right

ssh — Signed shift

csl — Circular shift left

csr — Circular shift right

csh — Circular signed shiftt

shr — Shift right

shr *points*

points number of points to shift data right

shr shifts the data in the work space right by the number of points specified by the *points* parameter. Data values of zero are added to all points that lie to the left of the shift.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

shr — Shift right

ssh — Signed shift

csl — Circular shift left

csr — Circular shift right

csh — Circular signed shiftt

smo — Binomial smooth

smo *times*

times number of times to smooth (optional, default=1)

smo performs a binomial smooth on the data in the work space by subjecting the data to a three-point smooth with binomial weighting. **smo 2** is equivalent to a five-point binomial smooth while **smo 3** is equivalent to a seven-point binomial smooth, and so on. Repetitive binomial smoothing is, in the limit of large numbers for the parameter *times*, equivalent to applying Gaussian convolution.

See also

flf — FaceLift baseline correction

sp — Stack plot

sp

sp produces a stack plot of the current region on the defined graphics device (display or plotter). The command **sp** uses the reserved symbols **deltax** for stack plot, **deltay** for stack plot, **rowinc** for row increment, and **cutoff** for peak height cutoff. These symbols control the appearance of the plot. **sp** produces plots that have a 3D appearance.

sp autoscales the peak heights. The autoscaling can be disabled by setting the reserved symbol **absint** to one. In this case, **sp** uses the scaling factor calculated for the most recent autoscaled **cp**, **ip**, or **dr** command.

Symbol dependence

deltax Delta X

deltay Delta Y

rowinc Row increment

bigpt biggest point

scale scale factor

Symbols changed

disply Current display type

sqz — Squeeze a matrix

sqz *file threshold*

sqz compresses matrices so that any point whose value is less than *threshold* is not included in the new squeezed matrix. Squeezed matrices are read-only, yet it is possible to define reference information using **rmx** after squeezing.

srv — Set range to value

srv *first last real imaginary*

first	first data point to set value for
last	last data point to set value for
real	real data value to be set for the range
imaginary	imaginary value to be set for the range (optional)

srv can be used to set any range of the data in the work space to a single value. Typically, the **srv** command is used to generate window functions that avoid truncation errors while minimally altering spectral intensities. For example, the following series of commands:

```
set 1

ss 100 90

shr 600

srv 1 600 1
```

defines a window equal to one for the first 600 points, which then drops smoothly to zero from point 600 to point 700, as a 90 degree shift sinebell function.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

set — Set work space to a value

ss — Sinebell squared window

ss *size shift*

size window size in complex points
shift phase shift in degrees

ss performs a sinebell squared multiplication of the data in the work space using the parameters specified by *size* and *shift*. If these parameters are not entered, the global parameter data size (**datsiz**) with a phase shift of zero degrees is used. The **ss** command is a very good window function for apodizing absolute magnitude spectra.

See also

sb — Sinebell window

qsb — Skewed sinebell window

qss — Skewed sinebell squared window

ssh — Signed shift

ssh *n1 n2 scale*

n1 first scaling point
n2 second scaling point
scale scale Factor for shift

ssh shifts the data in the work space to the left or right a certain number of points specified by [*n1* - *n2*]**scale*. Negative values shift the spectrum left while positive values shift the spectrum right. The **ssh** command can be used for tilt-

ing spectra where the *scale* parameter is usually set to the ratio of the w1 to w2 digital resolution.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

shr — Shift right

ssh — Signed shift

csl — Circular shift left

csr — Circular shift right

csh — Circular signed shift

ssp — Synthesize spectrum from peak list

ssp *peaks*

peaks 1D peaks entity

ssp generates a synthetic spectrum from peak parameters in the peaks entity *peaks*. The generated synthetic spectrum retains the data type (real or complex) of the original data. This is useful following peak fitting (**fit**) to generate the modeled spectrum.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

pic — Peak pick and label

fit — Fit 1D peaks

ste — Stella peak picker

ste *peak_entity pick_mode*

peak_entity peak entity where the picked peaks should be stored
pick_mode positive peaks (0), negative peaks (1) or both (2)

The **ste** command picks the peaks using example peaks in the xpk:example entity. It uses the current matrix limits and the current contour threshold to pick the peaks. The previous contents of the peak entity are preserved.

Symbol dependence

cosami minimum match factor

tresnb neighbor threshold

minbok minimum number of neighbor points within the search limits above the threshold

locsiz1 search size from maximum along d1 in points

locsiz2 search size from maximum along d2 in points

locsiz3 search size from maximum along d3 in points

locsiz4 search size from maximum along d4 in points

maxmet method to locate the maximum (0=rough maximum, 1=interpolation, 2=center of gravity)

trsint peak box threshold

facint hump tolerance factor

nextra extra points to increase by the peak box

outlev output level (0=quiet, 1=low, 2=medium, 3=high)

stb — Store work space to buffer

stb *buffer*

buffer buffer number

stb copies the contents of the work space into the specified buffer. The previous contents of the buffer are overwritten during the write process. The data in the work space and the stack depth parameter (**stack**) remain unchanged. The **stb** command is useful for temporarily saving spectral data.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

psh — Push the work space onto the buffer stack

ldb — Load buffer into work space

pop — Pop the display stack

xsh — Exchange stack head with work space

sto — Store vector to matrix

sto *dim1 dim2 dim3 ...*

dim1	vector coordinate to be stored in D1 dimension
dim2	vector coordinate to be stored in D2 dimension
dim3	vector coordinate to be stored in D3 dimension

sto stores the work space to the specified vector in the matrix. For example, **sto 0 1** stores to the vector along D1 that passes through point 1 of D2; and **sto 1 0** stores to the vector along D2 that passes through point 1 of D1. **sto** must be given exactly one parameter that is zero. You can think of the zero as specifying the dimension along which the vector is loaded, and the other parameters locating the vector position in all other dimensions.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

loa — Load vector from matrix

swb — Store work space to bundle

str — String manipulation operators

string	the given character string to operate on
substring	a second string to try to find in given string
matchstring	a wildcard string to compare with given string
begin	the first character wanted from given string
end	the last character wanted from given string
symbol	the symbol name to receive the result
word_symbol	the base symbol name to receive string words
cnt_symbol	the symbol name to receive the word count
delimiters	extra characters that break a string into words

str provides a variety of string manipulation operators. As a group, these let you perform almost any desired action involving character strings. To include blank characters into a string, enclose it in single quotes.

For use with the **rea** command, *string* may always be the meta-string $str to denote the line most recently read from an ASCII file.

str length *string symbol*

Return the length of the given string. The result is an integer greater than zero.

str sub *string begin end symbol*

Return a substring of the given string. Specify the first and last character locations wanted. The result is a string. This command is identical to SUB.

str index *string substring symbol*

Return the starting position in the given string where the first instance of the substring was found. Returns zero if the substring was not present in the given string. The result is an integer.

str match *string matchstring symbol*

Determines whether a wildcard string matches the given string. A valid wildcard string is of the form: **abc***, ***abc**, ***abc***. The result is an integer, where: 0 = not a match 1 = a match.

str parse *string word_symbol cnt_symbol*

Delimiters

Parse the given string into words, based on blanks and any other given delimiters. Each word in the string is stored in a symbol, and the total number of words is stored in a symbol.

For example, the command line:

```
str parse 'This is a string' word count
```

yields the following symbols and symbol values:

```
count = 4 word1 = This word2 = is word3 = a word4 = string
```

Another example:

```
str parse (123.01,278.31) value count ().,
```

parses based on blanks and the extra delimiters '().,' to yield:

```
count = 4 value1 = 123 value2 = 01 value3 = 278 value4 = 31
```

Note that any ASCII characters may be used as delimiters. The blank is always a delimiter.

str exact *string matchstring symbol*

Determines whether a string exactly matches the given string. The wildcards (*) are taken literally. The result is an integer, where: 0 = not a match 1 = a match.

sub — Sub-string extraction

sub *text begin end symbol*

text	text string from which to extract sub-string
begin	first character of sub-string to be extracted
end	last character of sub-string to be extracted
symbol	symbol to receive the text sub-string

sub extracts a sub-string from the symbol *text* based on the symbols *begin* and *end* and stores it into *symbol*. This command can be used after the **rea** command to extract portions of text lines from an ASCII file. The meta-string $str is used for *text* in this context.

See also

rea — Read record from ASCII file

swb — Store work space to bundle

swb

swb stores the contents of the data in the workspace into the vector last accessed by the load workspace from bundle command (**lwb**). Before using the **swb** command, you must use the bundle command (**bun**) to define the matrix as a bundle of vectors along a specified dimension. **swb** must follow **lwb**, as **lwb** increments the vector count in the matrix.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

bun — Set bundle mode

lwb — Load work space from bundle

sys — Execute system commands

sys *'DOS_command'*

sys puts you in touch with the command interpreter of the operating system. This command enables you to execute DOS commands with FELIX 2002. The DOS_command must be single-quoted.

Note: Windows allows spaces in a file or folder name. To avoid the potential problems caused by spaces in file or folder names, you are advised to double-quote all symbols that hold file or folder names, as in the examples below:

```
sys 'copy "&xxpath&xxfile" "&xxpath&yyfile"
sys 'del/f/q "&xxpath&xxfile"
```

For your convenience, the following new subcommands are added to achieve the common file manipulations:

```
sys del 'filename'
sys copy 'from_file' 'to_file'
sys rename 'from_file' 'to_file'
sys type 'filename'
```

Since these subcommands are not executed through the DOS command, the filenames must be single-quoted, *not* double-quoted.

Executing a DOS command from FELIX opens an DOS window which is closed upon the compeletion of the command. If you want to see the text displayed in the DOS window, you can do something like the following:

```
sys 'dir > temp.txt'
sys type temp.txt
```

The first command issues a DOS command dir and redirect the output to a file temp.txt. The second command displays the content of temp.txt in the output window of FELIX for your review.

tex — Text annotation

tex *x0 y0* (*z0 a0*) (*fix anchor*) *number text*

x0	x dimension for placing text
y0	y dimension for placing text
z0	optional z dimension for placing text
a0	optional a dimension for placing text
fix	scaling of text — fixed (0), according to Y size of plot (1), according to X size of plot (2), according to both sizes (3)
anchor	centering — left justify (0), center point (1), right justify (2)
number	number of text characters
text	text to be placed

tex draws text with its origin at the point (*x0*,*y0*) (optionally *z0* and *a0* if strip plot of a 3D or 4D matrix). The size of the text is set by defining the text size (**annsiz**) and the angle is set with the text angle symbol (**annang**). The color of the line is defined with the text color symbol (**anncol**). All coordinates are normalized device coordinates. **tex** can draw the full ASCII character set. How-

ever, you may not include the special characters "&" and ";" in the **tex** command within a macro. For a Greek character, use the **gre** command.

Symbol dependence

anncol annotation color

annunt annotation units

annang annotation angle

annsiz annotation text size

slant annotation text slant

thick annotation text thickness

See also

ann — Annotate plot

gre — Greek text annotation

til — Tile plot

til allows 2D subspaces of *N*-dimensional data to be plotted in segmented tiles with intervening data omitted. This permits detailed comparison of cross peaks from several parts of the spectrum at once.

There are three methods for generating a tile display. You can tile a cross peak and see all other peaks that align with the selected peak. You can tile a set of spins, and see the spin system based on the assigned shifts. Lastly, you can tile a pattern file containing pairs of atom names and see the spin system based on the assigned shifts.

To build a tile entity automatically from one cross peak, use **til make**. To display the data in these tiles, use **til on**. To eliminate one row or column of tiles, use **til reduce**. To return to normal non-tiled spectral displays, use **til off**. You can generate any tile display you want by creating your own tile entity, then turning tiles on and specifying the new tile entity.

til xpk *tile_entity xpk_entity item# correlation tile_size*

tile_entity	DBA entity for storing tile set
xpk_entity	cross peak entity defining selected cross peak
item#	selected cross peak number in *xpk_entity* (-1 denotes cursor select)
correlation	correlation cutoff (0.0 to 1.0)
tile_size	fixed size for each tile in points

To build a tile entity from a single cross peak use the peak *item#* as a center, and find all peaks that overlap this peak in both dimensions. When *correlation* is non-zero, eliminate those peaks whose line shape correlation with the specified peak is below *correlation*. Finally, make tile segments for each of these peaks. Each segment is *tile_size* points in size, or sized to the peak footprint if *tile_size* is zero. The tile segments are stored in the DBA entity *tile_entity*.

til atoms *tile_ent spins spin_ent shift_ent tile_size*

tile_ent	DBA entity for storing tile set
spins	specifies the set of spins to tile
spin_ent	spins entity of all spins in the sample
shift_ent	shift entity of shift for each spin
tile_size	fixed size for each tile in points

The above example builds a tile based on an atom name. Use an atom name specifier (may include *****) to find all matching names in the spins entity that have assigned shifts defined in the shift entity. Build tile segments from each chemical shift for each of these spins. Each segment is *tile_size* points in size, or will be sized to the line width of each spin if *tile_size* is zero. The tile segments are stored in the DBA entity *tile_ent*.

til pattern *tile_ent pat_file spin_ent shift_ent tile_size*

tile_ent	DBA entity for storing tile set
pat_file	ASCII pattern file
spin_ent	spins entity of all spins in the sample
shift_ent	shift entity of shift for each spin
tile_size	fixed size for each tile in points

The above example builds a tile entity from a pattern file. This is very similar to **til atoms**, except the spin names are extracted from an ASCII file that contains two spin names per line. A typical pattern file can contain expected NOE or J interactions between a set of spins, where each line specifies the names of two spins expected to interact.

til off

til on *tile_ent*

tile_ent	DBA entity for storing tile set

This operator turns tiling on and off. You can switch back and forth between a normal contour plot and a tile plot based on a specified tile entity.

til overlap *tile_ent*

tile_ent	DBA entity for storing tile set
overlap	minimum overlap to consider
size	tile size in points

overlap size

This operator combines overlapping tiles based on the *overlap* variable.

tile_ent	DBA entity for storing tile set

til reduce *tile_ent dimension point*

The above example reduces a tile display by removing one tile segment from one dimension. This has the effect of removing one entire row or column of tiles from the display. If *dimension* is -1, then a crosshair cursor is enabled to select the tile row or column to reduce by clicking on its axis label.

til strip *tile_ent*

tile_ent DBA entity for storing strip set

This operator allows you to make 2D strips from a 3D data set where each strip can be taken from a different plane.

til who *number*

number DBA entity for storing strip set

This command allows you to query the cursor position with respect to a strip plot — specifically, which strip plot the cursor pointed to.

Symbols changed

disply Current display type

tim — A basic clock and chronograph

tim *op symbol* (*symbol2*)

op one of the following operators
- **e** — echo current time as HH:MM:SS
- **s** — set symbol to seconds since midnight
- **b** — both, same as **e** and **s**
- **z** — zero chronograph, (i.e., start the stop watch)
- **r** — read chronograph, sets symbol to number of seconds since chronograph was zeroed
- **c** — initializes CPU time chronograph
- **n** — read CPU chronograph, sets symbols *symbol* to elapsed time and *symbol2* to CPU time

symbol symbol to receive number of seconds set by *op*'s **s**, **b**, **r**, and **n**

symbol2 symbol to receive number of CPU seconds set by *op* **n**

tim provides some basic timer abilities. Most notably, you can time long processing macros.

tm — Trapezoidal multiplication

tm *p1 p2 p3*

p1 first point for window function (optional, default=1)

p2 second set of constant points for window function

p3 last point for window function (optional, default = **datsiz**)

tm multiplies the data in the work space by a trapezoidal window that rises from zero at the first point up to one at *p1*, is equal to one from *p1* to *p2*, and falls to zero from *p2* to *p3*.

ty — Type text

ty *text*

text text to be written

ty types the one line of text that follows it. **ty** can be used only within macros, and is useful for writing tutorial macros or for generating messages so you can monitor the execution of a macro.

tyf — Type a file of text to the user

tyf *file_name*

file_name text file to type to user

tyf provides a way to put a large amount of text to the user with one command.

See also

ty — Type textt

tym — Type text to motif

tym *text*

text text to be written

tym displays a line of message on the status bar of the FELIX 2002 main window. It does *not* display the message in the text-port window. If you want to prevent the message from being overwritten, you should also use the **ty** command to display it in the text-port window..

unf — Unfold work

unf

unf produces a symmetric vector in the workspace by creating the mirror image of a spectrum and placing it on the right side of the current 1D vector. The **unf** command is almost the inverse of the fold work command (**fol**). The data size is doubled.

Symbol dependence

datsiz Number of datapoints

datype Data type

Symbols changed

datsiz Number of datapoints

ver — FELIX version number and release date

ver

Types the version number and release date to the text frame. The reserved symbol, **flxver**, is set to the version number as an integer, i.e., 230 for version 2.3..

vol — Integrate cross peak volumes

vol *peaks volumes slot ordinate*

peaks	DBA peak entity
volumes	DBA volume entity
slot	slot to store volumes
ordinate	reference value for this slot (mixing time)

vol extracts volumes of each cross peak in the database entity *peaks* from the current matrix. The footprint for each cross peak is stored in the DBA peak entity. The **vol** command stores cross peak volumes into the specified slot of the DBA volume entity. Volumes can be output in ASCII using standard DBA capabilities, or used by the **md** (model data) command.

Symbol dependence

hafwid half width factor

wa — Write an ASCII data file

wa *file_name*

file_name	name of file to be written (required)

wa will write an ASCII 1D data file to the disk with the specified name *file_name*. The **wa** command is the easiest way to transfer spectra to an alien program.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

ra — Read ASCII data

wr — Write a file (old format)

wn — Write a file (new format)

wai — Wait a while

wai *time*

time number of seconds to wait, which must be greater than zero

wai causes the program to wait for the specified number of seconds. The **wai** command is useful for causing delays in tutorials and may also be used to allow other users of your computer access to the processor while a multidimensional transform is being performed.

wai *-1 menufile*

wai *-1 menufile* causes FELIX 2002 to pause the execution of the current macro until a modeless dialog box (defined by a menu file "menufile") is closed..

wm — Write macro

wm *file_name*

file_name name of macro file (.mac default extension)

wm writes the macro from the macro work space to a .mac file.

Symbol dependence

macpfx Macro prefix

See also

rm — Read macro

lm — List macro

wn — Write a file (new format)

wn *file_name*

wn writes the contents of the work space to the specified file name. Files written using **wn** may be read after being transferred to hardware of differing byte order. **wn** allows you to write multiple 1D vectors into one file.

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

wr — Write a file (old format)

rn — Read file (new format)

wr — Write a file (old format)

wr *file_name*

wr writes the data in the current work space to a disk file with the specified name *file_name*. Subsequently, the data file can be retrieved using the read file command (**re**).

Symbol dependence

datsiz Number of datapoints

datype Data type

See also

wa — Write an ASCII data file

wn — Write a file (new format)

re — Read a file (old format)

xpa — Cross-peak assignments from shifts and spins

xpa *peaks spins shifts*

peaks cross peak entity
spins spins entity
shifts shifts entity

xpa uses information in the shifts and spins entities to generate parent names in an existing cross peak entity. For each cross peak in the *peaks* entity, **xpa** searches for the most appropriate parent with a shift defined in the *shifts* and *spins* entities. The parent name from the spins entity will be added to the cross peak entity as well as a spin pointer. If a parent name already exists in the cross peak entity, the name will not be changed.

See also

xps — Generate spins and shifts from cross peaks

xpk — Cross-peak operations

xpk provides operators for adding, deleting, and manipulating cross peak information stored in the database. Each operator acts on a single cross peak

item that may be specified explicitly by item number or by selecting a cross peak using the crosshair cursor (item = -1). Each operator and its function are described explicitly below.

Symbol dependence

hafwid cross peak half width factor

See also

xpl — Make a list of peaks

xpk who *peaks d1_pt d2_pt ... symbol*

xpk who *peaks* -1 *symbol*

peaks cross peak entity
dn_pt data point in dimension *N*
symbol symbol to receive chosen peak's item number

This operation finds the single cross peak in the specified cross peak entity *peaks* that is closest to the designated *N*-dimensional point location. Specifying a D1 location of -1 enables a crosshair cursor that may be used to select a location graphically. The selected point location must lie inside the cross peak footprint. The result stored in *symbol* is the item number of the closest cross peak, or zero if no peak footprint touches the specified location.

xpk add *peaks*

peaks cross peak entity

This operation allows you to manually add a single cross peak to the *peaks* entity. **xpk add** enables a crosshair cursor which is then used to locate the center of the desired cross peak, then a rubber box which is used to define the shape of the desired footprint. In contrast with **pic**, this operator does not reference the data values, and allows you to define footprints in regions of noise for the purpose of defining base intensities or noise levels. The new cross peak footprint is displayed in red.

xpk delete *peaks item*

peaks cross peak entity
item item number (or -1 to use crosshair cursor)

xpk delete deletes a single peak item from the cross peak entity *peaks*. The explicit item number may be specified or the peak may be selected graphically by specifying an item number of -1. The deleted peak is then overdrawn in black.

xpk edit *peaks item symbol*

peaks cross peak entity
item **-1** (uses crosshair cursor)
symbol symbol to receive selected peaks item number

This operator allows you to edit a cross peak footprint in the *peaks* entity. A single cross peak is selected by using the crosshair cursor. The selected cross peak is displayed in green. The mouse is then used to edit the cross peak footprint in one of two ways. The footprint center may be moved using a click-and-drag of the mouse, while the size of the footprint remains the same. If the mouse button is pressed near a cross peak corner, movement of the mouse adjusts the size of the footprint in both dimensions. Both modes terminate when the mouse button is released. The old footprint is then redrawn in black, and the new footprint is drawn in red to show the edited result. The item number of the selected peak (or zero if no peak is selected) is stored in *symbol*.

xpk name *peaks item dimen name*

peaks	cross peak entity
item	item number (or **-1** to use crosshair cursor)
dimen	dimension for assignment
name	cross peak parent name for specified dimension

xpk name inserts a cross peak parent name into the *peaks* entity. The cross peak may be selected explicitly by item number or graphically using a crosshair cursor if *item* is **-1**. The parent name should obey a nomenclature which is consistent with all structures and assignment libraries. The *name* **null** is used to indicate an unknown parent.

xpk volume *peaks item symbol1 symbol2*

peaks	cross peak entity
item	item number (or **-1** to use crosshair cursor)
symbol1	symbol to receive peak's volume
symbol2	symbol to receive peak item number

This operator integrates the intensity within the cross peak footprint. The cross peak may be selected explicitly by item number or graphically using a crosshair cursor if *item* is **-1**. The intensity of each data point within the cross peak footprint is summed and returned in *symbol1*. The item number of the selected peak is returned in *symbol2* (zero if no peak was selected).

xpk correlate *peaks item1 item2 mode symbol*

peaks	cross peak entity
item1	item number for cross peak 1 (or **-1** to use crosshair cursor)
item2	item number for cross peak 2 (or **-1** to use crosshair cursor)
mode	correlation mode: **1**=correlate projected line shapes of actual data, **2**=correlate footprint alignment only
symbol	symbol to receive correlation coefficient between two peaks

xpk correlate calculates a correlation coefficient between two cross peaks based on either line shape projections or position. The dimension along which the correlation is calculated is automatically chosen to be the dimension along which the peaks are closest. The result will be in the range 0.0 to 1.0, with zero indicating no correlation and one indicating perfect correlation.

xpl — Make a list of peaks

xpl provides operators for creating item lists of cross peaks. For general information about lists, see Chapter 6, *The Database and Tables*. Each of the following operators can be constructed from a set of DBA list commands and a small macro, but they are provided in the **xpl** form for simplicity and faster execution when dealing with lists of cross peaks.

Symbol dependence

hafwid cross peak half width factor

frsize frame size (biggest list size)

nframe number of buffers (number of lists)

See also

cfg — Configure memory

dba — Database facility

xpk — Cross-peak operations

xpl box *list# peaks lo1 hi1 ... loN hiN symbol*

xpl box *list# peaks* -1 *symbol*

xpl box *list# peaks* 0 *symbol*

list#	buffer number to store item list
peaks	cross peak entity
loN	low limit in points for each dimension
hiN	high limit in points for each dimension
symbol	symbol to receive number of items in list

xpl box makes a list of all peaks inside an *N*-dimensional box. The center point of a peak must be within the specified box for the cross peak to be added to the list. The first form of this operator uses explicit low and high limits for all matrix dimensions to define the *N*-dimensional box. If *lo1* is zero, the current plot limits are used to define the box. If *lo1* is -1, a rubber box cursor is enabled allowing you to define the box graphically. The number of cross peaks selected for the list is returned in *symbol*.

xpl touch_box *list# peaks lo1 hi1 ... loN hiN symbol*

xpl touch_box *list# peaks* -1 *symbol*

xpl touch_box *list# peaks* 0 *symbol*

list#	buffer number to store item list
peaks	cross peak entity
lo_pt	low limit in points for each dimension
hi_pt	high limit in points for each dimension

symbol symbol to receive number of items in list

xpl touch_box makes a list of all peaks inside an *N*-dimensional box. Any portion of a peak must be within the specified box for the cross peak to be added to the list. The first form of this operator uses explicit low and high limits for all matrix dimensions to define the *N*-dimensional box. If *lo1* is zero, the current plot limits are used to define the box. If *lo1* is -1, a rubber box cursor is enabled allowing you to define the box graphically. The number of cross peaks selected for the list is returned in *symbol*.

xpl inside_box *list# peaks lo1 hi1 ... loN hiN symbol*

xpl inside_box *list# peaks* -1 *symbol*

xpl inside_box *list# peaks* 0 *symbol*

list# buffer number to store item list
peaks cross peak entity
lo_pt low limit in points for each dimension
hi_pt high limit in points for each dimension
symbol symbol to receive number of items in list

xpl inside_box makes a list of all peaks inside an *N*-dimensional box. The entire footprint of a peak must be within the specified box for the cross peak to be added to the list. The first form of this operator uses explicit low and high limits for all matrix dimensions to define the *N*-dimensional box. If *lo1* is zero, the current plot limits are used to define the box. If *lo1* is -1, a rubber box cursor is enabled allowing you to define the box graphically. The number of cross peaks selected for the list is returned in *symbol*.

xpl point *list# peaks pt1 ... ptN symbol*

xpl point *list# peaks* -1 *symbol*

list# buffer number to store item list
peaks cross peak entity
ptN point along dimension *N*
symbol symbol to receive number of items in list

This operator builds a list of all peaks that touch an *N*-dimensional point. The peak footprint must include the specified point to be included in the list. The first form of the operator specifies explicit data points in all dimensions. Alternatively, if *pt1* is **-1**, a crosshair cursor is enabled allowing you to specify the point graphically. The number of selected peaks is returned in *symbol*.

xpl line *list# peaks dimen pt symbol*

xpl line *list# peaks* -1 *symbol*

xpl line *list# peaks* 0 *pt symbol*

list# buffer number to store item list
peaks cross peak entity
dimen dimension to specify subspace

pt	point along dimension *dimen*
symbol	symbol to receive number of items in list

xpl line builds a list of all cross peaks that touch a specified line. A line is defined in this context as an (*N* - 1) dimensional subspace of the matrix. This would be a line in a 2D matrix, a plane in a 3D matrix, and a rectangular prism in a 4D matrix. All cross peaks that touch point *pt* along dimension *dimen* are included in the specified list. The first form of the line operator specifies an explicit dimension and point. The second form specifies a single point, and selects cross peaks that touch that point along any dimension. Alternatively, you can specify *dimen* as -1 to enable a crosshair cursor and select the point graphically. In this case cross peaks are selected that touch either line of the crosshair cursor. The number of selected peaks is returned in *symbol*.

xpl range *list# peaks dimen lo_pt hi_pt symbol*

list#	buffer number to store item list
peaks	cross peak entity
dimen	dimension along which to specify range
lo_pt	low point along dimension *dimen*
hi_pt	high point along dimension *dimen*
symbol	symbol to receive number of items in list

This operator builds a list of all cross peaks whose centers lie within the specified range between *lo_pt* and *hi_pt* along the selected dimension *dimen*. The number of selected peaks is returned in *symbol*.

xpl name *list# peaks dimen name symbol*

xpl name *list# peaks* 0 *name symbol*

list#	buffer number to store item list
peaks	cross peak entity
dimen	dimension along which to specify range
name	parent name along dimension *dimen*
symbol	symbol to receive number of items in list

xpl name builds a list of all cross peaks whose parent name along dimension *dimen* matches *name*. If *dimen* is zero, the list will include any cross peak having a parent matching *name* along any dimension. A wild card (*) may be used in *name* to select partially defined parents.

xpl freq *list# peaks dim freq_list resolution symbol*

xpl freq *list# peaks* 0 *freq_list resolution symbol*

xpl freq *list# peaks* -1 *freq_list# resolution symbol*

list#	buffer number to store list
peaks	cross peak entity
dim	dimension to search
freq_list#	one of the frequency lists
resolution	+/- in ppm about each frequency
symbol	symbol to receive number of items in list

This subcommand makes a list of peaks that align with a frequency list. Each peak in the list will have its center in that dimension within *resolution* of a frequency in that frequency list.

Using a *dim* of zero will give all peaks that align with a frequency in *any* dimension; while a *dim* of minus one will give only the peaks that align with a frequency in *all* dimensions.

The number of peaks in the list is returned in *symbol*.

xpl pattern *list# peaks dim pattern_# resolution spectrum_id symbol*

list#	buffer number to store list
peaks	cross peak entity
dim	dimension to search
pattern_#	one pattern in Assign database
resolution	+/- in ppm about each frequency
spectrum_id	spectrum-specific shifts or generic shifts (0) to use from pattern
symbol	symbol to receive number of items in list

This subcommand makes a list of peaks that align with the frequencies of a pattern from the Assign database. Each peak in the list has its center in that dimension within *resolution* of a frequency in that pattern. You can use either generic shifts of the frequencies (*spectrum_id* = **0**) or a specific spectrum's shifts.

Using a *dim* of zero will give all peaks that align with a frequency in *any* dimension; while a *dim* of minus one will give only the peaks that align with a frequency in *all* dimensions.

The number of peaks in the list is returned in *symbol*.

xpl proto *list# peaks dim proto_# resolution symbol*

list#	buffer number to store list
peaks	cross peak entity
dim	dimension to search
proto_#	one pattern in Assign database
resolution	+/- in ppm about each frequency
symbol	symbol to receive number of items in list

This subcommand makes a list of peaks that align with the frequencies of a protopattern from the Assign database. Each peak in the list will have its center in that dimension within *resolution* of a frequency in that protopattern.

Using a *dim* of **0** gives all peaks that align with a frequency in *any* dimension; while a *dim* of **-1** gives only the peaks that align with a frequency in *all* dimensions.

The number of peaks in the list is returned in *symbol*.

xpl clipboard *list# peaks dim resolution symbol*

list#	buffer number to store list
peaks	cross peak entity
dim	dimension to search
resolution	+/- in ppm about each frequency

symbol symbol to receive number of items in list

This subcommand makes a list of peaks that align with the frequencies of the frequency clipboard in the **Assign** database. Each peak in the list will have its center in that dimension within *resolution* of a frequency in the clipboard.

Using a *dim* of **0** gives all peaks that align with a frequency in *any* dimension; while a *dim* of **-1** gives only the peaks that align with a frequency in *all* dimensions.

The number of peaks in the list is returned in *symbol*.

xpl pt *list# peaks dim assignment_pointer symbol*

list# buffer number to store list
peaks cross peak entity
dim dimension to search
asg_ptr assignment pointer
symbol symbol to receive number of items in list

This subcommand makes a list of peaks that have their assignment pointers identical to the target *assignment_pointer* in the required dimension.

Using a *dim* of **0** gives all peaks that have that pointer in *any* dimension; while a *dim* of **-1** gives only the peaks that have that pointer in *all* dimensions.

The number of peaks in the list is returned in *symbol*.

xpl pb *list# peaks loppm1 hippm1 ... loppmN hippmN symbol*

xpl box *list# peaks* 0 *symbol*

list# buffer number to store item list
peaks cross peak entity
loppmN low limit in ppm for each dimension
hippmN high limit in ppm for each dimension
symbol symbol to receive number of items in list

xpl pb makes a list of all peaks inside an *N*-dimensional box defined in ppm. The center point of a peak must be within the specified box for the cross peak to be added to the list. It is necessary to define the explicit low and high limits in ppm for all matrix dimensions to define the *N*-dimensional box.The number of cross peaks selected for the list is returned in *symbol*.

xpl tp *list# peaks loppm1 hippm1 ... loppmN hippmN symbol*

list# buffer number to store item list
peaks cross peak entity
loppmN low limit in ppm for each dimension
hippmN high limit in ppm for each dimension
symbol symbol to receive number of items in list

xpl tp makes a list of all peaks inside an *N*-dimensional box defined in ppm. Any portion of a peak must be within the specified box for the cross peak to be added to the list. It is necessary to define the explicit low and high limits in ppm for all matrix dimensions to define the *N*-dimensional box. The number of cross peaks selected for the list is returned in *symbol*.

xpl ip *list# peaks loppm1 hippm1 ... loppmN hippmN symbol*

list#	buffer number to store item list
peaks	cross peak entity
loppmN	low limit in ppm for each dimension
hippmN	high limit in ppm for each dimension
symbol	symbol to receive number of items in list

xpl ip makes a list of all peaks inside an *N*-dimensional box defined in ppm. The entire footprint of a peak must be within the specified box for the cross peak to be added to the list. It is necessary to define the explicit low and high limits in ppm for all matrix dimensions to define the *N*-dimensional box.The number of cross peaks selected for the list is returned in *symbol*.

xpl pp *list# peaks ppm1 ... ppmN symbol*

list#	buffer number to store item list
peaks	cross peak entity
ppmN	ppm value along dimension *N*
symbol	symbol to receive number of items in list

This operator builds a list of all peaks that touch an *N*-dimensional point defined in ppm. The peak footprint must include the specified point to be included in the list. It is necessary to define explicit data points in ppm in all dimensions. The number of selected peaks is returned in *symbol*.

xpl pl *list# peaks dimen ppm symbol*

list#	buffer number to store item list
peaks	cross peak entity
dimen	dimension to specify subspace
pppm	ppm value along dimension *dimen*
symbol	symbol to receive number of items in list

xpl pl builds a list of all cross peaks that touch a specified line. A line is defined in this context as an (*N* - 1) dimensional subspace of the matrix. This would be a line in a 2D matrix, a plane in a 3D matrix, and a rectangular prism in a 4D matrix. All cross peaks that touch point *ppm* along dimension *dimen* are included in the specified list. It is necessary to define explicit dimension and point in ppm. The number of selected peaks is returned in *symbol*.

xpl pr *list# peaks dimen loppm hippm symbol*

list#	buffer number to store item list
peaks	cross peak entity
dimen	dimension along which to specify range
loppm	low ppm along dimension *dimen*
hippm	high ppm along dimension *dimen*
symbol	symbol to receive number of items in list

This operator builds a list of all cross peaks whose centers lie within the specified range between *loppm* and *hippm* along the selected dimension *dimen*. The number of selected peaks is returned in *symbol*.

See also

fli — Frequency list manipulation

xps — Generate spins and shifts from cross peaks

xps *peaks spins shifts mode*

mode new/append switch: **0**=build new spins and shifts entities,
1=append to existing spins and shifts entities

The **xps** command provides a mechanism to build spin and shift entities from cross peak parent assignments contained in the cross peak entity. The cross peak entity allows two different ways of assigning a parent. One simply involves entering a text parent name into the cross peak item defining a parent along a given dimension. The other involves setting a parent pointer that refers to an item in a spin's or parent's entity. The pointer mechanism is favored for assignment purposes, since the parent name actually only occurs in one place, and points to all its cross peak "children" in all experiments.

Since spin and shift entities are required for back calculation, this command allows you to generate these entities from a cross peak entity with explicit parent names. For each unique parent name in the specified *peaks* entity, **xps** will add this name to the *spins* entity and the shift and line width information to the *shifts* entity. If the same parent name occurs in more than one cross peak, the shift and line width information from all cross peak children of that parent are averaged together. Any significant discrepancies are reported, allowing you to screen these for possible mis-assignments.

See also

bck — Back-calculate NOE intensities

xpa — Cross-peak assignments from shifts and spins

xsh — Exchange stack head with work space

xsh

xsh exchanges the data in the work space with the buffer stack head. The **xsh** command reverses the position of the lower two plots drawn with the draw command (**dr**).

Symbol dependence

datsiz number of datapoints

datype data type

stack stack depth

See also

ldb — Load buffer into work space

stb — Store work space to buffer

pop — Pop the display stack

psh — Push the work space onto the buffer stack

xss — Simulated annealing assignment functions

xss build *method experiment1* (*experiment2* (*experiment3*)) *spins_system_#* (*occurance*) *remove_intraspin_peaks iteration std_dev* (*htol*) (*ctol*) *output*

xss pattern *first_residue last_residue min_score discard store temp_fact iter_fact seq_fact output*

The simulated annealing assignment commands consist of functions that: 1) find spin systems (prototype patterns) in TOCSY and optionally COSY and optionally 1H-13C-HSQC cross peak sets using simulated annealing algorithm, 2) fit spin systems (patterns) with residue probabilities and neighbor probabilities to the sequence using simulated annealing.

xss build *method experiment1* (*experiment2* (*experiment3*)) *spin_system_#* (*occurance*) *remove iteration std_dev* (*htol*) (*ctol*) *output*

method	type of the protopattern search — TOCSY only (0), TOCSY and COSY (1), TOCSY and COSY and 1H-13C-HSQC (2)
experiment1	number of TOCSY experiment in the assignment project
experiment2	optional parameter — number of COSY experiment in the assignment project — necessary if *method*=1 or *method*=2
experiment3	optional parameter — number of 1H-13C-HSQC experiment in the assignment project — necessary if *method*=2
spin_system_#	spin system type number — from the residue type entity (reg:rseq), if *spin_system_#*=0, search for all spin systems
occurance	number of spin systems to look for if a particular spin system type is selected by *spin_system_#*, should not be present if *spin_system_#*=0
remove	remove intraspin system peaks from consideration — *remove* = 1 or use all peaks for prototype pattern detection — *remove* = 0
iteration	number of iterations
std_dev	maximum number of standard deviation to use from spin system chemical shift table
htol	interspectral tolerance between TOCSY and COSY spectra, required if *method* = 1 or 2, should not be present if *method* = 0
ctol	interspectral tolerance between TOCSY and 1H-13C-HSQC spectra, required if *method* = 2, should not be present if *method* = 0 or 1
output	output level — quiet (0), low (1), medium (2) or high (3)

The command finds specified spin systems or all spin systems (*spin_system_#*=**0**) in TOCSY, combined TOCSY and COSY, or combined TOCSY, COSY, and 1H-13C-HSQC spectra, using simulated annealing. This command can be used only within the Assign module, since it needs entities built by the Assign setup — project entity, prototype pattern entity, residue list and all the used experiments with their peaks picked.

You must supply the numbers of the experiments from the project entity. The resulting spin systems are stored in the prototype pattern entity.

This search should begin with the longest spin systems. As the algorithm tries to fit peaks into a defined motif, it will not take care of possible additional correlated frequencies, which means that an AMX portion of a long spin system could be assigned to a four-spin system. Initially, the program should be run on the whole residue set of the primary sequence (which will automatically take into account the above-mentioned priorities) and on the patterns examined with the usual interactive tools, then rerun on specific missing amino-acid types. To compensate for the limited number of iterations in the simulated annealing, the process should be run for several loops (typically 6), from among which the program will retain the best results. One loop of the program for the whole sequence of a 53 residue protein requires about ten minutes of computation time on an R4000 Silicon Graphics Indigo workstation. For aromatic residues, this method assigns only the AMX subsystems, therefore the aromatic resonances should be found with the systematic search method and added through the clipboard.

xss pattern *first_residue# last_residue# min_score discard store temp_fact iter_fact seq_fact output*

first_residue#	first residue in the sequence to consider for fitting
last_residue#	last residue in the sequence to consider for fitting
min_score	minimum residue type score to consider in the pattern
discard	discard previous assignments (0) or use it (1)
store	store the result of the sequential assignment in the pattern as a comment (1) or just print it to the output window (0)
temp_factor	initial temperature adjustment factor (between 0.1 and 10)
iter_factor	iteration adjustment factor (between 0.1 and 10)
seq_factor	neighbor probability/residue type probability weighting factor — relative weighting of neighbor probability vs residue type
output	output level — quiet (0), low (1), medium (2) or high (3)

This command can be run on any set of homonuclear or heteronuclear patterns (spin systems). It uses only the type and neighbors scores, obtained by any method, to find the sequence-specific assignment via simulated annealing optimization. Optionally, previous assignments are loaded and respected. The amino-acid type and / or residue number are considered assigned for a pattern if they are consistent over all frequencies of the pattern (unique or specified assignments).

After careful inspection of the patterns and scoring of types and neighbors, the process might be run on the full sequence. Then you might inspect the result,

modify it, perhaps try another run, and identify some satisfactory parts from the scores listed. You should then discard the ambiguous assignments and rerun the program with the correct residues used as anchor points. If several such iterative processes still fail to determine unambiguously the complete assignment, then some additional information should be input, like a more accurate scoring or some new patterns.

The results are stored in assignment pointers for all frequencies of the patterns (and set as the current specified frequencies). Note that there should not be any residue named "null" in the molecule, else its assignment will be discarded.

Optionally, some parameters of the simulated annealing might be adjusted (scaled by a factor of 0.1 to 10) according to the complexity of the problem:

temp_factor, *iter_factor*: if most parts of the sequence are well defined these parameters can be decreased to speed up the program.

seq_factor: weight is accorded to the neighbors information, relative to the spin system fit scores.

xyl — Atom list manipulation

The **xyl** command makes a database list of atom item numbers or frequencies based on specific selection criteria. The subcommands dealing with frequencies or patterns can be only used in conjunction with the Assign module (**xyl frequency, xyl pattern, xyl shift, xyl multiple, xyl fp)**

See also

xyz — Atom manipulation

xyl neighbors *atom radius others list# symbol*

atom	atom defining center of a neighborhood (this may be an atom name, an item number, or **-1** to select using the cursor)
radius	neighborhood radius in Angstroms
others	name match string to limit neighborhood selection
list#	buffer number to receive the selected atoms
symbol	symbol to receive number of atoms in list

xyl neighbors builds a list of atoms within a specified distance of a single atom. The list may be filtered based on a match with the string *other*, which may contain a wild card (*). The number of selected atoms is deposited in *symbol*.

xyl name *atom list# symbol*

atom	atom name match string to define selection criteria
list#	buffer number to receive the selected atoms
symbol	symbol to receive number of atoms in list

xyl name builds a list of atoms matching the string *atoms*, which may contain a wild card (*). The number of atoms selected is deposited in *symbol*.

xyl atom *atom radius others type list# symbol*

atom	atom defining center of a neighborhood (this may be an atom name, an item number, or **-1** to select using the cursor)
radius	neighborhood radius in Angstroms
others	name match string to limit neighborhood selection
type	type of atoms: 0 all, 1 assigned, 2 unassigned
list#	buffer number to receive the selected atoms
symbol	symbol to receive number of atoms in list

xyl atom builds a list of atoms within a specified distance of a single atom. The list may be filtered based on a match with the string *other*, which may contain a wild card (*). Also you can specify *type*, whether any atoms matching the above two criteria are to be collected to this list, or you need only assigned or only unassigned atoms. The number of selected atoms is deposited in *symbol*.

xyl frequency *atom list# symbol*

atom	atom name match string to define selection criteria
list#	buffer number to receive the selected frequencies
symbol	symbol to receive number of frequencies in list

xyl frequency builds a list of assigned frequencies from the Assign database whose assignments match exactly the string *atom*, which may contain a wild card (*) for pseudoatoms. The number of frequencies selected is deposited in *symbol*.

xyl pattern *atom list# symbol*

atom	atom name match string to define selection criteria
list#	buffer number to receive the selected patterns
symbol	symbol to receive number of frequencies in list

xyl pattern builds a list of assigned patterns from the Assign database, whose assignments match the string *atom*, which may contain a wild card (*). The number of patterns selected is deposited in *symbol*.

xyl shift *center delta spectrum_id list# symbol*

center	center of the search in ppm
delta	tolerance in ppm
spectrum_id	generic shifts (0) or spectrum specific shifts to be searched
list#	buffer number to receive the frequencies
symbol	symbol to receive number of frequencies in list

xyl shift builds a list of singly assigned frequencies from the Assign database, within a range *delta* from a specified *center*. You have to specify whether the generic shifts or spectrum-specific shifts of the frequencies are to be compared with the target shift. The number of selected frequencies is deposited in *symbol*.

xyl multiply *center delta spectrum_id list# symbol*

center	center of the search in ppm
delta	tolerance in ppm

spectrum_id	generic shifts (0) or spectrum specific shifts to be searched
list#	buffer number to receive the frequencies
symbol	symbol to receive number of frequencies in list

xyl multiply builds a list of multiply-assigned frequencies from the Assign database within a range *delta* from a specified *center*. You have to specify whether the generic shifts or spectrum-specific shifts of the frequencies are to be compared with the target shift. The number of selected frequencies is deposited in *symbol*.

xyl fp *frequency list# symbol*

frequency	frequency number to define selection criteria
list#	buffer number to receive the selected frequencies
symbol	symbol to receive number of frequencies in list

xyl fp builds a list of patterns from the Assign database which contains the target frequency. The number of patterns selected (usually one) is deposited in *symbol*.

xyp — X,Y data pair manipulation

In addition to real and complex data, FELIX supports yet another data type, namely (x,y) pairs of data. Each data point is actually stored as a triplet and consists of an abscissa (x), an ordinate (y), and a sigma or error term for y.

These lists of (x,y) pairs can be displayed using the **dr** command, saved and retrieved from buffers, edited to add and delete points, written to and read from files, and fitted to a variety of functions to yield model parameters. The reserved symbol **linpts** specifies the style in which the data is displayed:

linpts equal to	specifies
0	connecting line only
1	points with error bars
2	points, error bars, and connecting lines

Symbol dependence

linpts Display style

Symbols changed

datype data type

datsiz number of datapoints

disply current display type

stack stack depth

See also

dr — Draw work space and stack

lvo — Load volume time course

xyp zero

xyp zero zeroes the workspace, and changes the data type to (x,y) pairs.

xyp add *x y sigma_y*

x	x value
y	y value
sigma_y	sigma value

xyp add adds a single data point to the work space.

xyp delete *x y*

x	x value
y	y value

xyp delete deletes a single data point from the work space.

xyp cursor add

xyp cursor add adds a single point to the workspace. This operation requires a current plot to establish graphics context. A crosshair cursor is enabled, allowing you to select an (x,y) location and click the mouse button to add the point. The new *sigma_y* is set to one.

xyp cursor delete

xyp cursor delete allows you to use the cursor to select a point to be deleted from the workspace. A crosshair is enabled, which may be used to delete a point by placing it over the point and clicking the mouse button.

xyp get *pt# symbol_x symbol_y symbol_sigma*

pt#	point number
symbol_x	symbol for x value
symbol_y	symbol for y value
symbol_sigma	symbol for sigma value

xyp get loads the values from a specified point number *pt#* into the symbols *symbol_x*, *symbol_y* and *symbol_sigma*.

xyp label *x_label y_label*

x_label	text for x-axis units
y_label	text for y-axis units

xyp label specifies the label units for the x and y axis to be annotated on plots.

xyp sort *who order*

who	value to sort by: **1**=x value **2**=y value **3**=sigma value

order sort-order specification
1=ascending order
2=descending order

xyp sort sorts the (x,y) points based on x value, y value, or sigma value. To draw the data with lines connecting the points, the data must be sorted by x value in ascending order.

xyp read *file_name*

file_name file name of (x,y) pairs to read

xyp read reads in an ASCII file containing (x,y) pairs into the work space. The format of the file is:

Line 1: label text for both axes (2a20)

Line 2-N: X_value Y_value sigma_value (space delimited)

xyp write *file_name*

file_name filename of (x,y) pairs to write

xyp write writes the (x,y) pairs in the work space to an ASCII file with format as specified above.

xyp fit *function convergence parameters*

parameter	option	description
function		specifies fitting function
	0	help — lists all functions
	1	linear regression (straight line fit) f(x)=A0+A1*x
	2	polynomial (order is next parameter) f(x)=A0+A1*x+A2*x*x+A3*x*x*x...
	3	simple exponential with explicit zero intercept f(x)=A0*exp(A1*x)
	4	general simple exponential f(x)=A0+A1*exp(A2*x)
	5	bi-exponential f(x)=A0+A1*exp(A2*x)+A3*exp(A4*x)
	6	NOE time course f(x)=A0*x*exp(A1*x)
	7	general NOE f(x)=A0+A1*x*exp(A2*x)
	8	Hyperbolic rectangular f(x)=A0*(x/(x++A1)

xyp fit performs a least-squares fit of the (x,y) data in the work space to the specified model function. The resulting coefficients are saved into the symbol **coefN**, where N is the coefficient number. The sigma value for each coefficient is saved into the symbol **sigmN**, corresponding to each coefficient. The error of the fit is returned in the **chisq** symbol. Following the fit, buffer 1 contains the model data and stack is set to one. Subsequent draws using **dr** plot both the original and model data.

xyp build *function*

parameter	option	description
function		specifies fitting function

0	help — lists all functions
1	linear regression (straight line fit) f(x)=A0+A1*x
2	polynomial (order is next parameter) f(x)=A0+A1*x+A2*x*x+A3*x*x*x...
3	simple exponential with explicit zero intercept f(x)=A0*exp(A1*x)
4	general simple exponential f(x)=A0+A1*exp(A2*x)
5	bi-exponential f(x)=A0+A1*exp(A2*x)+A3*exp(A4*x)
6	NOE time course f(x)=A0*x*exp(A1*x)
7	general NOE f(x)=A0+A1*x*exp(A2*x)

xyp build performs a curve calculation using the symbols **coefN**, where N is the coefficient number. The function for the curve calculation is defined by the parameter *function* and the command will expect the coefficients to be in right symbol. The reconstruction will happen only if the workspace is in (x,y) pair form. The reconstructed curve is stored in buffer 1 and by subsequent drawing command the theoretical curve is getting drawn on the current x,y plot.

xyz — Atom manipulation

The **xyz** command provides a number of operators that act on atomic coordinates. These operators allow you to display, identify, label and interact with structures graphically while maintaining a connection to all NMR-related information.

Symbols changed

disply current display type

See also

bck — Back-calculate NOE intensities

xyl — Atom list manipulation

xyz draw *sphere play*

sphere	radius of van der Waals cloud to display (0=bonds only)
play	display mode: **0**=flat draw, **1**=3D display (if projct=4)

xyz draw displays the current molecule. The atoms are drawn using their current attributes, which include color, visibility and label. These attributes may be changed using other **xyz** operators.

xyz who *symbol*

xyz who enables a crosshair cursor and allows you to select an atom, then returns the item number of the atom in *symbol*. If no atom is selected, *symbol* is set to zero (0). This operator works both on "flat" and 3D displays.

xyz read *type file*

type filetype: **0**=PDB format, **1**=Insight II format, 2=X-PLOR PDB format and the names stored as Insight II names (used in Assign) **3**=MDL MOL format (used in **Analytical**)

file name of existing coordinate file

xyz read reads an ASCII file containing atomic coordinates. This operator builds DBA entities for atoms, attributes and residues. For PDB files, the file name can have an explicit extension (the default is .pdb). Insight II files should not include an extension, as the command will look for both *file*.mdf and *file*.car.

If *type*=**2** the X-PLOR type PDB file can be read in and the names of the atoms stored in Insight II type notation: 1:residuename_residuenumber:atomnamenumber

If *type*=**3** the MDL MOL file can be read in.

xyz label *atoms on/off redraw*

atoms atoms(s) to label (by item#, DBA, match string, or **-1** for cursor)

on/off label operation: **1**=turn label on, **0**=turn label off

redraw redraw status: **0**=no redraw, **1**=redraw now to show label change

xyz label labels one or more atoms in the current display with corresponding full atom names. Labels may be turned on and off. The atom(s) may be selected by item number, crosshair cursor, or by giving a DBA list specifier (1#) or a match string. A match string may include a wild card character (*). The display can be automatically redrawn following the label change, or you can issue a number of **xyz label** operations before redrawing the display.

xyz visible *atoms on/off redraw*

atoms atoms(s) to label (by item#, list, match string, or -1 for cursor)

on/off label operation: **1**=turn label on**, 0**=turn label off

redraw redraw status: **0**=no redraw**, 1**=redraw now to show label change

xyz visible makes atoms visible or invisible. The atom(s) may be selected by item number, crosshair cursor, or by giving a DBA list specifier (1#) or a match string. A match string may include a wild card character (*). The display can be automatically redrawn following the visibility change, or you can issue a number of **xyz visible** operations before redrawing the display.

xyz color *atoms pen redraw*

atoms atoms(s) to label (by item#, DBA, match string, or -1 for cursor)

pen color number

redraw redraw status:**0**=no redraw, **1**=redraw now to show label change

xyz color sets the display color of the specified atoms. The atom(s) may be selected by item number, crosshair cursor, or by giving a DBA list specifier (1#) or a match string. A match string may include a wild card character (*). The

display can be automatically redrawn following the visibility change, or you can issue a number of **xyz color** operations before redrawing the display.

xyz distance *atom1 atom2 symbol*

atom1 first atom (by item number, DBA list, or atom name string)

atom2 second atom (by item number, DBA list, or atom name string)

symbol symbol to receive distance between *atom1* and *atom2*

xyz distance calculates the distance between two specified atoms. Each atom may be selected by item number, DBA list, crosshair cursor, or by atom name. If both atoms exist, the interatomic distance in angstroms is returned in *symbol*. If any of the atoms is a pseudoatom, the effective distance is calculated:

Any error returns a distance of zero.

xyz pattern *atom1 atom2 pen redraw*

atom1 first atom (by item number, DBA list, or atom name string)

atom2 second atom (by item number, DBA list, or atom name match string)

pen color number

redraw redraw status: **0**=no redraw**, 1**=redraw now

xyz pattern draws a line of the specified color connecting two atoms. Each atom may be selected by item number, DBA list, crosshair cursor, or by atom name. This operator is commonly used to graphically display interactions between spin pairs (e.g., distance monitors).

xyz clear

xyz clear deletes all the entities connected to the molecule

`xyz:atoms`, `xyz:bonds`, `xyz:residues`, and `xyz:atm_atr`.

xyz current *type atoms bonds residues atom_atr patterns*

type type of the molecule (0=PDB, 1=Insight II, 2=X-PLOR, 3=MDL MOL)

atoms atom entity (xyz:atoms)

bonds bond entity (xyz:bonds)

residues residue entity (xyz:residues)

atom_atr atom attribute entity (xyz:atm_atr)

patterns pattern entity (xyz:pattern)

The **xyz current** command makes the entities connected to the molecule current. This command acts as if an xyz read command were executed, therefore this is an alternative to that command, if the molecule was read into a database during a previous session and saved. Use this command to let FELIX know that there is a valid molecule in the memory.

xyz assembly *entity_root file #_of_mol*

entity_root root name of the entity — e.g., xyz:arc, the subsequent molecules will be stored as xyz:arc1, xyz:arc2...

file name of existing Insight II archive file in sequential format (.arcs) without the suffix

#_of_mol number of molecules to read in from the .arcs file, the maximum number is 99

xyz assembly reads an ASCII file containing an ensemble of molecules in Insight II sequential archive format (.arcs). The specification should not include an extension, as the command will look for both *file*.mdf and *file*.arcs.

ze — Zero workspace

ze

ze causes the data in the workspace to be set to 0.0.

Symbol dependence

datsiz number of datapoints

See also

set — Set work space to a value

zf — Zero fill

zf *points*

points new size for data

zf expands the data in the work space to a larger size, filling the additional space with zeros. The **zero fill** command is used before Fourier transformation to improve digital resolution. The default new size is the next integral power of two that is larger than the current size.

Symbol dependence

datsiz number of datapoints

Symbols changed

datsiz number of datapoints

zgt — Zero greater than

zgt *threshold*

threshold threshold value for zeroing

zgt zeroes all points in the work space that are greater than the specified threshold value. This is a brutal way to wipe out big peaks in your spectrum.

The **zgt** command is sometimes used to remove a water peak from a spectrum in order to speed up the plotting process.

Symbol dependence

datsiz number of datapoints

datype data type

See also

zlt — Zero less than

zi — Zero imaginary

zi

zi zeroes the imaginary part of the data in the workspace.

Symbol dependence

datsiz number of datapoints

datype data type

See also

zr — Zero real

zlt — Zero less than

zlt *threshold*

threshold threshold value for zeroing

zlt zeroes all points in the work space that are less than the specified threshold. This is a drastic way to wipe out baseline noise, as well as small peaks.

Symbol dependence

datsiz number of datapoints

See also

zgt — Zero greater than

zr — Zero real

zr

zr zeroes the real part of the data in the workspace.

Symbol dependence

datsiz number of datapoints

datype data type

See also

zi — Zero imaginary

zp — Null plot

This command is similar to np, except that it does not draw the axis and labels.

B Symbol Reference

All symbols listed in this appendix are reserved symbols.

absint — Absolute intensity

This symbol controls the automatic scaling of 1D plots and 2D stack plots. By default, each plot is scaled to fill the plot area. Set **absint** before plotting.

Symbol Settings

0	scale the next plot to best fill the plot area
1	draw the next plot using the scaling from previous plot

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

sp — Stack plot

absmg1, absmg2, absmg3, absmg4 — Absolute magnitude peak search window size

This symbol controls the size of the window used to find antiphase peaks during peak picking. The search window half-width is given for each dimension to reflect the resolution of the matrix. Set **absmgn** before picking antiphase peaks.

Commands affected

pic — Peak pick and label

animat — Animation switch

This symbol controls the use of double buffered graphics for smoother display of changing plots. By default, each plot erases the frame, draws a box and axes, and then draws the spectrum. Rapid replotting causes flickering of the image. Turning on animation minimizes the flicker by not showing the new plot until it is finished.

Symbol Settings

0	no double buffering
1	use double buffering for smoother display

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

sp — Stack plot

annang — Annotation angle

This symbol controls the angle for displaying annotation text. All angles between 0° and 360° are allowed.

Symbol Settings

0	horizontal, from left to right
90	vertical, from bottom to top
270	vertical, from top to bottom

Commands affected

tex — Text annotation

gre — Greek text annotation

annasz — Annotation arrow size

This symbol controls the size of the arrow head on annotation arrows. The size is specified in inches.

Commands affected

arr — Arrow annotation

anncol — Annotation color

This symbol controls the color of subsequent annotations. Any valid pen number is a valid annotation color.

Commands affected

arr — Arrow annotation

lin — Line annotation

rec — Rectangle annotation

tex — Text annotation

gre — Greek text annotation

annfil — Annotation file

This symbol specifies the current annotation file. This file will be automatically drawn on the next plot whenever the symbol **pltann** is on, or explicitly drawn on the current plot by the **ann** command.

Commands affected

ann — Annotate plot

See also

pltann — Annotation switch

annlst — Annotation line style

This symbol controls the line style of arrow and line annotations. Lines may be solid or dashed.

Symbol Settings	
0	solid lines
1	large dashes (~ 1 per quarter inch)
2	medium dashes (~ 2 per quarter inch)
3	small dashes (~ 3 per quarter inch)

Commands affected

arr — Arrow annotation

lin — Line annotation

annpfx — Annotation prefix

This symbol specifies the directory for reading and writing annotation files.

annsiz — Annotation text size

This symbol controls the size of annotation text. The size is in inches, and determines the height and width of each character drawn.

Commands affected

tex — Text annotation

gre — Greek text annotation

annunt — Annotation units

This symbol controls the interpretation of the position parameters for all annotation commands. Each annotation command has at least one set of x,y coordinates that define the location for that annotation. The units of these position parameters are given by this symbol, and may be any one of the choices shown below.

Normalized device coordinate units (NDC) treat any region as a range from zero to one, where the lower left corner is (x=0, y=0) and the upper right corner is (x=1, y=1).

Symbol Settings	
0	NDC within the plot
1	NDC within the frame
2	axis units
3	points

Commands affected

arr — Arrow annotation

lin — Line annotation

rec — Rectangle annotation

tex — Text annotation

gre — Greek text annotation

autox, autoy, autoz — Autorotation X-, Y-, and Z-angle increments

autpse — Autorotation pause

autrot — Autorotation switch

cntrot — Autorotation count

These symbols control the autorotation of 3D plots and coordinates. Within the 3D display interface, the display can be automatically rotated for a set duration without the use of the mouse.

Symbol Settings

autrot	**0** = no autorotation, **1** = do autorotation
autpse	seconds of pause between each step of rotation
cntrot	number of steps of rotation to do
autox, autoy, autoz	the angle in degrees to turn the display by during each step of the rotation

Commands affected

cp — Contour plot

ip — Intensity plot

xyz — Atom manipulation

See also

projct — Graphics projection type

axsobj — Axis object switch

This symbol shows the display status of the plot axes in the 3D display interface.

Symbol Settings

0	Axes not visible
1	Axes visible

Changed by commands

cp — Contour plot

ip — Intensity plot

np — Null plot

pla — Redisplay 3D object

axtype — 1D Axis type

This symbol controls the axis units used on 1D plots. This affects axis labeling and the units returned by the **cur** command.

Symbol Settings	
0	display no axis labels
1	display axis labels in points
2	display axis labels in Hertz
3	display axis labels in ppm
4	display axis labels in seconds
5	display axis labels in 1/cm

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cur — Cursor control

b1size, b2size, b3size, b4size — Matrix brick size

These symbols tell the size of each brick in the current matrix. Some matrix manipulation macros can minimize the number of disk accesses by knowing the number of vectors per brick.

Changed by commands

mat — Open matrix

cmx — Close matrix file(s)

basent — Baseline points entity

This is the name of the current baseline points entity.

Commands affected

abq — Automatic selection of baseline points

bas — Baseline points manipulation

csp — Cubic spline baseline correction

pol — Polynomial baseline correction

bcfrac — Baseline correct fraction

This symbol is not currently used.

bckrad — Back-calculation cutoff radius

This symbol controls which interatomic contacts contribute to the back-calculated spectrum. Only atom pairs that are closer than this distance, in angstroms, are included in the calculation.

Commands affected

bck — Back-calculate NOE intensities

See also

minzee — Cutoff Z-magnetization for back-calculation

bigpt — Big point in workspace

This symbol contains the largest data value of all points in the current 1D plot.

Changed by commands

dr — Draw work space and stack

exp — Expanded display

ful — Full display

See also

smalpt — Small point in workspace

blkwht — Black/White reverse switch

This symbol controls the color convention used by the FELIX 2002 graphics window. The symbol must be set before the FELIX graphics window is created, and is traditionally set in the init.mac file. **blkwht** affects every command that draws anything on the graphics window.

Symbol Settings	
0	black background with white text and lines
1	white background with black text and lines

button — Button status

This symbol contains the number of the most recently pressed button in a dialog box or database spread sheet. This allows a macro to act based on which button the user pressed. For dialog boxes, the button numbers are defined in the menu file that creates that dialog box.

Symbol Settings

0	quit, cancel, or the <Esc> key
1	save, OK, or the <Enter> key
2...	available, as defined by each dialog box

Changed by commands

mnu — Menu manager

dba — Database facility (dba ent edit Entity Viewer)

dba — Database facility (dba ent select Entity Selector)

celpxx — Cell X pixels

celpxy — Cell Y pixels

These symbols contain the size of a character cell in pixels, as used by the Menu Interface. The symbols are useful for converting between pixel and character locations on the FELIX graphics window. **celpxx** and **celpxy** are set when the FELIX graphics window is first opened. The size of a character cell is based upon, and slightly larger than, the font size.

center — Center plot switch

This symbol controls the centering of 1D plots.

Symbol Settings

0	no centering, zero is near the bottom
1	center the plot, zero is at the middle

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

See also

ovrlap — Stack overlap

scale — Plot scale factor

chi — Minimum chi-square value

This symbol stores the minimum chi-square value calculated by the **chi** command. The **flt** command uses this value to determine baseline segments of a spectrum.

Commands affected

flt — FLATT baseline flattening

clmode — Contour level mode

This symbol controls the method used to space multiple contour levels on a plot.

Symbol Settings

0	linear spacing
1	geometric spacing

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

See also

conmod — Contour spacing modifier

drwclv — Draw contour levels switch

nlevel — Number of contour levels

cntrot — Autorotation count

See autox.

colcur — Color of cursor

This symbol controls the color of all cursors used by FELIX 2002, to the extent that the computer hardware allows. Any valid pen number is a valid cursor

color. This symbol must be set before the FELIX graphics window is created. It is traditionally defined in the init.mac file. Be forewarned that the number of bitplanes in your computer and the setting of the symbol **blkwht** can affect the appearance of cursor colors.

Commands affected

cur — Cursor control

conmod — Contour spacing modifier

This symbol controls the spacing of multiple contour levels on a plot. Each successive contour level is based on a factor of the previous level. This symbol is that factor. With linear spacing (**clmode** = 0) the factor is additive, while for geometric spacing (**clmode** = 1) the factor is multiplicative. It must have a value greater than 0.0 for **clmode** = 0, and greater than 1.0 for **clmode** = 1; the larger the value, the wider the spacing between levels.

Commands affected

dr — Draw work space and stack

cp — Contour plot

ip — Intensity plot

See also

clmode — Contour level mode

drwclv — Draw contour levels switch

nlevel — Number of contour levels

contyp — Contour interpolation type

This symbol controls the method of interpolation used when contouring.

Symbol Settings	
0	no interpolation (faster and more angular)
1	cubic-spline interpolation (slower and smoother)

Commands affected

cp — Contour plot

ip — Intensity plot

cutoff — Cutoff for stack plot

This symbol controls the maximum peak height of stack plots by clipping off the tops of peaks that are taller than a given height. This helps to keep small peaks visible by preventing tall peaks from completely filling the plot. The cutoff height is given in inches.

Symbol Settings

0	no clipping, draw entire peak
N	clip all peaks taller than *N* inches to be exactly *N* inches

Commands affected

sp — Stack plot

cycle — Color cycle length

cycle controls the cycling of colors when drawing multiple 1D buffers or ND contours. By default, each plot is drawn in only one color. To make multiple colored plots, increase the value of **cycle** to the number of different colors to use.

Commands affected

dr — Draw work space and stack

ovc — Overlay contour plot

cp — Contour plot

ip — Intensity plot

sp — Stack plot

See also

nlevel — Number of contour levels

pennum — Starting color

stack — Stack depth

d1size, d2size, d3size, d4size — Matrix size

These symbols contain the size in points of the current matrix, for each dimension.

Changed by commands

mat — Open matrix

cmx — Close matrix file(s)

See also

dimen — Number of Matrix Dimensions

d1vect, d2vect, d3vect, d4vect — Current vector

These symbols contain the indices of the vector most recently loaded out of the matrix. Note that one index is zero, denoting the dimension that the vector spans.

Changed by commands

loa — Load vector from matrix

datfil — 1D Data file

This symbol contains the name of the current 1D data file.

Changed by commands

re — Read a file (old format)

rn — Read file (new format)

wr — Write a file (old format)

wn — Write a file (new format)

datpfx — 1D Data file prefix

This symbol specifies the directory for reading and writing 1D data files.

datsiz — Data size

datsiz specifies the size of the spectrum in the workspace, in points. This is also the size of any spectra in the buffers. The data size is used or set by virtually all the 1D commands.

Changed by commands

dbl — Double data size

hav — Halve data size

lpl — Linear predict last points

re — Read a file (old format)

rn — Read file (new format)

zf — Zero fill

See also

datype — Data type

datype — Data type

This symbol defines the type of data in the work space and buffers. The data type is used or set by most of the 1D commands.

Symbol Settings

0	real data
1	complex data
2	xy pairs

Changed by commands

cpl — Real to complex

hft — Hilbert transform

lvo — Load volume time course

re — Read a file (old format)

red — Reduce complex to real

rn — Read file (new format)

xyp — X,Y data pair manipulation

dbafil — Database file

This symbol specifies the name of the current database file.

dbapfx — Database file prefix

This symbol specifies the directory for reading and writing database files.

deltax, deltay — Delta x and y for stack plot

These symbols control the skew and tilt of stack plots. Valid values are in the range 1.0 to -1.0 and represent the fractional shift per data point.

Commands affected

sp — Stack plot

dimen — Number of Matrix Dimensions

This symbol contains the number of dimensions in the current matrix.

Symbol Settings

1	no matrix is currently open
2	current matrix is 2D
3	current matrix is 3D
4	current matrix is 4D

Changed by commands

mat — Open matrix

cmx — Close matrix file(s)

dimplt — Number of plot dimensions

This symbol contains the number of dimensions in the current plot.

Changed by commands

cp — Contour plot

ip — Intensity plot

np — Null plot

See also

plotd1, plotd2, plotd3, plotd4 — Plot dimension

disply — Current display type

This symbol contains the type of the current plot. This is useful for knowing what kind of picture is in the current frame.

Symbol Settings

0	no current plot
1	1D plot
2	contour or intensity plot
3	stack plot
5	tile plot
6	XY pairs plot
7	atom plot

8	object plot
9	object with Axes plot

Changed by commands

clr — Clear frame

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

np — Null plot

sp — Stack plot

xyp — X,Y data pair manipulation

xyz — Atom manipulation

draw3d — 3D capability switch

This symbol flags whether FELIX can perform 3D rotations on this computer. Note that FELIX does rotations using 3D hardware on some computers and emulates this functionality in software on other computers. This symbol is set when the FELIX graphics window first opens.

Symbol Settings

0	3D rotations not supported
1	3D rotations supported

drwbox — Draw box switch

This symbol controls whether a box is drawn around plots.

Symbol Settings

0	no box
1	draw a box

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

np — Null plot

xyp — X,Y data pair manipulation

drwclv — Draw contour levels switch

This symbol controls whether contour levels are drawn on 1D plots.

Symbol Settings

0	do not draw levels
1	draw contour levels

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

drwpks — Draw peaks switch

This symbol controls the automatic drawing of peaks on the current plot. When on, **drwpks** causes the plot to automatically draw the current peaks. This is very useful for hard copy plots.

Symbol Settings

0	do not draw peaks
1	draw peaks
2	draw peaks colored by plane position
3	draw peaks colored by assignment status
4	draw peaks colored by protopattern

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

hcp — Hard copy plot

np — Null plot

drwxpk — Draw cross peaks switch

This symbol is not currently used. See drwpks — Draw peaks switch.

dspmod — Display mode

This symbol controls the type of data displayed in a 1D plot. By default, FELIX displays only the real components of 1D spectra.

Symbol Settings

0	real data only
1	imaginary data only
2	real and imaginary, side by side

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

erase — Erase display switch

This symbol controls whether the display is erased prior to drawing a plot. By default, FELIX clears the display before each plot is drawn.

Symbol Settings

0	no auto erase
1	always erase

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

np — Null plot

etcpfx — Runtime files prefix

This symbol specifies the directory for reading and writing various runtime temporary files.

first — First point

This symbol controls the extent of data displayed by a 1D expanded plot. **first** holds the first data point to be displayed.

Commands affected

dr — Draw work space and stack

exp — Expanded display

See also

last — Last point

flxver — FELIX version number

This symbol contains the version number of the FELIX program. **flxvr** is set when FELIX starts.

fontsw — Font switch

This symbol controls the method for generating text in PostScript hard copy plots. By default, FELIX strokes out characters using a stroke generator. This is excessively bulky and inefficient when the hard copy device has scalable fonts.

Symbol Settings

0	stroke characters
1	use PostScript font

Commands affected

hcp — Hard copy plot

fontsz — Font size

This symbol is not currently used. The font closest to the text size requested is used.

frsize — Frame size

This symbol contains the maximum data size that can be held in the workspace or any buffer. This also is the maximum number of items that can be held in a database list. Reconfiguring memory is the only way to change the frame size.

Changed by commands

cfg — Configure memory

See also

nframe — Number of stack frames

gbroad — Gaussian coefficient

This symbol determines the degree of Gaussian character that the **Lorentzian/Gaussian** resolution enhancement apodization function possesses. This function is intended to behave as the **gm** function on a Bruker spectrometer.

Commands affected

gm — Gaussian/Lorentzian window

gibbs — Gibbs filter switch

Controls the Gibbs filter function of the fast Fourier transform operation. The Gibbs filter divides by two the first and last points of the data in the time domain.

Symbol Settings

0	Gibbs filter off
1	Gibbs filter on

Commands affected

bft — Bruker-Fourier transform

ft — Fast Fourier transform

rft — Real Fourier transform

hft — Hilbert transform

dft — Fast Fourier transform of digitally oversampled data

ift — Inverse Fourier transform

graysc — Gray scale switch

This symbol controls the PostScript plot output rendering. When gray scale is off, the plot is rendered in color using the FELIX 2002 pen colors. Turning gray scale on causes FELIX to render the plot in a graded gray scale.

Symbol Settings

0	gray scale off
1	gray scale on

Commands affected

hcp — Hard copy plot (PostScript Only)

grid — Grid switch

grid controls the drawing of a grid on plots. The value of **grid** determines the number of grid lines or tick marks that appear for each axis label.

Symbol Settings	
0	no grid
N	*N* grid lines per label
-N	*N* tick marks per label

Commands affected

dr — Draw work space and stack

cp — Contour plot

ip — Intensity plot

sp — Stack plot

gridco — Grid color

Sets the color to use for grids. This is usually set to 15 in mninit.mac

gridst — Grid style

Sets the style of grid lines. By default, this is set to -1 in mninit.mac, which produces dashed lines. If **gridst** is greater than or equal to zero, then straight lines are produced.

hafwid — Halfwidth factor

Controls the size of cross peak footprint boxes. The footprint size is twice the product of **hafwid** and the half-width at half-height values that are stored by the peak picker in the cross peak entity. Several commands operate on the area inside the peak footprints, so they are influenced by this symbol.

Commands affected

drx — Display cross peaks

fit — Fit 1D peaks

md — Model data

til — Tile plot

vol — Integrate cross peak volumes

harddv — Hardcopy destination

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

hardmo — Hardcopy mode

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

hardx0, hardy0, hardxs, hardys — Hardcopy origin and size

These symbols are not currently used.

Commands affected

hcp — Hard copy plot

hfwid1, hfwid2, hfwid3, hfwid4 — Minimum peak half width

This symbol controls the peak pick command. **hfwidn** defines the minimum halfwidth of a valid peak, in points, for all dimensions. Any peak with less than the minimum halfwidth in any dimension is not entered into the cross peak entity.

Commands affected

pic — Peak pick and label

hilim — Current plot region high limits

See lolim below.

hndshk — HPGL plotter handshake

This symbol determines the handshaking values set in the HPGL hardcopy file. Setting **hndshk** to **0** instructs FELIX 2002 to use the handshaking values that were specified in versions 230 and earlier of FELIX. These values set software XON/XOFF handshaking with the default logical buffer size and the plotter sends XOFF to the computer when 161 free bytes remain in the plotter buffer. A more conservative handshaking scheme occurs when **hndshk** is set to **1**. This again selects XON/XOFF handshaking but instructs the plotter to send XOFF when 1023 free bytes remain in the buffer, i.e., after one byte enters the buffer. If neither of these software handshaking choices is satisfactory, set **hndshk** to **3**, which instructs FELIX to omit handshaking initialization commands, and create a file of the appropriate handshaking initialization instructions called `pref.31` in the directory where you run FELIX. The contents of `pref.31` will be appended to the beginning of the hard copy output file. Finally, setting **hndshk** to **2** disables software handshaking and enables hardware handshaking.

Commands affected

hcp — Hard copy plot

inbias, inslop — Integral bias and slope corrections

These symbols determine the amount of bias and slope correction applied to the calculation of 1D integrals. Note that accepted values are 0.0 to 1.0.

Commands affected

int — Integral

seg — Integral segments

See also

intolp — Integral overlap

segint — Segmented integral switch

intolp — Integral overlap

This symbol controls the extent to which drawn integrals overlap the spectrum. All values between zero and one are valid.

Symbol Settings

0	no overlap
1	full overlap

Commands affected

dr — Draw work space and stack

See also

segint Segmented Integral Display Mode

item — Menu item

This symbol returns the line number of the most recently selected menu on which the user clicked the left mouse button. When no item is selected, the symbol has the value 0. This symbol is set by the menu wait command.

Changed by commands

mnu — Menu manager

See also

menu — Menu name selected

iwidth — Interval width for baseline correction

This symbol controls noise effects in baseline correction by defining the number of baseline data values that are averaged about each baseline point to determine the baseline value used for the correction. **iwidth** point values above and below the baseline point are used.

Commands affected

csp — Cubic spline baseline correction

pol — Polynomial baseline correction

keyhit — Keyboard character struck

This symbol returns the keyboard character the user struck in response to any of several commands. The possible values of **keyhit** include: **escape**, **mouse**, **motion**, **null**, **key_a...key_z**, **and key_0...key_2**. Any command that accepts the <Esc> key to terminate that command will also set **keyhit** to show the termination event.

Changed by commands

cur — Cursor control

mnu — Menu manager

cp — Contour plot

ip — Intensity plot

last — Last point

This symbol determines the last point of an expanded 1D plot.

Commands affected

dr — Draw work space and stack

exp — Expanded display

See also

first — First point

lbroad — Line broadening

This symbol determines the amount of line broadening, in Hertz, applied by apodization functions.

Commands affected

em — Exponential multiply

gm — Gaussian/Lorentzian window

mf — Matched filter

level — Contour level

This symbol controls the first contour level plotted. The product of the reserved symbols **level** and **mscale** determines the absolute data value of the

first contour level. All matrix data points greater than that value will be plotted.

Commands affected

cp — Contour plot

ip — Intensity plot

pic — Peak pick and label

See also

mscale — Matrix scale factor

posneg — Negative level switch

linpts — Lines/Points for draw command

This symbol controls whether the **dr** command displays data as isolated points, line segments or histograms. This symbol is also used by **xyp** to control whether x,y pairs are displayed as lines and/or points with error bars, or histograms.

Symbol Settings	
0	lines
1	points
2	lines and points (xyp only)
3	histogram (in 1D as MS spectrum, in xy as histogram plot)
4	histogram (with error bars only in xy plot)

Commands affected

dr — Draw work space and stack

xyp — X,Y data pair manipulation

loc3x0, loc3x1, loc3y0,loc3y1, loc3z0, loc3z1 — 3D Locator endpoint coordinates

These symbols are not currently used.

lolim1, hilim1, lolim2, hilim2, lolim3, hilim3, lolim4,hilim4 — Current plot region low and high limits

These symbols contain the data point limits of the current matrix plot region. In addition to reporting the current limits, they can be set by the user to affect the limits of the subsequent plot.

Changed by commands

lim — Matrix limits

mat — Open matrix

macfil — Macro file

macfil defines the name of the current macro file.

Changed by commands

ex — Execute a macro

exr — Execute a macro and return

rm — Read macro

macpfx — Macro file prefix

This symbol specifies the directory for reading and writing macro files.

matfil — Matrix file

This symbol contains the name of the current matrix file.

Changed by commands

mat — Open matrix

cmx — Close matrix file(s)

matpfx — Matrix file prefix

This symbol specifies the directory for reading and writing matrix files.

maxvol — Maximum volume slots in entity

This symbol controls the number of volume slots created when a new volume entity is built. The number of slots can not be changed once the volume entity exists. The default is six slots, and the maximum value is sixteen slots. \

mdlcf1, mdlcf2 — Model data coefficients 1 and 2 (matrix factor)

mdlpke — Model data peak entity ID

mdlvle — Model data volume entity ID

mdlvsl — Model data volume slot number

These symbols contain the state of the most recent model data command. They correspond to the five parameters on the **md** command line.

Changed by commands

md — Model data

menu — Menu name selected

This symbol contains the name of the most recently selected menu on which the user clicked the mouse button. When no menu is selected, the symbol has the value **null**. This symbol is set by the menu wait command.

Changed by commands

mnu — Menu manager

See also

item Menu Item

mframe — Matrix framesize

This symbol controls the maximum size of matrix data files. When building a matrix, the size of the file on disk will not be allowed to exceed this size, in megabytes. Multiple disk files will be created, with the possibility of each file being on different devices.

Commands affected

bld — Build a matrix file

minzee — Cutoff Z-magnetization for back-calculation

This symbol controls the number of cross peaks generated by a back-calculation run. In theory, the potential exists to create cross peaks for all pairs of protons in the molecule. In practice, we generate only those peaks that contain more than some minimum amount of magnetization. The units are a fraction of one, with the default value being 0.001.

Commands affected

bck — Back-calculate NOE intensities

See also

bckrad — Back-calculation cutoff radius

mnufil — Menu file

This symbol contains the name of the current menu file.

Changed by commands

mnu — Menu manager

mnumod — Default frame position

This symbol specifies that how far the frame left border should be (in characters) from the window left edge. The default is **1** character, and is set in `mninit.mac`.

mnupfx — Menu file prefix

This symbol specifies the directory for reading menu files.

mscale — Matrix scale factor

This symbol controls the interpretation of what constitutes the base contour level. All data points in the current matrix that have a value greater than the product of **level** and **mscale** will be drawn by **cp** or **ip**. This symbol is not usually changed by the user.

Commands affected

cp — Contour plot

ip — Intensity plot

See also

level — Contour level

msgfil — Message file

This symbol contains the name of the most recently read message file.

msgpfx — Message file prefix

This symbol specifies the directory for reading message files.

msgc1, msgc2, msgi1, msgi2, msgr1, msgr2 — Character, integer, and real message parameters

These symbols are used by the FELIX message utility to give error and warning messages to the user that contain information specific to the most recent error or warning. These symbols are not changed by the user.

ndctyp — Normalized device coordinate type

This symbol controls the meaning of **ndc** with respect to values given to and gotten from the cursor control command. **ndc** treats a region as going from zero to one, with the lower left at (x=0, y=0) and the upper right at (x=1,y=1). You can set the **ndc** region to be either the plot box or the entire current frame.

Symbol Settings	
0	NDC within the current plot
1	NDC within the current frame

Commands affected

cur — Cursor control

See also

x0pnt, y0pnt, x1pnt, y1pnt — Cursor position

y1pnt

newhpglc — control contouring in HPGL plots

If **newhpglc** is not set (which is the default), HPGL plots use the new contouring method. Or you can specifically set **newhpglc**:

Symbol Settings

0	Do old-style contouring.
1	Do new-style contouring.

newpostc — control contouring in PostScript plots

If **newpostc** is not set (which is the default), PostScript plots use the new contouring method. Or you can specifically set **newpostc**:

Symbol Settings

0	Do old-style contouring.
1	Do new-style contouring.

nframe — Number of stack frames

This symbol contains the total number of 1D buffers not including the workspace. Therefore when **nframe** is 5, FELIX is configured for the workspace and 5 buffers.

Changed by commands

cfg — Configure memory

See also

frsize — Frame size

nlevel — Number of contour levels

This symbol controls the maximum number of contours drawn on a plot.

Commands affected

cp — Contour plot

ip — Intensity plot

See also

conmod — Contour spacing modifier

level — Contour level

norval — Normalization value

This symbol is not currently used. The command to normalize integral segments uses an explicit command line argument.

See also

seg — Integral segments.

objmem — Object memory size limit

This symbol controls the maximum amount of memory that FELIX 2002 will use to store graphical objects. Earlier versions of FELIX were prone to crashing if all the computer memory was used. Set the object memory size limit, in bytes, before the FELIX graphics window is first created. This is usually done in the init.mac file. Setting the limit too low can impede working with objects, while setting the limit too high can allow FELIX to consume all computer memory and crash.

order1, order2, order3, order4 — Matrix dimension order

This symbol contains the order in which matrix dimensions will be plotted.

Changed by commands

ord — Matrix dimension order

Commands affected

cp — Contour plot

ip — Intensity plot

See also

plotd1, plotd2, plotd3, plotd4 — Plot dimension

orient — Postscript orientation

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

ovrlap — Stack overlap

This symbol controls the overlap of the work space and buffers when displaying a 1D plot.

Symbol Settings

0.0	no overlap
1.0	complete overlap

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

See also

stack — Stack depth

stkord — 1D Stack display order

paphgt — Paper height

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

See also

hardmo — Hardcopy mode

papwid — Paper width

pltorg — Plot origin

papwid — Paper width

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

See also

hardmo — Hardcopy mode

paphgt — Paper height

pltorg — Plot origin

pennum — Starting color

This symbol specifies the rendering color (i.e., the pen number) for data displays. If the pen cycle is greater than 1, the specified pen number serves as the initial color in the drawing cycle. The default pen number values are mapped to **rgb** colors as shown below.

Symbol Settings	
0	black
1	white
2	red
3	green
4	blue
5	violet
6	cyan
7	yellow
15	gray

On some computers, the pen number values are interpreted using modulo 7 arithmetic, that is, **pennum** = 8 is identical to **pennum** = 1.

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

sp — Stack plot

See also

cycle — Color cycle length

phase0 — Zero-order phase

This symbol specifies the zero-order (constant phase) parameter in degrees. Useful values range from -360 through 360 (phase0 = 180 implies that the data are to be negated).

Changed by commands

aph — Autophase spectrum

ra — Read ASCII data

re — Read a file (old format)

rn — Read file (new format)

rph — Real-time phase

Commands affected

ph — Phase correction

See also

phase1 — First-order phase

phase1 — First-order phase

This symbol specifies the first-order (frequency-dependent phase) parameter in degrees. The symbol value indicates the frequency dependent phase difference across the current workspace.

Changed by commands

aph — Autophase spectrum

ra — Read ASCII data

re — Read a file (old format)

rn — Read file (new format)

rph — Real-time phase

Commands affected

ph — Phase correction

See also

phase0 — Zero-order phase

picent — 1D peaks entity

This symbol designates the name of the current 1D peak entity.

Commands affected

pic — Peak pick and label

pksent — Cross peak entity

This symbol designates the name of the current **nd** peak entity.

Commands affected

drx — Display cross peaks

pic — Peak pick and label

pksobj — Peaks object switch

This symbol records whether the cross peak object was enabled / disabled in the 3D real time display interface.

Symbol Settings

0	peaks not visible
1	peaks visible

Changed by commands

cp — Contour plot

ip — Intensity plot

np — Null plot

pla — Redisplay 3D object

pkstyl — 1D peak display style

This symbol specifies the manner in which 1D peak extrema are marked when 1D peaks are drawn.

Symbol Settings

0	line only
1	line and arrow
2	arrow only
3	marks the peak width

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

pic — Peak pick and label

See also

drwpks — Draw peaks switch

pkunit — Units for picking peaks

This symbol designates the units used to display the position of 1D peak extrema.

Symbol Settings

0	none
1	points
2	Hertz
3	ppm

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

pic — Peak pick and label

plotd1, plotd2, plotd3, plotd4 — Plot dimension

These symbols record the displayed orientation of ND data. The default display orientation places d1 along the x axis (horizontal**: plotd1**), d2 along the y axis (vertical: **plotd2**), and d3 along the z axis (normal to the video plane: **plotd3**). This orientation may be changed by the **ord** command and the current plot limits. The subsequently displayed orientation is recorded in the **plotdn** symbols. The axis orientations may thus be determined by consulting the **plotdn** symbols (e.g., **plotd1** = 2 implies that the d2 dimension is currently displayed along the x axis, whereas **plotd2** = 3 implies that the d3 dimension is currently displayed along the y axis.

Changed by commands

cp — Contour plot

ip — Intensity plot

np — Null plot

See also

dimplt — Number of plot dimensions

pltann — Annotation switch

This symbol specifies whether or not annotations are automatically displayed on the current plot.

Symbol Settings

0	no auto annotate
1	auto annotate

Commands affected

cp — Contour plot

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

ip — Intensity plot

np — Null plot

See also

annfil — Annotation file

pltmod — Plot mode

This symbol records the type of graphics display employed. The symbol value is determined by which version (GL or X11) of FELIX is executed, and is set when the first graphics window is opened. If FELIX is executed without graphics, the value of the symbol should be set to -1 (def **pltmod** -1) in the init.mac file.

Symbol Settings

-1	open no graphics
13	Silicon Graphics GL
15	X11 graphics

See also

hardmo — Hardcopy mode

pltobj — Plot object switch

This symbol records whether the data object was enabled / disabled in the 3D real-time display interface.

Symbol Settings	
0	contours not visible
1	contours visible

Changed by commands

cp — Contour plot

ip — Intensity plot

pla — Redisplay 3D object

pltorg — Plot origin

This symbol is not currently used.

Commands affected

hcp — Hard copy plot

See also

hardmo — Hardcopy mode

paphgt — Paper height

papwid — Paper width

pltusr — Position plot manually switch

This symbol toggles manual control over plot sizing and positioning.

Symbol Settings	
0	control of plot size and offset defaults to fill frame
1	enforce user-defined values for plot origin and size

Commands affected

cp — Contour plot

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

ip — Intensity plot

np — Null plot

See also

xzero, yzero — Plot origin

xsize, ysize — Plot size

posneg — Negative level switch

This symbol specifies whether or not to display negative contour levels in 2D and 3D contour plots. Negative contours can be displayed in a different color from positive contours by setting the symbol **cycle** to 2.

Symbol Settings

0	positive contours only
1	negative and positive contours

This same symbol also specifies whether or not to pick negative peaks in the 1D peak pick command.

Symbol Settings

0	pick positive peaks only
1	pick negative and positive peaks

Commands affected

cp — Contour plot

hcp — Hard copy plot

pic — Peak pick and label

See also

cycle — Color cycle length

level — Contour level

proj1 — Type of 1D projection onto x axis

proj2 — Type of 1D projection onto y axis

This symbol designates the source of the 1D display projections that attend the x and y axes of a 2D plot.

Symbol Settings

0	no projection
1	sum projection
2	"skyline" or "shadow" projection
3	use contents of buffer #1
4	use contents of buffer #2
5	use contents of buffer #3

Commands affected

cp — Contour plot

ip — Intensity plot

See also

projsz — Size of 1D projections

projct — Graphics projection type

This symbol specifies the display (projection) mode.

Symbol Settings

2	draw everything as a flat 2D plane
3	draw 3D as a projection onto a flat 2D plane
4	draw 3D with a real-time rotation interface

Note that the real-time rotation (play) mode is enabled only when **projct** = 4.

Commands affected

cp — Contour plot

ip — Intensity plot

np — Null plot

pla — Redisplay 3D object

xyz — Atom manipulation

See also

steang — Stereo angle

stereo — Stereo switch

stesep — Stereo separation

xangle, yangle, zangle — Euler angles

projsz — Size of 1D projections

This symbol designates the size of the x and y projections as a fraction of the plot sizes in their respective dimensions. Values from 0.1 to 1.0 are valid. This lets you control how much room is reserved for the projections.

Commands affected

cp — Contour plot

ip — Intensity plot

See also

proj1 — Type of 1D projection onto x axis

proj2 — Type of 1D projection onto y axis

pwidth — Pen width

This symbol defines the pen width in inches. This affects all axis labels, peak labels, and annotations.

See also

thick — Character thickness

redraw — Automatic backing store of plots

This symbol enables / disables automatic storage of the contents of frames. When **redraw** = 1, this feature provides for a rapid recovery of the frame contents following many kinds of forced (asynchronous) redraw events (like pushing and popping the FELIX window; an exception would be resizing the FELIX window). On certain slower platforms, enabling this feature may lead to an apparent decrease in performance due to the computer's slower storage speed.

Symbol Settings	
0	no automatic backing store
1	automatic backing store

Commands affected

cp — Contour plot

dr — Draw work space and stack

exp — Expanded display

ful — Full display

ip — Intensity plot

drx — Display cross peaks

refpt — Reference point

This symbol records the data point position corresponding to the reference shift value for the current contents of the workspace and all buffers.

Changed by commands

re — Read a file (old format)

rn — Read file (new format)

ref — Set shift reference

See also

axtype — 1D Axis type

refsh — Reference shift

sfreq — Spectrometer frequency

swidth — Spectral width

refsh — Reference shift

This symbol records the chemical shift value corresponding to the reference point for the current contents of the workspace and all buffers. Note that the reference chemical shift is always stored in Hertz.

Changed by commands

re — Read a file (old format)

rn — Read file (new format)

ref — Set shift reference

See also

axtype — 1D Axis type

refpt — Reference point

sfreq — Spectrometer frequency

swidth — Spectral width

rowinc — Row increment

This symbol specifies the vector increment for stack plots and 3D contour plots. **rowinc** = 1 implies that every vector is drawn, whereas a **rowinc** = 2

implies that every other vector will be drawn. For 3D contour plots, **rowinc** affects the density of the "chicken-wire" mesh which constitutes the cross peak rendering. It is possible for cross peak contours to not appear if **rowinc** is larger that the diameter of the cross peak footprint. In both plot types, the greater the value of **rowinc**, the faster the plot completes (with some loss of detail).

Commands affected

sp — Stack plot

cp — Contour plot (ND displays only)

scale — Plot scale factor

This symbol specifies the plot scale of 1D displays and stack plots. A symbol value of **1** ensures auto-scaling of the data to fit the frame when absolute intensity mode is disabled. Scale values other than **1** apply the implied multiplicative scaling.

Symbol Settings	
1	auto-scale data to fit frame if **absint** = **0**

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

sp — Stack plot

schfil — Schema file

This symbol specifies the name of the current schema file.

schpfx — Schema file prefix

This symbol specifies the directory (prefix) for reading schema files.

Commands affected

dba — Database facility

segent — Integral segments entity

This symbol designates the entity in which output from the integral segments utility is stored.

Commands affected

seg — Integral segments

segint — Segmented integral switch

This symbol defines segmented integral display attributes.

Symbol Settings

0	no integral
1	integrate entire spectrum
2	integrate over segments, no values displayed
3	integrate over segments, display values above data
4	integrate over segments, display values below data

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

seg — Integral segments

See also

intolp — Integral overlap

sfreq — Spectrometer frequency

This symbol records the spectrometer frequency value for the contents of the work space and buffers.

Changed by commands

ra — Read ASCII data

re — Read a file (old format)

rn — Read file (new format)

Commands affected

ppm — Convert Between points and PPM

ref — Set shift reference

See also

swidth — Spectral width

slant — Character slant angle

This symbol specifies the character slant angle in degrees. This affects all text including axis and peak labels, and annotations.

Commands affected

gre — Greek text annotation

tex — Text annotation

See also

thick — Character thickness

smalpt — Small point in workspace

This symbol records the minimum data value of all points in the current 1D display.

Changed by commands

dr — Draw work space and stack

exp — Expanded display

ful — Full display

See also

bigpt — Big point in workspace

stack — Stack depth

This symbol specifies the number of buffers displayed. A symbol value of zero implies that only the work space will be displayed, whereas a symbol value of one implies that one buffer and the work space will be displayed. The maximum value of **stack** is equal to **nframe**.

Changed by commands

der — Derivative

int — Integral

pop — Pop the display stack

psh — Push the work space onto the buffer stack

xyp — X,Y data pair manipulation

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

See also

cycle — Color cycle length

ovrlap — Stack overlap

stkord — 1D Stack display order

status — Command status

This symbol records whether or not a FELIX command completed without error. Many FELIX commands are capable of indicating their successful (or unsuccessful) completion using this symbol.

Symbol Settings

0	successful completion
1	general error, command failed
other	special error of some type

See the individual commands for explanation of special error numbers. When an error occurs, FELIX reports:

```
Error in Command...
```

to the parent window. It is the user's (or macro's) job to explicitly clear the status flag by entering:

```
def status >0
```

steang — Stereo angle

This symbol specifies the stereo angle for stereo plots in degrees. A value of 6° works well for most people. Note that this symbol's value is reset to 6° by all display commands (see below).

Commands affected

cp — Contour plot

ip — Intensity plot

pla — Redisplay 3D object

xyz — Atom manipulation

stereo — Stereo switch

This symbol specifies whether (**stereo** = 1) or not (**stereo** = 0) to display a 3D object as a stereo pair.

Commands affected

cp — Contour plot

ip — Intensity plot

pla — Redisplay 3D object

xyz — Atom manipulation

stesep — Stereo separation

This symbol specifies the separation between individual components of a stereo pair display of a 3D object in object point units. The symbol value is determined by the display utility, and this value will be recalculated by all display commands (see below).

Commands affected

cp — Contour plot

ip — Intensity plot

pla — Redisplay 3D object

xyz — Atom manipulation

stkord — 1D Stack display order

This symbol designates whether the stack of buffers will be displayed with the highest numbered buffer at the top (**stkord** = 0) or with the lowest number buffer at the top (**stkord** = 1).

Symbol Settings	
0	display stack bottom next to work space
1	display stack top next to work space

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

swidth — Spectral width

This symbol specifies the spectral width in Hertz. Note that this symbol is set by the **re** and **rn** commands using the value in the data file header.

Changed by commands

ra — Read ASCII data

re — Read a file (old format)

rn — Read file (new format)

Commands affected

em — Exponential multiply

gm — Gaussian/Lorentzian window

gmh — Gaussian multiply in Hz

ppm — Convert Between points and PPM

ref — Set shift reference

See also

sfreq — Spectrometer frequency

taucee — Correlation time

This symbol specifies the correlational time in nanoseconds (that is, the time it takes a molecule to rotate one radian) and is used to conduct back-calculation simulations.

Commands affected

bck — Back-calculate NOE intensities

thick — Character thickness

This symbol specifies the number of "restrokes" for pen plotters. A symbol value of 0 implies a single stroke. This affects all text, including axis and peak labels.

Commands affected

gre — Greek text annotation

tex — Text annotation

See also

slant — Character slant angle

thresh — Threshold for 1D peakpick

This symbol specifies the threshold data value for picking 1D extrema. Any peaks with heights below this threshold are not picked.

Commands affected

pic — Peak pick and label

title — Spectrum title

This symbol designates a plot title, but is not currently used.

tphas0, tphas1 — Total zero and first-order phase

These symbols are not currently used.

value — Data value

This symbol is not currently used.

vector — Vectors in bundle

This symbol contains the total number of vectors along the specified dimension of the matrix. The bundle mode command sets the value of **vector**.

Changed by commands

bun — Set bundle mode

verify — Macro verification mode

This symbol controls the diagnostic monitoring of macro execution. This is useful for debugging macros, or for tracing the flow of macro and menu file calls. All output generated goes to the output window..

Symbol Settings	
0	silent, no diagnostics

1 display only macro and menu file names
2 display every command line as it executes

volent — Volume entity name

This symbol contains the name of the current volume entity. The volumes calculated by the **vol** command are stored in **volent**.

Commands affected

fit — Fit 1D peaks

md — Model data

vol — Integrate cross peak volumes

volobj — Volume object switch

This symbol is not currently used.

wshift — Sinebell shift

This symbol is not currently used.

wskew — Sinebell skew

This symbol is not currently used.

x0pnt, y0pnt, x1pnt, y1pnt — Cursor position

These symbols interact with the locator device using the **cur** command. The symbols are used to set the cursor position when pre-positioning the cursor, and to return the cursor position when the **cur** command is completed. The units for the cursor position in both cases depend upon the map argument given to the **cur** command. **Cur** commands that return a single point place the cursor position values in **x0pnt** and **y0pnt**. Commands that return a pair of points fill **x1pnt** and **y1pnt** too.

The **mnu wait** command sets the values of **x0pnt** and **y0pnt** to the pixel values of the cursor position when the menu item is selected.

Changed by commands

cur — Cursor control

mnu — Menu manager

Commands affected

cur — Cursor control

pic ND Peakpick (pick modes 4 and 5 only)

xangle, yangle, zangle — Euler angles

These symbols contain the angle of rotation about the x, y, and z axes applied to the matrix displayed by the 3D display interface. The angles are given in units of degrees.

When **projct** = 3, these angles control the projection of 3D plots onto the 2D display surface.

Commands affected

cp — Contour plot

ip — Intensity plot

pla — Redisplay 3D object

xyz — Atom manipulation

xcells, ycells — Number of character cells

These symbols provide the width and height of the current FELIX window in character cells. These are the units for menus and dialog boxes. The symbols are set by FELIX when the window is created or resized.

xpixel, ypixel — Number of pixels

These symbols provide the FELIX window size in pixels. These are set by the user in the init.mac file at startup to define the initial size of the FELIX window. Thereafter, using the window manager widgets to resize the FELIX window updates these symbols.

xpklbl — Cross peak label switch

This symbol controls the display of cross peak labels in 2D and ND plots.

Symbol Settings	
0	no labels
1	peak item numbers

2	peak assignments
-1	draw only crosshair, no box around peak

Commands affected

drx — Display cross peaks

hcp — Hard copy plot

See also

drwpks — Draw peaks switch

xsize, ysize — Plot size

These symbols control the size of the next plot, or contain the size of the current plot, depending on the value of **pltusr**. Sizes are in inches.

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

cp — Contour plot

ip — Intensity plot

sp — Stack plot

See also

pltusr — Position plot manually switch

xzero, yzero — Plot origin

xwalk, ywalk — Walking menu position

These symbols return a position near the cursor in character cell units when the user clicks the mouse button on a menu selection. The position represents the best location to place the next menu to create the appearance of walking menus.

Changed by commands

mnu — Menu manager

xyzent — Coordinate entity

This symbol contains the name of the current atom coordinate entity.

Commands affected

bck — Back-calculate NOE intensities

xyl — Atom list manipulation

xyz — Atom manipulation

xzero, yzero — Plot origin

These symbols control the origin of the current plot relative to the origin of the current graphics frame or plot device.

Commands affected

dr — Draw work space and stack

exp — Expanded display

ful — Full display

hcp — Hard copy plot

cp — Contour plot

ip — Intensity plot

sp — Stack plot

np — Null plot

See also

xsize, ysize — Plot size

y0pnt

See x0pnt, y0pnt, x1pnt, y1pnt — Cursor position.

y1pnt

See x0pnt, y0pnt, x1pnt, y1pnt — Cursor position.

yangle

See xangle, yangle, zangle — Euler angles.

ycells

See xcells, ycells — Number of character cells.

ypixel

See xpixel, ypixel — Number of pixels.

ysize

See xsize, ysize — Plot size

ywalk

Seexwalk, ywalk — Walking menu position.

yzero

See xzero, yzero — Plot origin.

zangle

See xangle, yangle, zangle — Euler angles.

C Macro Examples

This appendix gives you some examples of macros. You may use the macros that appear here or write your own. For more detailed information on working with macros, see Chapter 4, *Macros*.

4dv.mac

```
>   ; Macro manufactures a simple 4D (16x16x16x16)
>   ; matrix with synthetic cross peaks located
>   ; at (D1,D2,D3,D4) indices (8,8,8,4),
>   ; (8,8,8,8), and at (8,8,8,13).
;
;
   >     tim z ; start chronograph
   >  ;
   >     bun 0 ; initialize bundle mode
   >  ;
   >  ; initialize matrix values
   >       ty Initializing matrix &matfil
   >       def count 0
   >       def limit 1000
   >       bun 1
   >       for d1vec 1 &vector
```

```
>              lwb
>              ze
>              swb
>       nex
>       bun 0
>       ty
>  ; make cross peaks
>       ty Creating crosspeak at (8,8,8,4)...
>       for loop1 7 9
>              for loop2 7 9
>                     for loop3 7 9
>                            for loop4 3 5
>                                   mpv &loop1 &loop2 &loop3
   &loop4 1000000
>                            nex
>                     nex
>              nex
>       nex
>       mpv 8 8 8 4 2500000
>  ;
>       ty Creating crosspeak at (8,8,8,8)...
>       for loop1 7 9
>              for loop2 7 9
>                     for loop3 7 9
>                            for loop4 7 9
>                                   mpv &loop1 &loop2 &loop3
   &loop4 1000000
>                            nex
>                     nex
>              nex
>       nex
>       mpv 8 8 8 8 5000000
```

```
>  ;
   >       ty Creating crosspeak at (8,8,8,13)...
   >       for loop1 7 9
   >              for loop2 7 9
   >                     for loop3 7 9
   >                            for loop4 12 14
   >                                   mpv &loop1 &loop2 &loop3
   &loop4 1000000
   >                            nex
   >                     nex
   >              nex
   >       nex
   >       mpv 8 8 8 13 2500000
   >  ;
   >       ty -done-
   >       time r time
   >       ty Elapsed time was &time seconds.
   >       ret
   >       end
```

diag.mac

```
>  ; Macro extracts the diagonal plane out of a
>  ; square-cube 3D matrix and stores diagonal
>  ; as a 2D matrix. Caution, macro will overwrite
>  ; matrix defined as 'mx2d'.
>  ;
>  ; Macro employs vector shuttle which effectively
>  ; exploits FELIX bricks.
>  ;
   >       get '3D matrix -->' mx3d
   >       get '2D extract -->' mx2d
```

```
>  ;
>       inq mat &mx3d exist
>       if &exist eq 0 then
>              ty Matrix &mx3d not found!
>              go scram
>       eif
>       inq mat &mx2d exist
>       if &exist eq 0 then
>              ty Matrix &mx2d not found!
>              go scram
>       eif
>  ;
>       mat &mx3d r
>       def nd1 &d1size
>       def nd3 &d3size
>       bld &mx2d 2 &nd1 &nd1 0 y
>       mat &mx2d r
>       def brk &b2size
>       ty brick=&brk
>       eva npass (&nd3/&brk)
>       eva nbuff (&brk+1)
>       ty nbuff=&nbuff
>       def vec 0
>  ;
>       cmx
>  ;
>       dat 0
>       cfg &nd1 &nbuff
>       ty buffers=&buffers
>       for pass 1 &npass
>              ty pass=&pass
```

```
>             mat &mx2d r
>             for xbuff 1 &brk
>                    eva nvec (&vec+&xbuff)
>                    loa 0 &nvec &nvec
>                    stb &xbuff
>                    ty loading vector=&nvec
>             next
>             mat &mx2d w
>             for xbuff 1 &brk
>                    eva nvec (&vec+&xbuff)
>                    lwb &xbuff
>                    sto 0 &nvec
>                    ty storing vector=&nvec
>             next
>             eva vec (&vec+&brk)
>       next
>       cmx
>   scram:
>       end
```

dss.mac

```
>   ; Macro generates a so-called 'doubly-phase-shifted'
>   ; window function. Shows that as expected, anything is
>   ; possible. For cognoscente only...
>   ;
>   ; Initial phase is 90-degrees, final phase is 165-degrees.
>   ;
>       def begin 90
>       def end 165
>       def size 512
>       eva incre (1.0*(&begin-&end)/&size)
```

```
>       ze
>       stb 1
>       def animat 1
>       ty ' '
>       for point 1 &size
>              esc out
>              if &out ne 0 scram
>              eva shift (&begin-(&point-1)*&incre)
>              set 1
>              ty shift=&shift $
>              ss &size &shift
>              dr
>              gv &point value
>              ldb 1
>              pv &point &value
>              stb 1
>       next
>   scram:
>       def animat 0
>       ret
>       end
```

eval_point.mac

```
>   ; Macro evaluates a particular point in each
>   ; vector of a serial 2D data file and stores
>   ; output in a FELIX macro file. Macro presents
>   ; example of using one macro to build another
>   ; macro.
>   ;
>       get 'Input Data File -->' inpdata
>       get 'Output Data File -->' outdata
```

```
>       get 'Data Point to Measure -->' point
>       get 'Number of Exps -->' num_exp
>       opn txt &outdata 0
>       for loop 1 &num_exp
>              re &inpdata
>              gv &point dataval
>              put &dataval
>       next
>       cls
>       ret
>       end
```

lpf_d2.mac

```
>   ; Simple macro to adjust the 1st point in a
>   ; D2(t1)-vector of a transformed 2D matrix
>   ; using LP. Removes missampling artifact in
>   ; virtual acqisition. Caution, original
>   ; matrix is overwritten.
>   ;
      >       get 'Specify matrix -->' mat
      >       inq mat &mat exist
      >       if &exist eq 0 then
      >              ty Matrix &mat not found!!!
      >              go scram
      >       eif
      >   ;
      >       cmx
      >       mat &mat w
      >   ;
      >       for row 1 &d1size
      >              loa &row 0
```

```
>             hft
>             ift
>             lpf
>             ft
>             mul 2
>             red
>             sto &row 0
>             ty col=&row $
>       next
>   scram:
>       ret
>       end
```

madd.mac

```
>   ; Macro sums two matrices in a D1-vector-wise
>   ; fashion. Macro exploits the FELIX matrix format
>   ; to improve efficiency. Caution, macro will
>   ; overwrite 'sum matrix'.
>   ;
      >       get 'Matrix 1 --> ' mat1
      >       get 'Matrix 2 -->' mat2
      >       get 'Sum Matrix --> ' mat3
      >  ;
      >       inq mat &mat1 exist
      >       if &exist eq 0 then
      >             ty Matrix &mat1 not found!
      >             go scram
      >       eif
      >       inq mat &mat2 exist
      >       if &exist eq 0 then
      >             ty Matrix &mat2 not found!
```

```
>              go scram
>       eif
>  ;
>       mat &mat1
>       if &dimen ne 2 not2d
>       def ncol &d1size
>       def nrow &d2size
>       def brick &b2size
>       mat &mat2
>       if &dimen ne 2 not2d
>       if &d1size ne &ncol badsize
>       if &d2size ne &nrow badsize
>       ty Building matrix &mat3...
>       bld &mat3 2 &ncol &nrow 0 y
>       cmx
>       eval ccol (&ncol/2)
>       eva nbuf (&brick+1)
>       cfg &ccol &nbuf
>       mat &mat1 r
>       mat &mat2 r
>       mat &mat3 w
>       ty Adding &mat1 and &mat2 to store in &mat3
>       def count 0
>       loop:
>              if &count ge &nrow done
>              mat &mat1
>              for ibuf 1 &buffers
>                     eva row (&count+&ibuf)
>                     loa 0 &row
>                     stb &ibuf
>                     ty loading= &row $
```

```
>              next
>              mat &mat2
>              for ibuf 1 &buffers
>                     eva row (&count+&ibuf)
>                     loa 0 &row
>                     adb &ibuf
>                     ty adding= &row $
>              next
>              mat &mat3
>              for ibuf 1 &buffers
>                     ldb &ibu
>                     eva row (&count+&ibuf)
>                     sto 0 &row
>                     ty storing= &row $
>              next
>              eva count (&count+&buffers)
>              go loop
>   done:
>   cmx
>   end
>   not2d:
>       ty Matrix must be 2-D.
>       end
>   badsize:
>       ty Matrices must be same size.
>   scram:
>       ret
>       end
```

msub.mac

```
>   ; Macro subtracts two matrices in a D1-vector-wise
```

```
>  ; fashion. Macro exploits the FELIX matrix format
>  ; to improve efficiency. Caution, macro will
>  ; overwrite 'difference matrix'.
>  ;
   >       get 'Matrix 1 -->' mat1
   >       get 'Matrix 2 -->' mat2
   >       get 'Difference Matrix -->' mat3
   >  ;
   >       inq mat &mat1 exist
   >       if &exist eq 0 then
   >              ty Matrix &mat1 not found!
   >              go scram
   >       eif
   >       inq mat &mat2 exist
   >       if &exist eq 0 then
   >              ty Matrix &mat2 not found!
   >              go scram
   >       eif
   >  ;
   >       mat &mat1
   >       if &dimen ne 2 not2d
   >       def ncol &d1size
   >       def nrow &d2size
   >       def brick &b2size
   >       mat &mat2
   >       if &dimen ne 2 not2d
   >       if &d1size ne &ncol badsize
   >       if &d2size ne &nrow badsize
   >       ty Building matrix &mat3...
   >       bld &mat3 2 &ncol &nrow 0 y
   >       cmx
```

```
>       eval ccol (&ncol/2)
>       eval nbuf (&brick+1)
>       cfg &ccol &nbuf
>       mat &mat1 r
>       mat &mat2 r
>       mat &mat3 w
>       ty Subtracting &mat2 from &mat1 to store in &mat3
>       def count 0
>       loop:
>              if &count ge &nrow done
>              mat &mat1
>              for ibuf 1 &nframe
>                     eva row (&count+&ibuf)
>                     loa 0 &row
>                     stb &ibuf
>                     ty Loading= &row $
>              next
>              mat &mat2
>              for ibuf 1 &nframe
>                     eva row (&count+&ibuf)
>                     loa 0 &row
>                     mul -1.0
>                     adb &ibuf
>                     ty Subtract= &row $
>              next
>              mat &mat3
>              for ibuf 1 &nframe
>                     ldb &ibuf
>                     eva row (&count+&ibuf)
>                     sto 0 &row
>                     ty Storing= &row $
```

```
>              next
>              eva count (&count+&nframe)
>              go loop
>       done:
>       cmx
>       end
>   not2d:
>       ty Matrix must be 2-D.
>       end
>   badsize:
>       ty Matrices must be same size.
>   scram:
>       ret
>       end
```

mult.mac

```
>   ; Complex data point in work is multiplied by a
>   ; user-specified value (real number or integer).
>   ;
>       get 'Specify point to vary -->' point
>       get 'Specify point multiplier -->' mult
>       gv &point rvalue
>       exc
>       gv &point ivalue
>       exc
>       eva real (&mult*&rvalue)
>       eva imag (&mult*&ivalue)
>       srv &point &point &real &imag
>       end
```

old2new.mac

```
>  ; Macro converts old serial FELIX data file to
>  ; new data file format.
   >  ;
   >       get 'old data file --> ' fil1
   >       get 'new data file --> ' fil2
   >       def count 0
   >       loop:
   >              re &fil1
   >              if &status ne 0 scram
   >              wn &fil2
   >              eval count (&count+1)
   >              ty count= &count$
   >       go loop
   >   scram:
   >       ty &count records converted.
   >       ret
   >       end
```

psi.mac

```
>  ; Macro conducts the imaginary-part of a D1 hypercomplex
>  ; transform on data acquired in separate real and
>  ; imaginary data files- for antiquaries only!
   >       cl
   >       for row 1 400
   >              loa 0 &row
   >              stb 1
   >              re imagpart.ser
   >              bc 0.05
   >              ss 400 90
```

```
>             zf 2048
>             bft
>             def phase0 89.95
>             def phase1 -21.43
>             ph
>             pol 4
>             exc
>             zr
>             adb 1
>             ldb 1
>             exc
>             sto 0 &row
>             ty row=&row $
>       next
>       ret
>       end
```

psr.mac

```
>   ; Macro conducts the real-part of a D1 hypercomplex
>   ; transform on data acquired in separate real and
>   ; imaginary data files- for antiquaries only!
>       cl
>       for row 1 400
>             re realpart.dat
>             bc 0.05
>             ss 400 90
>             zf 2048
>             bft
>             def phase0 89.9
>             def phase1 -21.43
>             ph
```

```
>             pol 4
>             zi
>             sto 0 &row
>             ty row=&row $
>       next
>       ret
>       end
```

rev_bundle.mac

```
>  ; Reverse vectors of a ND matrix using bundle-mode
>  ; access. Note that original matrix is overwritten!!!
>  ;
>       get 'Specify matrix -->' mat
>       inq mat &mat exist
>       if &exist eq 0 then
>             ty Matrix &mat not found!!!
>             go scram
>       eif
>  ;
>       get 'Specify dimension to reverse -->' revd
>  ;
>       cmx
>       mat &mat w
>       def count 0
>  ;
>       bun 0
>       tim z
>       for vec 1 &vector
>       bun &revd
>       ty There are &vector vectors to process...
>             lwb
```

```
>              rev
>              swb
>              ty Vector=&vector $
>       next
>  ;
>       tim r time
>       ty Elapsed time was &time seconds.
>  ;
>   scram:
>       ret
>       end
```

zap.mac

```
>   ; Original macro written by Dr. Walter Massefski.
>   ; Macro conducts a 'circular-shift' on data to place
>   ; 'solvent' peak at exact center, then applies the
>   ; convolution-based (cnv) solvent suppression and back
>   ; shifts the data.
>   ;
      >       ty Specify window type
      >       get '(Sinebell=0, Gaussian=1) -->' wintyp
      >       if &wintyp ne 0 or &wintyp ne 1 then
      >              ty Invalid window-type selection.
      >              go scram
      >       eif
      >       get 'Window size -->' zaparg
      >       eva midpt (&datsiz/2+1)
      >       cur 1 1 2 1
      >       if &x0pnt lt &first or &x0pnt gt &last scram
      >       eva getpt (&midpt-&x0pnt)
      >       csl &getpt
```

```
>       ift
>       cnv &wintyp &zaparg
>       ft
>       csr &getpt
>       dr
>   scram:
>       cur 0 1 1 1
>       ret
>       end
```

Index

K

L

M

O

P

R

S

T

U