Python Programming in TopSpin

User Manual

Version 003

Innovation with Integrity

NMR

Document Number: 10000058957

P/N: H166420

Contents

Contents

1 Introduction

Here we describe how to extend TopSpin by writing Python modules or scripts.

Python is the most popular programming language in the scientific community. The Python Package Index (PyPI), the official repository for third-party Python software, contains over 290,000 packages with a wide range of functionality.

TopSpin includes a Jython (Python 2.7 implementation) interpreter as a standard component for scripting and application development for 20 years. The Jython scripts are running in the same Java virtual machine as the TopSpin GUI is executed. This opens many possibilities, however, also brings several drawbacks. The built-in Jython interpreter stays at Python 2.7 language standard and does not support any modern packages like NumPy.

The requirement to support the whole spectrum of Python 3 packages was a reason for developing the Python API in Topspin. This new API allows to access Topspin from any Python 3.9+ environment.

This document describes both Python interfaces. The interfaces used for the data access are the same for Python and Jython. The Python or Jython specific details are described in the dedicated parts of this document.

Introduction

2 TopSpin Python API

In this chapter we describe the Python interface for accessing of Bruker data sets. This API is identical for both external Python 3 interpreter and Topspin internal Jython. However, there is still a small difference:

Python 3 scripts using this API require following imports

```
from bruker.api.topspin import Topspin
from bruker.data.nmr import *
```

Jython scripts do not require any imports.

The API is object oriented, there are following basic classes.

- Topspin – the base of the implementation

```
# Get the instance of a DataSetProvider
def getDataProvider(self):...

# Get the instance of Display controler
def getDisplay(self):...

#   Execute command in TopSpin
def executeCommand(self, command, dataSet=None, args=None, wait = False):...

# Get the version of TopSpin
def getVersion(self):...

# Get the installation path of TopSpin
def getInstallationDirectory(self):...

# Show an message dialog in TopSpin
def showMessage(self, message_text):...

# Show an error dialog in TopSpin
def showError(self, message_text):...
```

- DataProvider – implements handling of data sets

```
#
# Find NMR data sets with specified attributes
#
def find(self, directory, name=None, expno=None, procno=None, user=None, title=None,
         pulprog=None, dim=-1, type=FIND_ANY, spectyp=None):...
#
# A safe method to create the NMRDataSet object.
# returns None if the data sets does not exists
#
def getNMRData(self, path):...

#
#   Get the identifier of the data set visible on the screen
#   (current data set)
def getCurrentDatasetIdentifier(self):...

#
#   Get the data set visible on the screen as a NMRData object
#
def getCurrentDataset(self):...
```

- Display - controls the TopSpin display

```
#
# Show a NMRDataSet in Topspin
#
def show(self,dataset, newWindow = True):...

def arrangeWindowsHorizontal(self):...

def arrangeWindowsVertical(self):...

def arrangeWindowsMatrix(self):...

def closeAllWindows(self):...
```

- NMRDataSet – includes methods for data access. One NMRDataSet object corresponds to a one Bruker NMR data set consisting of one EXPNO and one PROCNO. Current implementation allows to access 1D and 2D data sets. Reading and writing of parameters as well as execution on commands are also included.

```
# Get the data set identifier
def getIdentifier(self):...

# Get the dimension of the data set
def getDimension(self):...

# Get the dimension of processed data
def getProcDim(self):...

# Get the basic information about the data set
def getInfo(self):...

# Get a file from the processed data folder
def getProcFile(self, filename):...

# Get a file from the raw data folder
def getRawFile(self, filename):...

# Get a structure file from the data set
def getStructureFile(self):...

# Get one parameter
def getPar(self, parName):...

# Write one parameter
def setPar(self, parName, parValue):...

# Get set of parameters at once, return value is a dictionary
def getParameters(self, parNames):...
```

```
# Convert the physical coordinates (ppm, s) to data set index
def getIndexFromPhysical(self,physical, direction = 0, component = PROCDATA):...

#   Get vector of data points in a component (spectrum, FID,...). The default component is the processed spectrum
def getSpecDataPoints(self, component = PROCDATA, row = -1, column = -1, physRange = None, ir = None,precision = 'd'):

#   Get vector of RAW data points (FID/SER file)
def getRawDataPoints(self, row=-1, ir=None, precision = 'd'):...

# Get the 1D integration regions or None, if the region file does not exists
def getIntegrationRegions(self):...

#  Get the peakList
def getPeakList(self):...

# Execute a command on the data set, wait until finished by default
def launch(self, command, arg=None, wait = True):...
```

2.1 TopSpin Python API Step by Step

This section demonstrates the basic features of the TopSpin API on practical examples.

First step is always the initialization of the interface. Topspin object is created.

```
top = Topspin()
dp = top.getDataProvider()

logging.info("Topspin installation :"+top.getInstallationDirectory())
logging.info("Topspin version :"+top.getVersion())
logging.info("Current dataset :"+str(dp.getCurrentDatasetIdentifier()))

top.getDisplay().closeAllWindows()
```

The first example gets the current data set from TopSpin, creates the 2nd PROCNO and displays it in TopSpin.

```
#
# Create an instance of Topspin interface
#
top = Topspin()
dp = top.getDataProvider()

print("Topspin installation :"+top.getInstallationDirectory())
print("Topspin version :"+top.getVersion())
print("Current dataset :"+str(dp.getCurrentDatasetIdentifier()))

top.getDisplay().closeAllWindows()
```

The new PROCNO is reprocessed and displayed in the multiple display together with the first one.

```
proton2 = dp.getCurrentDataset()
print(proton2.getIdentifier())
proton2.setPar("LB",2.0)
proton2.launch("efp")
proton2.launch(".md")
proton2.launch("rep 1")
```

In the next example, the data sets are searched using find.

```
# search the example data sets
examples = dp.find(top.getInstallationDirectory() + '/examdata',name = "exam_CMCse_1", dim = dp.FIND_1D)

# get the instance of the first one and process it
proton = dp.getNMRData(examples[0])
proton.launch("efp")
```

Display the data sets in TopSpin.

```
# show the data sets on the TopSpin display
top.getDisplay().show(proton)
top.getDisplay().show(examples[1])
top.getDisplay().arrangeWindowsVertical()
```

There are several methods for reading of data points.

Read the whole data set at once.

```
res = proton.getSpecDataPoints()
print(res)
```

The data points are always returned as a dictionary containing the data point array, corresponding physical coordinates and indexes of the region being read.

```
{
 'indexRanges': [{'firstPoint': 0, 'numberOfPoints': 65536}],
 'physicalRanges': [{'start': 14.686180114746094, 'end': -5.339362865836517}],
  'dataPoints': array('d', [781.03125, 786.4140625, 800.8125, 793.84375, 776.4140625,.......
}
```

Imaginary part of the spectra, or the RAW data can be read in a similar way. Please note, that the required part of the spectra may be specified either using the physical coordinates, or indexes. In both cases, an array of range objects is used even for 1D spectra.

```
#
# read the imaginary part between 4 and 5 ppm
#
print("Imaginary part")
print(proton.getSpecDataPoints(component=PROCDATA_IMAG, physRange=[PhysicalRange(5, 4)]))

#
# read the first 32 points of the FID
#
print("FID")
res = proton.getRawDataPoints(ir=[IndexRange(0, 32)])
print(res)
```

The same interface may be also used to read 2D data. Reading of rows or columns is simple.

```
nColumns = int(hsqc.getPar("2s SI"))

for i in range(0, nColumns):
    column = hsqc.getSpecDataPoints(column=i)
```

The following example is more complex. A submatrix of “mat_w” data points around each peak in a 2D peak list is read. The peak positions are stored in physical coordinates (ppm). The method “getIndexFromPhysical()” is used to get the corresponding indexes. The indexes are zero based (in the range from 0..SI-1, where SI is the size of the data matrix in the corresponding dimension). The 2D data points are returned as a 1D array – all rows in the submatrix are joined into one array.

```
#
#   read the peak list
#
peakList = hsqc.getPeakList()

#
# Read the area around the peak
#
m2 = mat_w / 2 + 1
for peak in peakList:
    print(peak['position'],peak['intensity'])
    pos = peak['position']
    i1 = hsqc.getIndexFromPhysical(pos[0], direction=0)
    i2 = hsqc.getIndexFromPhysical(pos[1], direction=1)

    if i1 > m2 and i1 + m2 < nColumns and i2 > m2 and i2 + m2 < nRows:
        dp = hsqc.getSpecDataPoints(ir=[IndexRange(i1 - mat_w / 2, mat_w),
                                        IndexRange(i2 - mat_w / 2, mat_w)])['dataPoints']
```

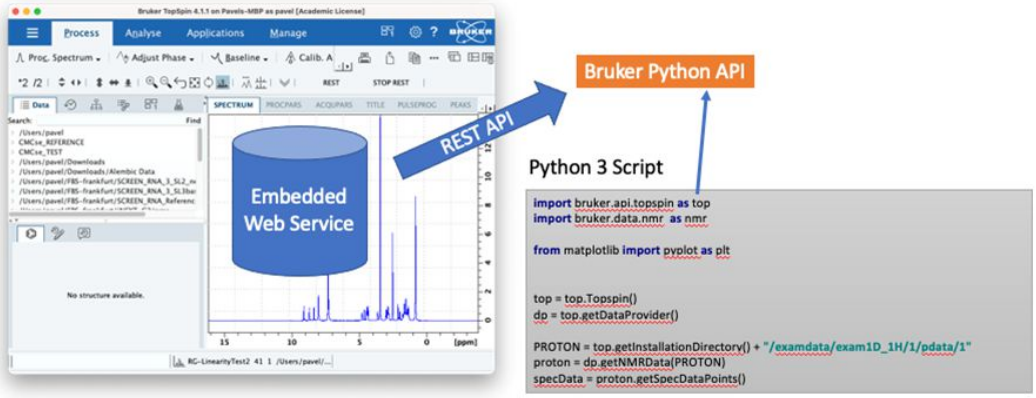
The next example requires Python 3: a portion of the 2D spectrum is read. The packages numpy and matplotlib are used to convert the data into a 2D matrix and display it.

```
HSQC   = top.getInstallationDirectory() + "/examdata/exam_CMCse_1/3/pdata/1/"

hsqc = top.getDataProvider().getNMRData(HSQC)

spectrum = hsqc.getSpecDataPoints(physRange = [PhysicalRange(2.4, 0.6), PhysicalRange(37,18)])
si1 = int(spectrum['indexRanges'][0]['numberOfPoints'])
si2 = int(spectrum['indexRanges'][1]['numberOfPoints'])

arr = numpy.array(spectrum['dataPoints'])
arr = arr.reshape(si2,si1)

# read th contour levels from Topspin
contourLevels = re.split("[|]", hsqc.getPar('LEVELSPAR LEVELS'))

plt.contour(arr, contourLevels)
plt.show()
```

3 Python 3

Python 3 interpreter is a standalone application. Therefore, the simplest possibility to create a bridge between Python application and TopSpin is to use a network connection. Topspin 4.1.2 and later include an embedded Web Server, which implements a REST API.

Bruker implemented Python 3 packages, which connect to this REST API endpoint and implement the API described in this document.



The embedded Web service is not activated on TopSpin start. You need to start it manually, the corresponding command **start_rest_interface** requires spectrometer administration privileges and requires a valid license (see Installation). The REST API is limited to the local network interface (localhost) and uses the port 3080. Alternative port number may be specified

start_rest_interface -p XXXX

In such case, you need to specify the port number in the Python interface.

```
#
# Create an instance of Topspin interface
#
top = Topspin(url = "http://localhost:XXXX/v1")
```

The Web service may be stopped at any time – execute command **stop_rest_interface**.

3.1 Installation

The TopSpin Windows installation includes an up-to-date python environment. The Bruker API packages and example files are already installed

<Topspin>/python

<Topspin>/python/examples

For other operating systems:

1. Install the current Python 3 release
2. Install the Bruker Python packages

   ```
   python3 -m pip install bruker_topspin_rest_api-XXXXX-py3-none-
   any.whl
   python3 -m pip install bruker_nmr_api_XXXXX-py3-none-any.whl
   ```

   If an older version of the package was installed, you may enforce the installation of the new one

   ```
   python3 -m pip install --force-reinstall <package>
   ```

3. Ensure, that additional python packages used in example scripts were installed

   ```
   python3 -m pip install numpy
   python3 -m pip install matplotlib
   ```

A common requirement is to install the Topspin REST API license. The corresponding license key is SHTSA01.

4 Jython

Jython is a Python dialect based on the Python 2.7 language standard. It was embedded in TopSpin to provide the user with a sophisticated tool allowing to add own functions to TopSpin.

Jython programs (modules, scripts) written for TopSpin are capable of

- executing TopSpin commands
- opening dialog windows for user input or to print messages
- opening NMR data sets for further processing
- fetching and setting NMR parameters
- reading TopSpin NMR data for further manipulation by the Python program
- displaying arrays of data calculated in a Python program
- using the Java swing classes which provide a rich set of functions with virtually no limit in user interface and graphics programming

4.1 Quick Start

Enter the TopSpin command **edpy** in the command line. Result: A window is opened showing the currently available Jython programs including the Bruker example programs. Please use those as a reference for your own programs. From the **edpy** window you can create a new Jython program and start it up.

Editing a new or an existing Jython program without using the **edpy** window: Enter the command **edpy** <program name>.

Executing an existing Python program without using the **edpy** window: Enter the command **xpy** <program name>, or just <program name>. The latter method only works if <program name> is not already the name of a TopSpin internal command or an AU program.

Making an external Jython program available in TopSpin: Enter the command **edpy** and choose *Import...* from the *File* menu.

4.2 The Location of Jython Programs and Function Libraries

The location of Jython programs delivered with TopSpin is

<TopSpin installation directory>/exp/stan/nmr/py.

TopSpins provides the following default location for user Jython programs:

<TopSpin installation directory>/exp/stan/nmr/py/user.

Both directories are available for selection in the upper right corner of the **edpy** window. In addition, a user may place his own python programs anywhere he wants. For the programs to become accessible via **edpy** the respective directories must be entered into a list provided by TopSpin: Open the list by clicking **Options | Manage Source Directories** in the menu bar of the **edpy** window. Navigate to the entry **Jython programs** and add your desired directories.

4.2.1 Using Functions Stored in Different Files

If your Python program (stored in the file *file1.py*) calls a function *func* stored in the different file *file2.py*, then the Python statement *from file2 import func* (or *from file2 import **) is required. By default this will only work if file2.py is located in one of the directories

<TopSpin installation directory>/exp/stan/nmr/py/user/

<TopSpin installation directory>/classes/lib/topspin_py/py/pycmd/

because this is the default „search path“ of the Python interpreter. If *file2.py* should be located in a different directory, then this directory must be made known to the interpreter as follows: Add the following line to the file *<userhome>/.topspin-<pcname>/prop/globals.prop* :

PYTHON_MODPATH=<directory> (e.g. PYTHON_MODPATH=c:/Users/guest)

It is also possible to specify several directories, e.g.

PYTHON_MODPATH=c:/Users/guest;c:/Users/guest/my-python-programs

Note the system dependent path separator “;“ for Windows. For Linux “:“ must be used.

If you don't want to employ PYTHON_MODPATH, you can alternatively add the following statement at the beginning of the Python program (as an example):

sys.path.append("C:/Users/guest/MyPyLib“)

4.3 Jython Functions For TopSpin

Please note: All Jython functions not being part of Jython itself, but added by Bruker interact with TopSpin are defined in the file

<TopSpin installation directory>/classes/lib/topspin_py/py/pycmd/TopCmds.py

In this file you may find functions that are not described explicitly below. The table below lists the most widely used and important functions. The table contains examples for each function. Proceed as follows to execute them:

Step 1: Enter the TopSpin command **help python (**or **python?)** to open this manual.
Step 2: Enter the command **edpy pytest**. An empty text editor will be opened for this filename. Select an example in the manual by marking the example text and copying it to the clipboard (e.g. with CTRL C, or with Edit | Copy of the Acrobat Reader menubar). Click into the editor and paste the example text there with CTRL V. Click on the **Execute** button of the editor. That's all.

Many of these functions operate on the current dataset (e.g. fetching/storing parameters, processing data). In a Jython program, the current dataset is defined as follows:

When the script is started, the current dataset is the dataset of the currently active TopSpin data window. If no such window is open, the current dataset is undefined. In order to define the current dataset inside the Jython program, use the appropriate dataset functions described the following table, e.g. RE().

Jython Function	Description
Output and Confirm Dialogs	
MSG(message = "", title=None)	A modal dialog with a *Close* button: *message* = the message to be shown *title* = Optional. The title of the dialog window

Jython Function	Description
Output and Confirm Dialogs	
	The Jython program will not continue until the *Close* button is clicked.
ERRMSG(message = "", title=None, details=None, modal=0) *Example 1:* `ERRMSG("Test Message Non Modal")` *Example 2:* `ERRMSG("Test Message Modal", modal=1)`	A modal or non-modal dialog with a *Close* button and a *Details* button: *message* = the message to be shown *title* = Optional. The title of the dialog window *details* = Optional. A message that is displayed when clicking of the *Details* button of the dialog. *modal* = Optional. The Jython program will not continue until the *Close* button is clicked if modal = 1. Otherwise it will continue, while the dialog stays on the screen.
value = CONFIRM(title=None, message="") *Example:* `if CONFIRM("Delete", "Delete this file?")== 0:` `EXIT()`	A confirm dialog with an *OK* button and a *Cancel* button: *title* = The title of the dialog window *message* = the message to be shown *value* = 1 if OK clicked, otherwise 0
value = SELECT(title=None, message="", buttons=["OK_M", "CANCEL_M"], mnemonics=None) *Example 1:* value = SELECT("Delete", "Delete these files?", ["Delete Selected", "Delete All", "Close"]) if value == 2 or value < 0: EXIT() *Example 2:* `value = SELECT("Delete", "Delete these files?",\` `["Delete Selected", "Delete All", "Close"], ['s', 'a', 'c'])`	A confirm dialog with an arbitrary number of buttons: *title* = the title of the dialog window *message* = the message to be shown *buttons* = a list of button labels *mnemonic*=Optional. A list of characters, shortcuts for the buttons. *value* = the number of the pressed button (0, 1, ...), or negative if ESCAPE pressed, or the dialog window's close icon in the upper right corner.
VIEWTEXT(title="", header="", text="", modal=1) *Example:* `mytext="""` `This is` `my multi-line`	A modal or non-modal viewer for big text, e.g. as read in from a file (for short text better use MSG()): *title* = the title of the text window *header* = a header text near the top of the window *text* = the text *modal* = Option. 1=modal window (default), 0 = non modal.

Jython Function	Description
Output and Confirm Dialogs	
example text """ VIEWTEXT("MyTitle", "MyHeader", mytext)	
SHOW_STATUS(message="") *Example:* SHOW_STATUS("Script XYZ In Progress.")	Displays a message in TopSpin's status line.
Input Dialogs	
result = INPUT_DIALOG(title=None, header=None, items=None, values=None, comments=None, types=None, buttons=None, shortcuts=None, columns=30) *Example 1:* result = INPUT_DIALOG("MyTitle", "This is an example\nNumber 1.", ["Solvent = ", "Nucleus = "], ["CDCl3", "1H"], ["",""], ["1", "1"]) if result <> None: MSG(result[0] + "\n" + result[1]) *Example 2:* result = INPUT_DIALOG("MyTitle", "This is an example\nNumber 1.", ["Solvent = ", "Nucleus = "], ["CDCl3", "1H"], ["",""], ["1", "1"], ["Accept","Close"], ['a','c'], 10) if result <> None: MSG(result[0] + "\n" + result[1])	A general input dialog window with an arbitrary number of text input lines, and 2 buttons to press: *title* = the dialog title header = an additional multiline header, lines to be separated by “\n” *items* = list of label names for text input fields (or None) *values* = list of initial values for the text input fields (or None) *comments* = list of comments to be appended to the text input fields (or None), may have several lines to be separated by “\n” *types* = the list of text input field sizes (1=single line, >1 multiple line field) *buttons* = list of button labels. If None, an OK and Cancel button are shown. *shortcuts* = list of button shortcuts. *columns* = number of text columns (width of input fields), default = 30. *result* = the list of values in the text fields, if the user pressed OK, or None if the user pressed ESC, or Cancel, or the Window's close icon.
result = DATASET_DIALOG(title=None, values=None) *Example 1:* result = DATASET_DIALOG("MyTitle", CURDATA()) if result <> None: MSG(result[0] + "\n" + result[1] + "\n" + result[2] + "\n" + result[3])	A dialog window for a user to enter a TopSpin dataset with the following buttons: OK, FIND, BROWSE, CANCEL. *title* = the dialog title *values* = Optional. Initial values of the text input fields *result* = the list of dataset specifiers [name, expno, procno, directory] entered or selected by the user.

Jython Function	Description
Output and Confirm Dialogs	
result = FIND_DIALOG() *Example 1:* `result = FIND_DIALOG()` `datapaths = ""` `if result == None:` `EXIT()` `for datapath in result:` `datapaths += datapath + "\n"` `MSG(datapaths)`	A „find“ dialog window for a user search for TopSpin data setes *result* = The list of full dataset paths. The list consists of those data sets selected by the user from the search result. None is returned if nothing found or the user cancelled the dialog.
System Functions	
CMDTHREAD *Example:* `id = prsc.util.DataChecks.getDataId( CMDTHREAD)` `MSG(id)`	Not a function, but a variable of type CmdThread representing the thread in which a Jython script executes. Provides access to the thread state. The example prints the dataset id of the thread's current dataset
result = EXEC_PYSCRIPT(pyscript, arg) *Example:* `script = """` `f = open("c:/foo.txt", 'w')` `f.write("test line\\n")` `f.close()` `"""` `EXEC_PYSCRIPT(script)`	Executes a Jython script in its own thread (i.e. in background): *pyscript* = the script text *arg* = an optional argument (any Object) *result* = the thread of type CmdThread
result = EXEC_PYFILE(pyfile, arg) *Example:* `EXEC_PYSCRIPT(script)`	Executes a Jython program in its own thread (i.e. in background): *pyfile* = the absolute path of the file containing the script text, or the name of the Jython program in the data base *arg* = an optional argument (any Object) *result* = the thread of type CmdThread
EXIT() *Example:* `if CONFIRM("Delete", "Delete this file?")== 0:` `EXIT()`	Terminates the current Jython script. Not required at the end of the Jython program.

Jython Function	Description
Output and Confirm Dialogs	
host = GETHOSTNAME() *Example:* `host = GETHOSTNAME()` `MSG("result="+host)`	Returns the host name of the PC where TopSpin is currently running.
sys.argv *Example:* *Prints the number of arguments, and the arguments when starting the script.* `args = "number of args = " + str(len(sys.argv)) + "\n"` `args += "arglist:\n"` `for arg in sys.argv:` `args += arg + "\n"` `MSG(args)`	The arguments passed to a Jython script. Assume you start a Jython script *myscript* by typing *xpy myscript a b c* or just *myscript a b c* into TopSpin's command line. The 3 arguments *a b c* (separated by space characters) can be retrieved in the script by using *sys.argv* This is an arry of Strings: *sys.argv[0]* = the complete path of the script *sys.argv[1]* = the first argument (if present) *sys.argv[2]* = the second argument (if present) Etc. *len(sys.argv)* = the number of actual arguments + 1.
SLEEP(seconds) *Example:* `SLEEP(2)`	Pauses execution of the current Jython program by the specified number of seconds. Shortcut for Jython's time.sleep() function.
ct = XCMD(cmd, wait = WAIT_TILL_DONE, arg = None) *Example 1:* *Prints result as a String: ft return 0 if succeeded, -1 if failed.* `ct = XCMD("ft")` `MSG("result="+str(ct.getResult())` *Example 2:* *Terminates script if Cancle pressed in SI Dialog.* `if XCMD("SI").getResult() == None:` `EXIT()` `FT()`	Executes an arbitrary TopSpin command: *cmd* = Command name *wait* = Optional. `WAIT_TILL_DO`NE or `NO_WAIT_TILL_DONE`. *arg* = optional argument for the command (depends on the command whether such an argument is evaluated) *ct* = an Object of type CmdThread, allowing access to the `result with ct.getResult()`. The result type depends on the command. The result is undefined for *wait* = `NO_WAIT_TILL_DONE` If *cmd* is a processing command (e.g. "`ft`"), `XCMD("<command>").getResult()` is -1 when execution failed. If *cmd* is a parameter name (e.g. "`SI`"), then usually a dialog window is opened to enter the parameter. `XCMD("<parameter<").getResult()` is None

Jython Function	Description
Output and Confirm Dialogs	
	when the user did not press the OK button of the dialog. Otherwise the result is a String with currently undefined value. This feature allows one to terminate the Jython script when Cancel or ESC etc. pressed. **Please Note:** For most TopSpin processing and acquisition commands there is a Jython equivalent, e.g. `FT(), APK(), EFP(), ...` `XCMD` should only be used when no suitable Jython function is available.
Defining The Current NMR Dataset	
RE(dataset = None, show = "y") *Example 1:* `RE(["exam1d_13C", "2", "1", "c:/mydatadir"])` *Example 1a:* `RE(["exam1d_13C", "2", "1", "c:/bruker/topspin"], "guest")` *Example 2:* `RE(["exam1d_13C", "2", "1", "c:/mydatdir"], "n")` *Example 3:* *Reads 7 data sets (expno = 1...7) into a new window.* `for i in range(7):` `data = ["exam1d_13C", str(i+1), "1", "c:/x y"]` `NEWWIN()` `RE(data)`	Makes the specified data set the current dataset: *dataset* = list of TopSpin dataset specifiers [name, expno, procno, directory] *show* = Optional: y = dataset is displayed (default), n = dataset is not displayed. Examle 1a) can be used if the dataset is stored in a traditional type directory of the form */bruker/topspin/data/guest/nmr.*
RE_PATH(dataset = None, show = "y") *Example:* `RE("c:/bruker/topspin/data/exam1d_13C/2/pdata/1")`	Makes the specified data set the current dataset: *dataset* = the complete disk path name of a TopSpin dataset *show* = Optional: y = dataset is displayed (default), n = dataset is not displayed.
value = CURDATA(cmdthread = None)	Returns the current dataset of a Jython script:

Jython Function	Description
Output and Confirm Dialogs	
Example: `curdat = CURDATA()` `MSG("name="+curdat[0] + ", procno="+curdat[2])`	*cmdthread* = Optional: The thread in which the script is running. Default is the current thread. *value* = the list of dataset specifiers [name, expno, procno, directory]. *Value* = None if no curdata defined.
RE_IEXPNO(dataset = None, show = "y") *Example:* `MSG("expno-before ="+CURDATA()[1])` `RE_IEXPNO()` `MSG("expno-after ="+CURDATA()[1])`	Increments the EXPNO of the specified dataset and makes it the current one: *dataset* = Optional: list of TopSpin dataset specifiers [name, expno, procno, directory] Default is the current dataset. *show* = Optional: y = dataset is displayed (default), n = dataset is not displayed. Does not create a new EXPNO if the new one doesn't exist!
RE_IPROCNO(dataset = None, show = "y") *Example:* `MSG("procno-before ="+CURDATA()[2])` `RE_IPROCNO()` `MSG("procno-after ="+CURDATA()[2])`	Increments the PROCNO of the specified dataset and makes it the current one: *dataset* = Optional: list of TopSpin dataset specifiers [name, expno, procno, directory] Default is the current dataset. *show* = Optional: y = dataset is displayed (default), n = dataset is not displayed. Does not create a new EXPNO if the new one doesn't exist!
RSER(fidnum = None, expno = None, show = "y") *Example:* `RSER("27", "777")` `FT()`	Reads the specified fid number from the current data set (must be a 2D/3D/... data set with raw data), stores it under the specified EXPNO, and makes this the current data set. *fidnum* = the fid number as a String (1, 2, ...) *expno* = the destination expno as a String *show* = Optional: y = expno is displayed (default), n = expno is not displayed.
RSR(row = None, procno = None, show = "y") *Example:* `RSR("27", "999")`	Reads the specified row from the current data set (must be 2D data set with processed data), stores it under the specified PROCNO, and makes this the current data set. *row* = the row number as a String *procno* = the destination procno as a String *show* = Optional: y = procno is displayed (default), n = expno is not displayed.
RSC(col = None, procno = None, show = "y") *Example:*	Reads the specified column from the current data set (must be 2D data set with processed data), stores it under the specified PROCNO, and makes this the current data set.

Jython Function	Description
Output and Confirm Dialogs	
`RSC("27", "999")`	*col* = the column number as a String *procno* = the destination procno as a String *show* = Optional: y = procno is displayed (default), n = expno is not displayed.
Saving The Current NMR Dataset	
WR(dataset = None, override = "y") *Example 1:* `WR(["exam1d_13C", "2", "1", "c:/"])` *Example 2:* `WR(["exam1d_13C", "2", "1", "c:/"], "n")`	Writes (copies) the current dataset to the specified destination data set: *dataset* = list of TopSpin dataset specifiers [name, expno, procno, directory] for the destination, must be different from the source *override* = Optional: y = new destination dataset is overridden silently if existing (default), n = a confirm dialog is displayed. The destination does NOT become the current data set. Use RE to make it the current if needed.
WR_PATH(dataset = None, override = "y") *Example 1:* `WR("c:/mydata/exam1d_13C/2/pdata/1")` *Example 2:* `WR("c:/mydata/exam1d_13C/2/pdata/1", "n")`	Saves (copies) the current dataset to the specified destination data set: *dataset* = the complete disk path name of a TopSpin dataset, must be different from the source *override* = Optional: y = new destination dataset is overridden silently if existing (default), n = a confirm dialog is displayed. The destination does NOT become the current data set. Use RE to make it the current if needed.
Creating A New (Empty) NMR Dataset	
NEWDATASET(dataset, expdir = None, exp = "PROTON") *Example :* `dataset = ["mydataset1", "1", "1", "c:/nmrdata"]` `NEWDATASET(dataset, None, "PROTON")` `RE(dataset)`	Create a new NMR data set on disk using the parameters below. An possibly existing data set with the same specifications will be removed silently! dataset = [name, expno, procno, dir] expdir = initialize dataset with parameter files from this parameter directory (if null: uses Bruker standard experiment dir). exp = init. dataset with parameter files from this experiment contained in expdir (if None: uses "standard1D")
Fetching And Storing NMR Parameters	
value = GETPAR2(name) **Examples for name:** **'1 SI' or 'SI' (fetch from file proc)**	Gets the value of a TopSpin parameter from the *current* data set: name = the parameter name **name has the form 'axis parname index'**

Jython Function	Description
Output and Confirm Dialogs	
's2 SI' (fetch from file proc2s, requires data dimension >= 2) **'2 TD' (fetch from file acqu2, requires data dimension >= 2)** **'4 TD' (fetch from file acqu4, requires data dimension >= 4)** **'1 CPDPRG 3' or 'CPDPRG 3' (fetch from file acqu)** **'1 P 4' or 'P 4'(fetch from file acqu)** **'1 PULPROG' or 'PULPROG' (fetch from file acqu)** **'1 SW_p' or 'SW_p' (fetch from file proc)** **'1 SWH' or 'SWH' (fetch from file acqu)** **'1 HZpPT' or 'HzpPT' (derived parameter)** **'1 USERP1' or 'USERP1' (fetch from file proc)**	**where axis = 1,2,3,4,... (up to the data dimension). The parameter is fetched from the file proc, proc2, proc3, proc4, ..., respectively, if the parameter is a processing parameter, or from the file acqu, acqu2, acqu3, acqu4, ..., respectively, if the parameter is an acquisition parameter. Specifying no axis defaults to 1. If the axis begins with an 's', the parameter is fetched from the respective status parameter file: procs, proc2s, ... acqus, acqu2s, ...** **index is only required for array type parameters.** **Please note:** **For various reasons the axis specifications define the parameter file number pnum (1 [which is omitted from the file name], 2, 3, 4,...), but not the „frequency axis number fnum“ F1, F2, F3, … .** **This is different from the function GETPAR below.** **The respective relation is: fnum = dim – pnum + 1,** **where dim is the dimension of the data set from where the parameter is to be fetched.**
value = GETPAR(name, axis = 0) *Example 1:* *computes 2*SI and prints result:* `SIstring = GETPAR("SI")` `twoTimesSI = 2*int(SIstring)` `MSG("result="+str(twoTimesSI))` *Example 2:* *computes 1/(2*D1) and prints result:* `d1String = GETPAR("D 1")` `j = 1.0 / 2*float(d1String)` `MSG("result="+str(j))` *Example 3:* computes 1/(2*D1) and prints result: `d1String = GETPAR("D 1")` `j = 1.0 / 2*float(d1String)`	Gets the value of a TopSpin parameter from the *current* data set: name = the parameter name axis = Optional. 0 (acquisition axis = default) 1, 2, 3, ... (F1, F2, F3, ... axis for nD data) value = parameter value as a String **Note:** In order to perform calculations with value, value must be converted from String to its proper type, e.g. using the Jython functions int() or float(). **Alternate method of invoking GETPAR():** Instead of specifying the axis as the second argument, the axis can be encoded into the parameter name:

Jython Function	Description
Output and Confirm Dialogs	
`MSG("result="+str(j))` *Example 4:* *computes 2*SI (from F1 direction of nD data) and prints result:* `SIstring = GETPAR("SI", 1)` `twoTimesSI = 2*int(SIstring)` `MSG("result="+str(twoTimesSI))`	GETPAR("1 SI") corresponds to GETPAR("SI", 1) This is similar to the TopSpin command line syntax. It also allows one to get access to status parameters: GETPAR("2s SI") corresponds to GETPARSTAT("SI", 2)
value = GETPARSTAT(name, axis = 0)	Gets the value of a TopSpin *status* parameter (i.e. as stored in the status parameters files acqus/ procs). Usage: See GETPAR above.
result = GETACQUDIM() *Example:* `MSG(str(GETACQUDIM()))`	Gets the dimension of the acquisition data of the current data set. *result* = the dimension as an integer 1, 2, ..., or -1 in case of an error (no acquisition data present)
result = GETPROCDIM() *Example:* `MSG(str(GETPROCDIM()))`	Gets the dimension of the processed data of the current data set. *result* = the dimension as an integer 1, 2, ... or -1 in case of an error (no processed data present)
PUTPAR(name, value) *Example 1:* *Sets the size for the acquisition direction* `PUTPAR("SI", "2048")` *Example 2:* *Sets the status parameter for the acquisition direction* `PUTPAR("status SI", "2048")` *Example 3:* *Sets the size for the F1 axis* `PUTPAR("1 SI", "2048")` *Example 4:* *Sets the status parameter for the F1 direction* `PUTPAR("1s SI", "2048")`	Sets the value of a TopSpin parameter for the *current* data set: *name* = the parameter name, possibly including axis (for data dimension > 1) and status, similar to the TopSpin command line syntax *value* = the parameter's value as a String

Jython Function	Description
Output and Confirm Dialogs	
Example 5: *Sets the array parameter P1* `PUTPAR("P 1", "0.5")` *Example 5:* *Sets the pseudo-array parameter CPDPRG5* `PUTPAR("CPDPRG5", "mycpd")`	
TopSpin Internal Windows	
result = NEWWIN(width = -1, height = -1) *Example 1:* `NEWWIN()` `RE(["exam1d_13C", "2", "1", "c:/bruker/topspin/data"])` *Example 2:* `NEWWIN(300, 400)`	Opens an empty new internal TopSpin window. *width* = Optiona: window width (pixels) *height* = Optiona: window height (pixels) *result* = a JPanel object, a container for e.g. NMR data or other components. RE() type functions will load their datasets into the last opened window, regardless what other windows are open.
result = GETWINID(userPanel = None) *Example 1:* *Prints the id of a new window* `MSG(GETWINID(NEWWIN()))` *Example 2:* *Prints the id of the default window.* `MSG(GETWINID())`	Gets the unique ID String of an internal TopSpin window. *userPanel* = Optional: The user panel of the window, as e.g. returned by NEWWIN(). If not specified, the id of the window is returned which was the current one at the time the script was started (default window). *result* = the unique ID String, or None if no window existing for this script.
CLOSEWIN(dataset = None) *Example 1:* `CLOSEWIN(["exam1d_13C", "2", "1", "c:/bruker/topspin/data"])` *Example 2:* *Closes the window containg the current dataset.* `CLOSEWIN(CURDATA())`	Closes the internal TopSpin window containing the specified dataset. Has no effect if no such window exists. *dataset* = a list of dataset specifiers [name, expno, procno, dir] RE() type functions will load their datasets into the last opened window.
ARRANGE(mode = "$v") *Example:*	Arranges all open internal TopSpin windows.

Jython Function	Description
Output and Confirm Dialogs	
`ARRANGE()` *Example 2:* `ARRANGE("$h")`	*mode* = $v (default, arrange vertically = in stack), $h (arrange horizontally = in side-by-side), $m ((arrange as a grid)
ARRANGE_WIN(winid, x, y, width, height) *Example 1:* `id = GETWINID(NEWWIN())` `ARRANGE_WIN(id, 20, 30, 300, 400)` *Example 2:* *Loads the data set into the current window (creates a new one if none exists) and arranges the window* `RE(["exam1d_13C", "2", "1", "c:/bruker/topspin/data"])` `id = GETWINID()` `ARRANGE_WIN(id, 20, 30, 300, 400)`	Arranges the nternal window with the specified ID. *winid* = window identifier as returned by GETWINID() *x* = x coordinate *y* = y coordinate *width* = window width (pixels) *height* = window height (pixels) *x/y/width/height* must be specified as integers. The point (0, 0) is the upper left corner of a window, the point (*width-1, height-1)* is the lower right corner.
result = ALL_WINDOWS() *Example:* *Prints the window Ids of all open windows.* `for win in ALL_WINDOWS():` `MSG(GETWINID(win.getUserPanelData()))`	Returns the all currently open internal windows (in their creation order) *result* = list of windows which have type InFrame. In order to get the user panel of an InFrame (which is required e.g. by GETWINID(), use the function *getUserPanelData()* of InFrame.
result = SELECTED_WINDOW() *Example:* *Prints the window Id of the selected window.* `win = SELECTED_WINDOW()` `MSG(GETWINID(win.getUserPanelData()))`	Returns the currently selected window, which is the window on which the user recently clicked. *result* = a window of type InFrame. In order to get the user panel of an InFrame (which is required e.g. by GETWINID(), use the function *getUserPanelData()* of InFrame.
SET_SELECTED_WIN(winid) *Example 1:* `SET_SELECTED_WIN("2")` `FT()`	Makes the window with the specified ID the current window of this script. All further actions will happen in this window. *winid* = the unique window ID. The script is cancelled if the id does not exist.
Array Functions	
SAVE_ARRAY_AS_1R1I(reals, imags) *Example:* *Fills 1r with a ramp, leaves 1i as it is.*	Replaces the real or imaginary part of a 1D spectrum (= the contents of the 1r and/or 1i files) by new values. *reals* = list of float numbers to replace 1r

Jython Function	Description
Output and Confirm Dialogs	
`curdat = ["exam1d_13C", "1", "1", "c:/bruker/topspin/data"]` `RE(curdat)` `reals = []` `imags = None` `for i in range(int(GETPAR("status SI"))):` `reals.append(i)` `SAVE_ARRAY_AS_1R1I(reals, imags)` `RE(curdat)`	*imags* = list of float numbers to replace 1i Note: The array may have a size different from the existing file sizes. The status SI parameter is automatically updated.
result = GETPROCDATA(fromppm, toppm, type = None) *Example 1:* `result = GETPROCDATA(-0.5, 0.5)` `text = ""` `for i in range(len(result)):` `text += str(i) + " " + str(result[i]) + "\n"` `VIEWTEXT("GETPROCDATA Test", "Read Real Data", text)` *Example 2:* `result = GETPROCDATA(-0.5, 0.5, dataconst.PROCDATA_IMAG)`	Reads a region of the current processed data set into a Python list (for 1D data only): *fromppm* = region start in ppm *toppm* = region end in ppm type = Optional: dataconst.PROCDATA_IMAG to read imaginary data. Default is real. *result* = list of float numbers or None if data file not found (files 1r or 1i doesn't exist). The list is always ordered from left to right, where “left” is the file start, regardless in which order the region limits are specified . To read the entire spectrum, specifiy large limits, e.g. From -500 to 500 ppm.
result = GETPROCDATA2D(from1, to1, from2, to2, type = None) *Example 1:* `RE(["exam2d_HH", "1", "1", "c:/bruker/topspin/data"])` `result = GETPROCDATA2D(7.4, 8.6, 4.5, 5.5)` `text = "ncols = " + str(len(result)) + "\n"` `text += "nrows = " + str(len(result[0])) + "\n"` `for i in range(len(result)):` `doubleArray = result[i]` `for k in range(len(doubleArray)):`	Reads a region of a 2D processed data set into a Python list: from1, to1 = region along horizontal axis from2, to2 = region along vertical axis These parameters must be specified with respect to the units of these axis. Typically the units are ppm, but for example the vertical axis may have millisecs when the data were transformed in 1 dimension only. Please note: The values of these parameters must lie within the lower/upper limits of the axes ranges! type = Optional: dataconst.PROCDATA_IMAG to read imaginary data. Default is real. result = list of list of float numbers or None if data file not found (files 2rr or 2ii doesn't exist). A list is always ordered from left to right, where “left” is the file start, regardless in which order the region limits are specified .

Jython Function	Description
Output and Confirm Dialogs	
`text += str(i) + " " + str(k) + " " + str(doubleArray[k]) + "\n"` `text += "end of column " + str(i) + "\n"` `VIEWTEXT("GETPROCDATA Test", "Read Real Data", text, 0)`	To read the entire spectrum, specifiy large limits, e.g. From -500 to 500 ppm.
DISPLAY_DATALIST(ydata, props = None, title = "", subwindows = 0) *Example 1:* *Displays 2 ramps without subwindows.* `array1 = []` `array2 = []` `for i in range(100):` `array1.append(float(i))` `array2.append(100-float(i))` `DISPLAY_DATALIST([array1, array2])` *Example 2:* *Displays 2 ramps with a window title and subwindows.* `array1 = []` `array2 = []` `for i in range(100):` `array1.append(float(i))` `array2.append(100-float(i))` `DISPLAY_DATALIST([array1, array2], None, "Example", 1)` *Example 3:* *Displays 2 ramps using display properties* `array1 = []` `array2 = []` `for i in range(100):` `array1.append(float(i))` `array2.append(float(2*i))` `props1 = GET_DISPLAY_PROPS(8.0, 1.0, "rad","x", "y", "Example", "line")` `props2 = GET_DISPLAY_PROPS(8.0, 1.0, "rad","x", "y", "Example", "dots")`	Displays a list of data arrays. Note that TopSpin supports printing and exporting the graphs generated with this function. *ydata* = A list of arrays. Each array must contain float numbers. *props* = Optional. A list of display properties, one for each array. See GET_DISPLAY_PROPS() function. *title* = the window title *subwindows* = 1: display each array in a separate subwindow. 0: display the arrays without using subwindows.

Jython Function	Description
Output and Confirm Dialogs	
`DISPLAY_DATALIST([array1, array2], [props1, props2])`	
DISPLAY_DATALIST_XY(ydata, xdata, props = None, title = "", subwindows = 0) *Example 1:* *Display some x-y pairs.* `yvalues1 = [0, 0, 0.4, 0.2, 0.8, 0.3, 0.1, 0, 0]` `xvalues1 = [0, 20, 50, 70, 120, 240, 300, 400, 400]` `props = GET_DISPLAY_PROPS(-40, 440, "rad","leg-x", "leg-y", "Example", "line")` `DISPLAY_DATALIST_XY([yvalues1], [xvalues1], [props], "X-Y plot test")` *Example 2:* *Display some x-y pairs. With this properties setup, the interactive y-scaling origin is now 0.3 in stead of 0.0. This means: When expanding this curve verically in TopSpin, the y value 0.3 is the expansion origin and will stay on the same spot on the screen.* `props = GET_DISPLAY_PROPS(-40, 440, "rad","leg-x", "leg-y", "Example", "line", "0.3")` *Example 3:* *Display 2 curves in the same window. With the specified properties setup* *– the vertical expansion origin remains at "0.0".* *Furthermore, the display is scaled such that* *– the smallest visible y value is 0.0* *– the biggest visible y value is 2.* *This ensures that both curves are initially entirely visible in the display window.* `yvalues1 = [0, 0, 0.4, 0.2, 0.8, 0.3, 0.1, 0, 0]` `xvalues1 = [0, 20, 50, 70, 120, 240, 300, 400, 400]`	Displays a list of data arrays where y and x values are given. *ydata* = A list of arrays. Each array must contain float numbers. *xdata* = The x axis values for *ydata*. *props* = Optional. A list of display properties (axis and other), one for each array. See GET_DISPLAY_PROPS() function. If *props* = *None*, the x axis units are points. *title* = the window title *subwindows* = 1: display each array in a separate subwindow. 0: display the arrays without using subwindows.

Jython Function	Description
Output and Confirm Dialogs	
`yvalues2 = [y*2 +0.1 for y in yvalues1]` `props = GET_DISPLAY_PROPS(-40, 440, "rad","leg-x",` `"leg-y", "Example", "line", "0.0", 0.01, 2)` `DISPLAY_DATALIST_XY([yvalues1, yvalues2],` `[xvalues1, xvalues1], [props, props], "X-Y plot test"` *Example 4:* *Similar to example 3, but changes the colors of the display and displays one of the arrays as dots rather than lines.* `yvalues1 = [0, 0, 0.4, 0.2, 0.8, 0.3, 0.1, 0, 0]` `xvalues1 = [0, 20, 50, 70, 120, 240, 300, 400, 400]` `yvalues2 = [y*2 +0.1 for y in yvalues1]` `props1 = GET_DISPLAY_PROPS(-40, 440, "rad","leg-x",` `"leg-y", "Example", "line", "0.0", 0.0, 2)` `props2 = GET_DISPLAY_PROPS(-40, 440, "rad","leg-x",` `"leg-y", "Example", "dots", "0.0", 0.0, 2)` `DISPLAY_DATALIST_XY([yvalues1, yvalues2],` `[xvalues1, xvalues1], [props1, props2], "X-Y plot test")`	

Jython Function	Description
Output and Confirm Dialogs	
`# change the color of displayed arrays` `import java` `colorProp = root.UtilProp.getProperties(root.UtilPath.getViewerProp());` `newColorProp = java.util.Properties()` `newColorProp.putAll(colorProp)` `newColorProp.remove("DRAW_DOTS") # because dots were explicitly set above` `newColorProp.setProperty("DATA_COLOR_1", "255 0 0") # red as RGB` `newColorProp.setProperty("DATA_COLOR_2", "0 255 0") # green as RGB` `newColorProp.setProperty("DATA_COLOR_3", "0 0 255") # blue as RGB` `nmr.jutil.misc.UtilSel.orderAction(mfw.UtilDesktop.getSelectedUserPanel(), "PropertiesChanged", newColorProp);`	
GET_DISPLAY_PROPS(xStart = None, xEnd = None, xUnit = None, xLegend=None, yLegend=None, info=None, drawMode="line") *Example:* `GET_DISPLAY_PROPS(2.0, 8.0, "rad","x", "y", "Example", "line")`	Returns display properties as optionally used by the DISPLAY_DATALIST functions. *xStart* – left x axis limit, e.g. 0.0 *xEnd* – right x axis limit, e.g. 10.5. If *xEnd* < *xStart,* axis labelling is reversed. *xUnit* – x axis unit, e.g. "ppm" *xLegend* – extra text displayed below x axis *yLegend* – extra text displayed left of y axis *info* – extra text displayed in the upper left of the data window *drawMode* – "dots" or "line". In the first case the data points are displayed in form of small circles, without connecting lines. In the second case the data points are drawn in form of a polyline.
Acquisition, Processing And Analysis Functions	
SAVE_SHAPE(name, type, amplitude, phase) *Example:* `import math`	Save a pulse shape on disk

Jython Function	Description
Output and Confirm Dialogs	
`# calc. a sine shape (0..PI)` `amplitudes = [] # normalized to 0...100` `for i in range(512):` `amplitudes.append(100*math.sin((3.1415*i)/512))` `phases = [] # in degrees` `for i in range(512):` `if i<256:` `phases.append(360*math.pow(float(i)/256, 3))` `else:` `phases.append(360*math.pow(float(512-i)/256, 3))` `# save the shape under this name in ASCII shape format` `shapeName = "SinBG.512"` `SAVE_SHAPE(shapeName, "Excitation", amplitudes, phases)`	
SAVE_SHAPE_AS_DATASET(amplitude, phase, destination, offset, sw) *Example:* `# save the shape as a NMR dataset, offset and sw define x axis scaling of shape on display` `offset = 1000.0` `sw = 1000.0` `dataset = ["SinBG.512", "1", "999", "c:/nmrdata"]` `SAVE_SHAPE_AS_DATASET(amplitudes, phases, dataset, offset, sw)`	Save a pulse shape on disk as an NMR dataset. Any existing data of the same name will be deleted!
Examples: `FT(), EFP(), ZG(), APK(), XFB() etc.`	TopSpin provides functions for most of the TopSpin commands for acquisition, processing and analysis. The name of a function is derived from the respective command by using capital letters.

Jython Function	Description
Output and Confirm Dialogs	
	In case a desired function is missing please use: XCMD("TopSpin Command") e.g. XCMD("abs n")
Reading Peak Lists	
`list = GETPEAKSARRAY()` `if list == None:` `  MSG("no peaks file found, do 'pp' first")` `  EXIT()` `slist = ""` `for peak in list:` `  slist += "ppm=" + str(peak.getPositions()[0]) +\` `  "  intens="+ str(peak.getIntensity()) + "\n";` `MSG(slist)`	TopSpin stores peak lists (command *pp*) in the file *peak.txt* or (from 2.1 on) in *peaklist.xml*. This example reads the peaklist of the currant dataset (regardless which file is present) and prints the found positions and intensities. The intensities are relative values scaled such that the maximum value in the list is equals to the parameter CY, as defined at peak picking time.

4.4 Special Applications

Changing the spectra display range

This peace of code sets the spectral range of the currently displayed data set to 7.4-6.5ppm.

```
fullrange = putil.DataChecks.getNMRDataOfSelectedFramePrintMsg().getFullPhysicalRange()
newRange = fullrange
newRange[0].setStart(7.5)
newRange[0].setEnd(6.5)
newRange[0].setUnit("ppm")
XCMD(".zx", 1, newRange)
```

Multiple display

- Checking whether the current display is in multiple display mode (several overlayed spectra): This function will return true or false
  `SELECTED_WINDOW().isMultipleDisplayActive()`
- Entering multiple display mode: XCMD `(".md")`
- Entering multiple display mode, but do not auto-reload previous data: XCMD `(".md no_load")`

4.5 User Interface Programming

TopSpin Jython scripts may display any kind of graphics and user interfaces by utilizing the Java Swing classes. The Java Development Kit (JDK) documentation is the basis for that.

The jython code in the table below will produce the following window. Pressing FT will send the command “ft“ to TopSpin. Pressing TD will *not* send the command “td“ to TopSpin, but get the parameter TD from the current data set and display the result in a dialog.

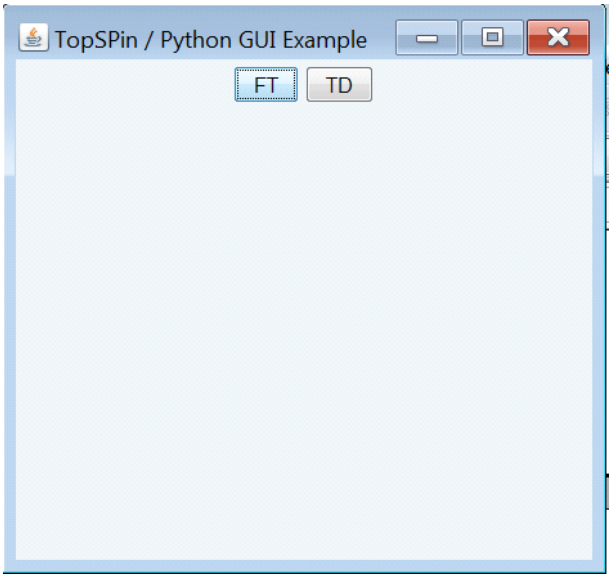


Figure 4.1: TopSpin / Python GUI Example

```
# Simple GUI example
from javax.swing import *
from java.awt import *

def execute_ft(event): # executes FT button
# execute command "ft" in a command thread
EXEC_PYSCRIPT('XCMD("ft", 0)')

def get_td(event): # executes TD button
# get TD and save it as a global property "MY_TD" of TopSpin
ct = EXEC_PYSCRIPT('root.Globals.globalProp.setProperty("MY_TD",
GETPAR("TD"))')
ct.join() # wait until EXEC_PYSCRIPT is done

td = root.Globals.globalProp.getProperty("MY_TD") # get "MY_TD" back from
the globals properties
EXEC_PYSCRIPT('MSG(str(' + td + '))') # show message dialog

# defined a frame with 2 buttons
button1 = JButton('FT', actionPerformed = execute_ft)
```

```
button2 = JButton('TD', actionPerformed = get_td)

frame = JFrame('TopSPin / Python GUI Example') # create window with title
frame.setSize(400, 400) # set window size x, y
frame.setLayout(FlowLayout()) # layout manager for horizontal alignment
frame.add(button1)
frame.add(button2)
frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE)
frame.setVisible(True)
```

Very important for GUI programming:

GUI programs usually contain “action listeners“. These are functions or classes such as `execute_ft` and `get_td` in the code above will get invoked when a button is pressed. The code in action listeners is executed in a program thread different from the thread of the “normal“ code of the python program. Now, many of the TopSpin functions require that they are executed in a thread of type “CmdThread“, a special TopSpin object. These functions won't work therefore inside action listeners.

The code above shows how to circumvent this problem. The critical functions must be converted to run in a CmdThread by using the `EXEC_PYSCRIPT` function. The following table explains that again:

Normal code	Respective code inside action listener
`XCMD("ft")`	`EXEC_PYSCRIPT('XCMD("ft", 0)')` (the argument 0 forces XCMD not to wait until ft is done to be able to quit the action listener as soon as possible to prevent blocking)
`td = GETPAR("TD")`	`ct = EXEC_PYSCRIPT('root.Globals.globalProp.setProperty("MY_TD", GETPAR("TD"))')` `ct.join() # wait until EXEC_PYSCRIPT is done` `td = root.Globals.globalProp.getProperty("MY_TD")` (TD must first be saved in the global store of TopSpin and retrieved from there as soon as EXEC_PYSCRIPT is complete which can be tested using join()).

5 Contact

Manufacturer

Bruker BioSpin GmbH
Rudolf-Plank-Str. 23
D-76275 Ettlingen
Germany

E-Mail: *nmr-support@bruker.com*
http://www.bruker.com
WEEE DE43181702

Bruker BioSpin Hotlines

Contact our Bruker BioSpin service centers.

Bruker BioSpin provides dedicated hotlines and service centers, so that our specialists can respond as quickly as possible to all your service requests, applications questions, software or technical needs.

Please select the service center or hotline you wish to contact from our list available at:

https://www.bruker.com/service/information-communication/helpdesk.html

Contact

Lastpage

Lastpage

Bruker Corporation

info@bruker.com
www.bruker.com

Part No: H166420