****
****

VnmrJ / VNMR Parameter Handling

by Rolf Kyburz, Agilent Technologies

****
****

Last update: 2011-03-14

Contents:

---

1. Introduction

Many of the problems that users have with VNMR have to do with the way VNMR handles parameters. Unfortunately there is no complete explanation on parameter handling in the VNMR manual, and there is also no "flow diagram" that would indicate what happens to parameters within VNMR. This document was created in an effort to help with questions dealing with VNMR parameters. Should there be important omissions or things that are wrong or unclear, please contact the author under

rolf.kyburz@varianinc.com

The following text refers to VNMR - it was written when VNMR was the standard Varian NMR software; however, the principles of parameter handling and storage haven't changed in VnmrJ, therefore this text is still valid also for VnmrJ.

2. Parameter Trees

First about the parameter trees - VNMR knows four parameter trees; each of them is stored in a separate disk file, and all of them have their particular set of properties and behavior. As long as VNMR is running, they all are kept in memory for speed reasons.

a) "systemglobal": this tree is stored in "/vnmr/conpar". Under normal circumstances this tree is read only once per session (when VNMR is started up), and never thereafter. "systemglobal" parameters (configuration parameters) can be changed "by hand" - just by entering a value, or with the command

```
setvalue('<parametername>',value,'systemglobal')
```

but that parameter value will usually be lost upon exiting VNMR, as "/vnmr/conpar" is NOT saved back on disk upon exiting VNMR. Usually, only "config" is used to change "/vnmr/conpar". That command will also update the "systemglobal" tree in the VNMR which called it - but for starting it will read the parameter values FROM THE DISK, even if "systemglobal" parameters have been changed temporarily. It is possible to force VNMR to read or write "/vnmr/conpar", using the commands

```
fread('/vnmr/conpar','systemglobal')
fsave('/vnmr/conpar','systemglobal')
```

(the latter of course only works if you have write permission on the file "/vnmr/conpar").

b) "global": this tree is stored in "~/vnmrsys/global" ("systemdir+'/global'")

and is also read in upon starting VNMR. Different from "/vnmr/conpar" it IS written back to the disk file upon exiting VNMR. This has some consequences for the user: it does NOT make sense to manipulate the disk file while VNMR is running - upon exiting VNMR will write back the copy that is kept in memory - unless you do

```
fread(userdir+'/global','global')
```

after the modification. A problem that has happened several times is the following: a user has not been careful, and the disk gets full while VNMR is running. When the user realizes that there is a problem, the first reaction often is, to exit and restart VNMR - which is exactly wrong! Upon quitting VNMR will erase and overwrite the "global" parameter file: in this situation it will very likely be able to erase the parameters, but then it may not be able to complete the write-back operation! this leaves a set of corrupted parameters - an incomplete file - on the disk. Upon restarting the user will then will then get error messages such as

```
parameter wcmax not found
```

and the like - a symptom for missing "global" parameters. In this situation even experienced users often do the wrong thing: they substitute the corrupted file with the intact backup copy in "/vnmr/user_templates" while VNMR is running! That's of course bad, as a) VNMR will not automatically read those parameters (unless it is started or forced to do that using fread) - the error messages therefore persist - and b) upon exiting VNMR will of course write back the incomplete parameter set it has kept in memory, and with the next restart one gets the same error messages again! There is a two simple rules to avoid that:

a) if the disk is full, FIRST free up some space on the disk BEFORE exiting VNMR
b) if you need to substitute "~/vnmrsys/global", ONLY do it while VNMR is NOT running!

c) "current": this is the parameter tree you are looking at through "dg" etc.; your "working parameter set". Over 99% of the parameter changes occur in this tree. It is experiment-related and stored in "curexp+'/curpar'". As long as you are working in a particular experiment, its "current" parameter tree (and also the "processed" tree, see below) is buffered in memory, and it is written back to the disk either upon quitting VNMR "regularly", as well as when joining a different experiment. You can force the experiment parameters to be written to the disk by using the command "flush" (this will also cause the "processed" and "global" trees to be written to the disk. You could also force the system to read "current" parameters from the disk using

```
fread(curexp+'/curpar')
```

For the same reasons as with "~/vnmrsys/global" it is recommended NOT to manipulate the disk files within the active experiment, but rather to join a different experiment first. You can READ the experiment files from the disk AFTER calling the "flush" command. While setting up new experiments, or even after starting an acquisition the disk files often have NOTHING to do with the "current" data, and it is absolutely essential to give VNMR a chance to write back the data it keeps in memory, otherwise the files in the current experiment may be inconsistent or even corrupted! For that reason it is essential that you quit VNMR by using the "Exit VNMR" button, or by typing "exit", rather than by just pulling "EXIT" from the desktop menu while VNMR is still running: this would kill all applications running in the windowing environment, including VNMR - and in such a situation VNMR has no chance to save its parameters before exiting.

d) "processed": this is the second, experiment-related parameter set. Its basic handling (buffering, reading/writing from/to the disk) is the same as with the "current" tree; the disk file is "curexp+'/procpar'". The principal purpose of this parameter tree is, to maintain/guarantee consistency between the experiment parameters and the acquired and/or processed data in that same experiment, see also below (parameters can be changed after typing "go", or after the acquisition - after typing "wft" the "current" tree will always be reset to the "proper" parameter values). To read in the "processed" tree by command you can use

```
fread(curexp+'/procpar','processed')
```

to force a write-back operation you can use "flush" (or "fsave", but that requires the same arguments as "fread").

3. Parameter Tree Precedences

The precedence among the parameter trees is

current > global > systemglobal

This normally is irrelevant for standard operation, as it is a very specific set of parameters that is stored in "/vnmr/conpar" ("systemglobal" tree), and another, very specific set of parameters is stored in the "global" tree. BUT: it does mean that if you incidentally create a local parameter (in the "current" tree) with the name of a "global" or "systemglobal" parameter it will OVERRIDE that other parameter! Conclusion:

- DON'T duplicate "systemglobal" parameters in either the "global" or the "current" tree, and
- DON'T duplicate "global" parameters in the "current" tree unless you know what you are doing!

4. Parameter Groups

Among the parameter characteristics (as defined through one of the numeric parameter qualifiers, see the user programming manual) is the parameter group: each parameter belongs to a certain functional group, as described in the manual for the "setgroup" command:

- acquisition
- processing
- display
- spin simulation
- sample
- "all" / "none" (not used)

this group determines how the parameters are handled by VNMR:

- upon typing "go", ONLY the ACQUISITION parameters from the "current" experiment (plus of course the configuration parameters) are transferred to the pulse sequence. Even if a parameter "delta" or "delta1" exists in the "current" tree - it is a display parameter and will not be available in a pulse sequence:

  getval("delta")

  will return 0.0 or an error message! So if you don't "see" a parameter in a pulse sequence, check whether it really is an acquisition parameter!
- only acquisition parameters will affect "array" and "arraydim" when they are arrayed (and even that only if protection bit 256 (bit #8) is not set).
- upon typing "go" the acquisition parameters (only) are copied from the "current" tree into the "processed" tree.
- upon processing, the "processing" parameters are copied from the "current" tree to the "processed" tree, and - after the FT - both the "acquisition" and the "processing" parameters are copied BACK from the "processed" tree into the "current" tree. Even when you change "acquisition" parameters after the acquisition: "wft" will always restore the "proper" parameters for the current data set.
- the display parameters are always taken from the "current" tree.
- "dg" ALWAYS looks at the "current" tree ONLY.
- direct parameter handling (e.g., "d1=1") ALWAYS refers to the "current" tree only.
- "ap" / "pap" takes the "acquisition" parameters from the "processed" tree to ensure a parameter printout that is consistent with the spectrum that is being plotted. As a little sideline: it is a known bug that decisions in the "ap" display template are based on parameter values that are taken from the "current" instead of the "processed" parameter tree!

5. Parameter Handling Utilities

Apart from the direct parameter entry facilities, VNMR has a complete set of parameter manipulation tools:

- parameters can be created, destroyed, displayed and manipulated IN ANY TREE

using the commands

```
create
destroy
setgroup
setprotect
setlimit
setvalue
setenumeral
display
paramvi / paramedit
```

Note that "setvalue" bypasses any internal parameter protection (protection bits, parameter limits, enumerals) and should be used with caution - but it is THE tool to set a parameter in the "processed" tree:

```
setvalue('ct',128,'processed')
```

"paramvi" also is to be used with caution, as it gives the user any chance to screw the parameter format.

- entire parameter groups can be copied or even destroyed using

```
groupcopy
destroygroup
```

the latter of course is to be used with caution - there will very rarely be a need for that command at all. If the entire "processed" parameter tree needs to be fixed / manipulated it is often easier to change / set up the parameters in the "current" tree, and then to replace an entire group of the "processed" tree using e.g.:

```
groupcopy('current','processed','acquisition')
```

(note that the argument "'all'" currently doesn't work with the groupcopy command).

- the command "prune" permits removing any extra parameter from the "current" tree that is NOT also defined in the specified parameter disk file.
- entire parameter trees can be saved to disk or read from a disk file using

```
fsave
fread
```

- entire parameter SETS (together with the text file) can be saved on disk and restored using

```
svp
rtp / rt
```

Note that "svp" / "rtp" work with the CURRENT parameter tree, whereas "svf" (and "rt" on a disk file) will work with the PROCESSED parameter tree. Note also that "rtp" can also be used on "*.fid" files, where it only reads the parameter part of the file into the "current" tree. The command "mp" which moves parameters between experiments only works with the "current" trees of the specified experiments..

- with the command

```
flush
```

the entire set of buffered parameters (and data) can be forced to be written onto the disk without quitting VNMR or joining a different experiment.

The one thing that is missing from VNMR is a utility to

- display ALL parameters - even those not shown by any "dg" / "ap" template,
- list ENTIRE parameter groups or even parameter trees in an easy-to-read, alphabetic (one parameter per line) format
- compare two parameter sets and produce easy-to-read output

This is provided in the on-line user library: just send an e-mail message to

```
userlib.request@varianinc.com
```

with a line

```
bin/parhandler
```

in the message body. this item hasn't yet been incorporated into the user library directory that is distributed with the VNMR software.

The actual parameter disk file format is rather convoluted and is described in the User Programming Manual. It is not recommended to manipulate parameter disk files directly - rather use the commands described above.

---

6. Related Articles from Agilent MR News

1994-07-07:

USING GLOBAL PARAMETERS CREATIVELY:

As you know, VNMR includes a series of "global" parameters which are common to all the experiments for a particular user. The standard parameters which exist are things like "plotter", "lockpower", etc. However, nothing prevents you from creating new global parameters for your own use:

```
create('gr1','real','global')
```

```
create('gn1','string','global')
```
Having such general purpose parameters, you can use them to transfer parameters _between_ experiments. For example,
```
gr1=rfl gr2=rfp jexp2 rfl=gr1 rfp=gr2
```
would transfer referencing from one experiment to another.

A second interesting use for such a parameter is to save an array. Let's say you typed in a complicated array of d2 values. You want to use the same array again, but just for a second you want to set d2=0. Before doing so, type gr1=d2 [yes, you can do this for arrayed parameters and it works]. Now set d2=0, do your experiment, and then d2=gr1 restores the array as you desire. One catch - whenever you set gr1 to be a series of values (an array), it sets the parameter array='gr1'; you then need to reset array to '' (or whatever is appropriate).

[Agilent MR News 1994-07-07]

1994-12-09:

CHANGING CONFIGURATION PARAMETERS

It is well known among VNMR users that /vnmr/conpar can only be changed through "config". Parameters CAN be changed "manually", though, e.g. by typing
```
vttype=0
```
but this will only affect the parameters in memory, and not /vnmr/conpar. If "config" is called after such a change, it will NOT reflect the parameter settings stored in memory, but it will rather read in /vnmr/conpar FROM THE DISK - that makes you believe that vttype=0 does not work! What may be even more confusing is, that upon quitting with "Exit & Save", config not only stores its own parameters in /vnmr/conpar, but also causes VNMR to re-read the configuration parameters from /vnmr/conpar, i.e., upon calling config any temporary parameter changes are LOST and would have to be re-entered!

Actually, there is a way to store "manual" parameter changes in /vnmr/conpar, if you have permission to overwrite /vnmr/conpar (usually vnmr1 only):
```
fsave('/vnmr/conpar','systemglobal')
```
This should NOT be used routinely under normal circumstances (it is used in the setlockfreq macro).

[Agilent MR News 1994-12-09]

1995-03-26:

NEW PARAMETER HANDLING TOOLS:

Have you ever been fed up with looking for "hidden" parameters? Did you ever have to use printouts to compare two different parameter sets (e.g.: parameters in two different experiments, or parameters in an experiment with those in a disk file)? Have you ever had a hard struggle trying to use the "diff" command to find the difference between two parameter sets? [Whoever has tried this knows that "diff" is virtually useless, as the VNMR parameter format uses three or more lines for each parameter, "diff" therefore would show the difference between the values, but not the associated parameter name! Another problem is, that the parameter sorting in VNMR is never exactly the same (and certainly not fully alphabetical).] In "bin/parhandler" the on-line user library now contains new parameter handling tools that allow listing COMPLETE parameter sets in a simple - one line per parameter - format (either all parameters, or just acquisition, processing, display, ... parameters). Additional tools permit printing out complete parameter difference tables (between experiments and/or disk files), which make parameter comparison a very simple task! The pardiff macro never "forgets" parameters!

[Agilent MR News 1995-03-26]

1997-02-11:

PARAMETER UNITS IN VNMR 5.3:

In VNMR 5.3, the units are no longer "built into VNMR", but they are all completely user-definable. The "standard" units "p", "d", and "k" are defined in the bootup macro (/vnmr/maclib/bootup) which in VNMR 5.3 contains the following three lines:
```
unit('p','ppm','reffrq')
unit('d','ppm','dfrq')
```

```
        unit('k','kHz',1000)
```
If on your system the unit suffixes (sw=10p etc.) don't work, then it most likely is because you have a local "bootup" macro. As noted in VMR News on 1996-11-20, you should NOT have a local bootup macro: customized actions in the bootup macro should rather be moved to a local macro "login". If you want to have a customized "bootup" macro (because you don't want to customize every account), then you must customize the VNMR 5.3 version of the "bootup" macro - don't use an older version of that macro!

[Agilent MR News 1997-02-11]

1997-05-02:

FIXING EXPERIMENT PARAMETERS AFTER A VNMR UPGRADE:

The macro "fixpar" has got two tasks:
- it ensures that parameters that are loaded into VNMR are consistent with the local soft- and hardware configuration, and
- if a parameter set was generated with an earlier VNMR release, it calls "parfix" to make it compatible with the current software release (every version of VNMR comes with its own version of the "fixpar" and "parfix" macros).

The macro "fixpar" is called with every "rt" or "rtp" (this includes "setup", loading through "files", etc.): parameter updating is implicit and automatic - with one exception: the parameters in the current experiments for every user are NOT updated. To cover this as well, every user would need to call "fixpar" in EVERY local experiment after a VNMR upgrade. Tom Frenkiel (MRC Mill Hill, London, U.K.) has proposed a method that makes this automatic. We first store the following macro in EVERY user's local maclib:

```
        "fixexp - fix experiment parameters"
        fixpar
        $ix=2 $exp=''
        repeat
          format($ix,1,0):$exp
          exists(userdir+'/exp'+$exp,'directory'):$e
          if $e then
            jexp($ix) fixpar
          endif
          $ix=$ix+1
        until $ix>9
        jexp1
```
This calls "fixpar" in every defined experiment. We now add the following segment to the bottom of /vnmr/maclib/bootup:
```
        if $bg=0 then
          exists(userdir+'/maclib/fixexp','file'):$e
          if $e then
            fixexp
            delete(userdir+'/maclib/fixexp')
          endif
        endif
```
This calls "fixexp" ONCE by the bootup macro for every user, then the fixexp macro is automatically removed (this can only be done in foreground, as joining experiments is not permitted in background).

[Agilent MR News 1997-05-02]

DOES FIXPAR FIX ALL PARAMETERS?

With the "fixpar" macro we ensure that when you type "go" your parameters are complete and comply with your spectrometer hardware. This not only covers the case of a simple parameter setup: consider the case where you want to repeat an experiment that you have performed under an earlier VNMR release, or on a different type of spectrometer: you can simply retrieve parameters from an existing FID using "rtp", check the power levels and pulse width settings, and type "go".

But there are cases where you DON'T want the parameters to be adjusted: when you process existing data, you want VNMR to indicate what parameters were used for the acquisition of your data - and those parameters may be incompatible with your software, maybe even dangerous for your hardware configuration! To give you an example: when you retrieve an old FID, "fixpar" is called, and your CURRENT parameters are updated (such that you could redo the experiment). But when you type "df" or "wft" (any command that accesses the FID itself),

the acquisition parameters from the PROCESSED tree are copied into the current tree and may now be incompatible with your setup! To avoid such problems, it is recommended not to use "rt" to retrieve parameters for acquisition.
"fixpar" also does NOT cover experiment subfiles (as created by combination macros such as hcosy, hccorr): these subfiles are normally used for processing only, and not to set up an experiment.

[Agilent MR News 1997-05-02]

1997-09-27:

WHAT TO DO WHEN GLOBAL PARAMETERS ARE MISSING:

Even though we have taken almost every possible measure to prevent the global parameters from getting corrupted (such as when the disk is full when exiting VNMR), we still occasionally get reports about problems with this parameter file (even with VNMR 5.3B) see also bug report global.5101. Typical symptoms of a corrupted global file are messages such as

```
variable "lastmenu" does not exist.
```

upon starting VNMR, and in most cases you will not even get the second row menu bar. In such an instance, don't try creating the missing VNMR parameter - you will find that many more are missing as well! Instead, EXIT VNMR (type "exit" on the VNMR command line), then type

```
cp /vnmr/user_templates/global ~/vnmrsys
```

and then restart VNMR (VNMR MUST NOT BE RUNNING when you copy in the parameter file). After that, you can restart VNMR. You will need to re-select the printer and plotter, as your global parameters will now be reset to their default values. It would be helpful to keep a backup copy of your current global parameter set (~/vnmrsys/global).

One of the key points for avoiding such situations (at least with VNMR 5.1 and earlier) is, to avoid "disk full" situations: periodically use "df -k" to ensure that there is enough free space on your disk!

[Agilent MR News 1997-09-27]

1997-11-21:

HOW TO AVOID PARAMETER CHANGES UPON "RT":

Whenever you retrieve a data file, the parameters for the new FID are copied both into the "current" and into the "processed" parameter trees. After that, the "fixpar" macro is called, which ensures that the current parameter tree is in accordance with your instrument and software configuration. This seems to be against all rules of GLP - but the big advantage of this mechanism is that after an "rt" you can basically re-acquire a spectrum, even if you are retrieving data that were acquired with earlier software, or even with a different spectrometer! We still maintain GLP requirements, in that after a "df", "wft" or "wft2d", the acquisition parameters from the processed tree are copied into the current parameter tree. Now, the parameters that you see are those used for the acquisition - however, you may not be able to type "go", if you have either changed the software or if you have recalled data from an other spectrometer!

On a datastation, the "fixpar" parameter updating is often undesirable. Fortunately, as of VNMR 5.3B, "rt" is a macro. We can therefore simply add the following lines at the end of the file /vnmr/maclib/rt:

```
if system='datastation' then
  if arraydim=1 then
    df dg
  else
    groupcopy('processed','current','acquisition')
    dg
  endif
endif
```

This doesn't prevent fixpar from altering the current parameter tree, but it copies the processed tree over the current one immediately thereafter.

[Agilent MR News 1997-11-21]

1998-01-09:

PRINTING THE VNMR CONFIGURATION PARAMETERS:

The VNMR "config" window includes a "Print" button. Note that this button

only works if you have a printer with the name "lp" ("config" does not look up which printer is selected in VNMR, nor does it check whether there is a LPDEST environment variable and what its value is). If you don't whether have a printer named "lp", you should use the procedure that is described in the VNMR Command and Parameter Reference Manual: first exit (and save) "config", then make sure you have selected the right printer in VNMR and type

```
printon config('display') printoff
```

[Agilent MR News 1998-01-09]

1998-01-30:

A PECULIARITY WITH THE GROUPCOPY COMMAND:

Some users (including the editor) have mis-interpreted the 'all' option of the groupcopy command: 'all' does NOT copy all parameters, but rather the parameters of the (rarely used) group named 'all' (i.e., parameters that would be associated with all parameter groups). Valid group types include 'sample', 'acquisition', 'processing', 'display', 'all', and 'spin'. In order to copy all parameters from one tree to another you need to call groupcopy for each of these parameter groups, including 'all', 'spin', and 'sample'. In most cases it is sufficient to copy the most commonly used groups 'acquisition', 'processing', and 'display'.

[Agilent MR News 1998-01-30]

NEGATIVE PARAMETER STEP SIZES:

The limits (maximum, minimum) and the step size of any VNMR parameter can be displayed with

```
display('<parametername>')
```

A few parameters (spectral widths and Fourier numbers) have negative stepsize values - what does this mean?

If parstep is less than -1, then the corresponding parameter is interpreted as a Fourier number, which must be set to a number which is a power of 2. For fn, the value is set to -2, although any value below -1 will give the same behavior.

If the value of parstep is between -1 and 0, the parameter is interpreted as a spectral width parameter. Here, the set of possible values consists of the inverse of the dwell times that the pulse programmer can execute, i.e., it depends on the timing resolution of the pulse programmer. The latter can be taken from the step size of the parameter (12.5, 25, or 100 nsec). The better the timing resolution is, and the smaller the sw parameter is, the smaller the steps between possible values for sw. An example for UNITY INOVA systems: at the upper end, spectral windows can be 5e6, 4705882.4, 4444444.4, 4210526.3, 4e6 Hz, etc.; at 100 KHz the step size is approximately 125 Hz, at 10 KHz it is around 1.25 Hz, and at sw=1000 and less the step size is well below 0.1 Hz.

[Agilent MR News 1998-01-30]

1998-08-21:

DEFINING PARAMETERS FOR R.F. POWER LEVELS:

When defining new parameters for power level control, users usually just use a command such as

```
create('satpwr','integer')
```

This is fine for entering the power level, however, that new parameter does not have proper limits from this command. The probe protection software does NOT work properly if values BELOW THE MINIMUM are entered (e.g.: satpwr=-20). One starts getting power levels close to maximum output when entering values just below the minimum for the hardware (see also the bug report power.6101).

We strongly recommend defining appropriate limits for parameters that are used for r.f. power levels. For power levels on the observe channel this can be done with

```
setlimit('satpwr',17)
```

For the decoupler channels the limit index is different:

```
setlimit(<parametername>,9)     "first decoupler channel"
setlimit(<parametername>,18)    "2nd decoupler"
setlimit(<parametername>,21)    "3rd decoupler"
```

This way, the parameter limit will reflect the limits as set up via config for each of the r.f. channel, and in case of high power levels, the hardware will be protected by the probe protection software. It may be even safer to use

lower maximum limits for parameters used for presaturation or decoupling power levels. This can be done with

```
setlimit('satpwr',40,-16,1)
```

for systems with 79 dB attenuator, and with

```
setlimit('satpwr',40,0,1)
```

for older systems with 63 dB attenuator. We strongly recommend adding such lines to pulse sequence setup macros such as /vnmr/maclib/cyclenoe.

[Agilent MR News 1998-08-21]

1998-08-28:

PARAMETERS WITH "INDIRECT" LIMITS:

VNMR parameters are equipped value entry limits, in order to force operator or MAGICAL entries into the allowed range. Even without explicitly specified limits, there are at least numeric limitations (given by the maximum / minimum 32-bit floating point number, -9.99999984307e17 .. +9.99999984307e17). String parameters have length limitations (given by the size of the allocated memory inside VNMR). Narrower limits and a defined stepsize can be specified using the "setlimit" command, e.g.:

```
setlimit('satpwr',40,-16,1)
```

With some parameter types (delay, frequency, integer, and pulse) narrower limits are automatically defined when defining the parameter using "create".

Many acquisition parameters have limits that depend on the spectrometer hardware, especially the pulse programmer. It would be impossible for "fixpar" to correct all parameter limits based on conpar, every time a parameter set is read in. Instead we have created the mechanism of "indirect" parameter limits. Parameters with indirect limits have the protection bit 13 (value 8192) set. This can be done using

```
setprotect('<parametername>','on',8192)
```

If this bit is set, then the limit fields in the parameter definition are no longer the actual numeric limits themselves, but rather indices into three arrayed parameters parmax, parmin, and parstep. These parameters are part of /vnmr/conpar and are set up by the config program. This way all parameters with indirect limits have their limits automatically adjusted by config.

In VNMR 6.1A, parmax, parmin and parstep are arrays of 21 numeric values. Here is a list of what the various limit values are used for:

Index	max/min/stepsize (typical)	Typical use
1	500/0/0.1	sc
2	840/5/0.1	wc
3	500/0/0.1	sc2
4	520/0/0.1	wc2
5	5e+06/100/-1.25e-08	sw
6	256000/1000/1000	fb
7	100000/-100000/0.1	tof (obsolete!)
8	99000/-99000/0.1	dof (obsolete!)
9	49/-16/1	dpwr, dhp
10	39/0/1	dlp
11	2e+06/1/1	dmf
12	500/-500/1	loc
13	8190/0/0.0125	p1,pw,pw90,rof1,rof2,alfa
14	8190/0/1.25e-08	d1,d2,d3
15	1e+06/0/0.0125	pw,p1,tau (solids NMR)
16	100000/-100000/0.1	dof2 (obsolete!)
17	63/-16/1	tpwr
18	49/-16/1	dpwr2
19	32767/-32767/1	(shim gradient DAC values)
20	100000/-100000/0.1	dof3 (obsolete!)
21	49/-16/1	dpwr3

The "create" command automatically uses indirect limits for pulses (13) and delays (14). Beyond that, you can turn on indirect parameter limits using the setlimit command, e.g.:

```
setlimit('satpwr',9)
```

If you only specify one limit parameter, the setlimit (as of VNMR 5.1) takes this as an index for all three limits. The above command is equivalent to

```
setlimit('satpwr',9,9,9)
setprotect('satpwr','on',8192)
```

The limits for tof (7), dof (8), dof2 (16) and dof3 (20) are marked as obsolete: since many releases, frequency offsets can be arbitrarily big, see also VMR News, 1997-11-14. You can deactivate indirect parameter limits

simply by entering explicit limits, e.g.:

```
setlimit('tof',1e9,-1e9,0.1)
```

You can display all properties (including the value(s), limits, and protection bits) of a parameter using the command

```
display('parameter_name'<,'parameter_tree'>)
```

where 'parameter_tree' is one of 'current' (the default parameter tree), 'processed', 'global', and 'systemglobal'.

Note that the parameters parmax, parmin and parstep are part of /vnmr/conpar and can only be permanently altered by vnmr1!

[Agilent MR News 1998-08-28]

1999-04-08:

READING VNMR PARAMETER FILES WITHIN A UNIX SHELL:

Reading parameter files from a UNIX shell script or a C program is somewhat non-trivial, because VNMR stores parameters on multiple lines, e.g.:

```
bs 7 1 32767 0 1 2 1 0 1 64
1 16
0
axisf 4 2 4 0 0 4 1 0 1 64
1 "s"
4 "m" "n" "s" "u"
```

Every parameter is stored on at least 3 lines (the actual values may be spread over multiple lines in arrayed parameters): the first line contains the name of the parameter and its characteristics (see also section 5.4 of the VNMR User Programming Manual), the second line contains the number of values (1 for non-arrayed parameters), the last line contains enumerative values.

This means that you cannot simply take "grep" to extract the value of a specific parameter from a VNMR parameter file. Fortunately, there is awk / nawk! To read the "system" parameter from /vnmr/conpar, you could use

```
vnmr1 - 1> nawk '/^system / {getline; print $2}' < /vnmr/conpar
"spectrometer"
```

Note the syntax in the matching string: by preceding the pattern with a caret (^) we only pick lines that start with the matching string; the blank after the string excludes longer parameter names that start with "system". The "getline" in the "nawk" string skips to the second line, where we pick the second token - the parameter value. You can of course also directly set a shell variable with the parameter value, and we can use "sed" to remove the double quotes from string values (this example is for Bourne shell scripts):

```
cd /vnmr
system=`nawk '/^system / {getline; print $2}' < conpar | sed 's/"//g'`
```

For arrayed parameters, this would of course only return the first value.

Alternatively, you could download and install "bin/parhandler" from the user library. This contribution contains a utility "listparam" that prints VNMR parameters in a simple, 1 line-per-value format, which can be used as follows:

```
system=`listparam /vnmr/conpar | awk '{print $2}' | sed 's/"//g'`
```

[Agilent MR News 1999-04-08]

2003-10-17:

MAGICAL TRICKS - MOVING SINGLE PARAMETERS BETWEEN EXPERIMENTS:

There are several commands that allow moving data or parameters between two VNMR experiments:

- "mf" moves the FID (with the associated parameters, of course)
- "mp" moves the parameters ("expN/curpar")
- "md" moves stored display settings ("expN/s1" .. "expN/s9", if present)
- "mt" moves the text ("expN/text"); for VNMR releases prior to VNMR 6.1C, "mt" is available with the contribution "maclib/mz" from the on-line user library; for the current version visit "For VnmrJ and VNMR Users" at
  http://www.varianinc.com/products/nmr/apps/vnmrusers.html
- "mz" ("maclib/mz" from the Agilent MR User Library, see above) moves integral reset points (the parameters "lifrq" and "liamp" from the current parameter tree)

Sometimes in a macro context one just would like to transfer single parameters between two experiments. The typical mechanism used for this is

```
$value=par jexp(N) par=$value
```

where "par" would be the name of a VNMR parameter, such as "d1". Note that if

"par" is arrayed, this will transfer the entire array - however, if "par" is an acquisition parameter, the above mechanism will NOT set "arraydim". In general, if you enter an arrayed parameter from another arrayed parameter in one command, this does NOT cause "arraydim" to be adjusted, i.e., calling "d1=d1" would not correct the situation. The solution is to use "calcdim":

```
$value=par jexp(N) par=$value calcdim
```

One user wanted to transfer the value of "llfrq", i.e., the frequencies of a line listing (as created by "nll", "dll", "dpf", or "ppf") from one experiment to the other. The construct

```
$llfrq=llfrq jexp(N) llfrq=$llfrq
```

failed because "llfrq" did not exist in the target experiment: this parameter is NOT "mandatory" - it is created by the above commands if it doesn't exist yet. The solution therefore would be to use

```
$llfrq=llfrq jexp(N) nll:$dummy llfrq=$llfrq
```

Superficially, this looks correct ("calcdim" is not needed, because "llfrq" is a display parameter) - however, line listings always consist of the frequency part ("llfrq") and the associated amplitude information ("llamp") - both parameters have the same number of elements. The above construct sets "llfrq", but leaves "llamp" untouched, leading to a potentially inconsistent situation. It would be a good idea also to transfer "llamp", even though the amplitude information is not needed (and most likely inappropriate!) in the target experiment:

```
$llfrq=llfrq $llamp=llamp jexp(N) nll:$dummy llfrq=$llfrq llamp=$llamp
```

For the same reason, the "mz" macro (see above) moves "lifrq" (the integral reset points) together with the integral values ("liamp").

If in your macro you already are "in" the target experiment you could also use "rtv" for the parameter transfer:

```
rtv(userdir+'/expN/curpar','llfrq','llamp')
```

This works irrespective of whether the parameter exists in the target experiment or not ("rtv" is used as part of the "psgset" macro). In the case of acquisition parameters, you also would need to add "calcdim" in order to keep "arraydim" adjusted. Typically, "rtv" is used to retrieve parameters from stored parameter sets (mostly from "parlib"); "rtv" on an experiment may not be a good idea, unless you just left the source experiment in the same macro: if you have two copies of VNMR running (foreground or background), the source experiment could be an active experiment, in which case "curpar" may not correspond to the real contents of that experiment. In our opinion, the "traditional" method above (using "jexp" and local variables) is preferable.

[Agilent MR News 2003-10-17]

2003-12-13:

POWER HANDLING PARAMETERS IN VNMR / VnmrJ:

Both the UNITY INOVA and the MERCURYplus use a very similar, "dual power handling parameter concept", except that they don't linearize the power attenuator values:

- one set of parameter ("tpwr", "dpwr", etc.) is in dB. It addresses a power attenuator between the transmitter board(s) and the power amplifier. Even though technically these are attenuators (i.e., more attenuation means less power), the power parameters are set up such that a value of 63 gives maximum output, while 0 (for 63 dB attenuators) or -16 (for 79 dB attenuators) yield minimum r.f. power output: more means more power (and hence potentially more danger for both the probe hardware and the sample), which seems better from a psychological point-of-view than to talk about attenuation. Typically, power parameters are defined as integers - and if they weren't, the acquisition software would round off or truncate such parameters to integer values anyway. It is the nature of such logarithmic attenuators that even at the minimum setting the output is NOT zero, but just very small ("tpwr=-16" means that the output voltage is 8913 times lower than at "tpwr=63", assuming a completely linear amplifier).
- a second set of parameters ("tpwrf", "dpwrf", etc., or "tpwrm", "dpwrm" in many solids NMR pulse sequences) addresses a linear r.f. power modulator that is located on the r.f. transmitter board. On the UNITY INOVA, the power modulator accepts values between 0 and 4095 - the power values at the output actually between 1 and 4096, i.e., also with the linear modulator, the minimum output is NOT zero - to obtain zero output we use transmitter gating (e.g., "xmtron()" and "xmtroff()").

See also Agilent MR News 1999-05-28 for related information and a comparison between linear and logarithmic power scales. On the UNITY INOVA, both the

linear modulator and the power attenuator (in dB) take the same instruction time when set via the pulse programmer (0.5 usec), but the linear modulator is not only more practical for pulse shaping (because pulse shapes are usually defined in linear amplitude units), it can also be controlled via a waveform generator, in which case its output can be changed at a rate as fast as every 200 nsec. For information on the time requirements for setting r.f. power attenuators and linear modulators on other instruments see section 2.12 "Internal Hardware Delays" in the manual "VNMR User Programming".

[Agilent MR News 2003-12-13]

2005-08-11:

ACQUISITION WITH CONDITIONAL PROCESSING:

VnmrJ / VNMR knows several types of conditional processing in connection with data acquisition:

- "wbs" processing occurs whenever the end of a block ("bs") of transients is reached - provided of course "bs" is active and smaller than "nt";
- "wnt" processing occurs whenever "nt" scans are reached, i.e., at the completion of an increment in an arrayed or nD experiment;
- "wexp" processing occurs at the completion of the entire experiment;
- "werr" processing may be invoked when an acquisition errors is detected.
- VnmrJ uses yet another type of conditional processing ("wdone"); this should be considered internal to VnmrJ - users should not need to deal with this option.

Such processing can be invoked in various ways:

- When an acquisition is started with "go", NONE of the above conditional processing options are invoked; however, AFTER an experiment is started using "go", you can still activate conditional processing, using the COMMANDS "wbs", "wnt", "wexp", and "werr", e.g.:
  ```
  wbs('testsn') wexp('wft')
  ```
  and once such processing is invoked, it can also be disabled or altered, e.g., with
  ```
  wbs('')
  ```
- Starting an experiment with "ga" is equivalent to
  ```
  go wnt('wft')
  ```
  (typically not a good idea with nD experiments!). Once the experiment is running, any conditional processing can still be invoked or altered
- When you start an experiment with "au" (this is the command that is used in automation), the actions specified in the PARAMETERS "wbs", "wnt", "wexp", and "werr" are invoked, as appropriate, e.g.:
  ```
  wbs='testsn' werr='react' wexp='procplot' au
  ```
  and again, the COMMANDS "wbs", "wnt", "wexp", and "werr" can be used AFTER the "au" to disable or alter any conditional actions.

Typically, "nt" is a multiple of "bs", hence the completion of an increment is at the same time the completion of the last block of "bs" scans; also, the completion of the entire experiment is also the completion of the last increment; this raises the question about the sequence of events at the completion of an increment or of the entire experiment:

- "wbs" processing is NOT done at the completion of an increment / FID.
- "wbs" processing is NOT queued, i.e., while "wbs" or "wnt" processing is active, further "wbs" processing calls are suppressed. This means that with a small "bs" and lengthy "wbs" processing, the system may NOT perform "wbs" processing at every block.
- The same applies to "wnt" processing for all but the last increment in an array or an nD experiment: with a very small "bs" or in arrays such as "nt=1,1,1,1,1,1,..." the system may not perform "wnt" processing for every increment / FID.
- "wnt" processing is ALWAYS done at the completion of the last FID in an experiment - this call is queued, if necessary. This is then followed by "wexp" processing.
- "wexp" processing is queued if necessary and is always performed, UNLESS an experiment is terminated by an error or an abort ("wexp" is then followed by "wdone" processing).
- In the case of errors, "werr" processing is done - "werr" is queued if necessary (similar to "wexp" processing).

Thanks to Dan Iverson (Varian Palo Alto) for providing information for this article.

[Agilent MR News 2005-08-11]

2006-10-26:

ALLOWED "sw" SETTINGS VS. SPECTROMETER ARCHITECTURE:

The vast majority of the numeric acquisition parameters have a finite range of allowed values that is determined by the spectrometer hardware. Examples:

- frequency parameters can only assume values in the range of the associated synthesizer (PTS, offset generator, decoupler modulator), and with a "granularity" (i.e., discrete values with a defined step size) that is also determined by that device;
- all time-related parameters (delays, RF and gradient pulse durations) can be set
  - to zero,
  - to the minimum duration / time event that the pulse programmer or controller board can perform (100 nsec on UNITY INOVA and DirectDrive architectures, 200 nsec on older systems),
  - above that to time durations given by the timing resolution of the pulse programmer / controller board (100 nsec on GEMINI/GEMINI 2000, the MERCURY family and early VXR-S, 25 nsec on UNITY and UNITYplus systems, 12.5 nsec on the UNITY INOVA and DirectDrive architectures),
  - up to a maximum that is either given by the software driving the pulse programmer or controller board (above 1 hour) or by some safety limit (pulses are typically restricted to values up to about 8.2 msec)

These restrictions are easy to understand, and they pose virtually no problems when moving parameters between architectures: the most prominent effect is that the difference in the timing resolution or the frequency step size may cause round-off effects when moving parameters from a newer or research instrument to an older or a routine system.

The parameters for the spectral window - "sw" for the direct observe dimension, "sw1", "sw2", etc. for the indirect dimensions in nD experiments - are different from this, and somewhat harder to rationalize: while the above parameter types feature equally spaced values, on traditional spectrometer architectures (prior to DirectDrive) "sw" and its nD equivalents have a set of allowed values that is given by the INVERSE dwell time (the time between sampling points) - and as a time event, dwell times are subject to the same limits as other time events, see above. The maximum spectral window that a pulse programmer can perform in theory is given by the inverse of the shortest executable delay (10 or 5 MHz); beyond that, in the observe dimension there are restrictions in the receiver / filter bandwidth which limit "sw" to 5 MHz with wideband receivers on UNITY INOVA, or to 500 KHz with the standard UNITY INOVA receiver, less on older and routine NMR spectrometers. There is no receiver involved in the indirect dimensions, therefore these latter restrictions don't apply there.

The STEP SIZE is very small at the lower end, but growing substantially towards the top end of the "sw" range. Here's an excerpt from the list of possible values and their applicability (WB = wideband receiver):

Dwell Time	"sw"	Applicability
0.2000 usec	5000000.0 Hz	UNITY INOVA (WB only)
0.2125 usec	4705882.4 Hz	UNITY INOVA (WB only)
0.2250 usec	4444444.4 Hz	UNITY INOVA (WB only)
0.2375 usec	4210526.3 Hz	UNITY INOVA (WB only)
0.2500 usec	4000000.0 Hz	UNITY INOVA (WB only)
...	...	
2.0000 usec	500000.0 Hz	UNITY INOVA
2.0125 usec	496894.4 Hz	UNITY INOVA
2.0250 usec	493827.2 Hz	UNITY INOVA
2.0375 usec	490797.5 Hz	UNITY INOVA
2.0500 usec	487804.9 Hz	UNITY INOVA
...	...	
10.0000 usec	100000.0 Hz	UNITY INOVA, UNITYplus, MERCURY
10.0125 usec	99875.2 Hz	UNITY INOVA
10.0250 usec	99750.6 Hz	UNITY INOVA, UNITYplus
10.0375 usec	99626.4 Hz	UNITY INOVA
10.0500 usec	99502.5 Hz	UNITY INOVA, UNITYplus
10.0625 usec	99378.9 Hz	UNITY INOVA
10.0750 usec	99255.6 Hz	UNITY INOVA, UNITYplus
10.0875 usec	99132.6 Hz	UNITY INOVA
10.1000 usec	99009.9 Hz	UNITY INOVA, UNITYplus, MERCURY
...	...	
20.0000 usec	50000.0 Hz	UNITY INOVA, UNITYplus, MERCURY

20.0125 usec	49968.8 Hz	UNITY INOVA
20.0250 usec	49937.6 Hz	UNITY INOVA, UNITYplus
20.0375 usec	49906.4 Hz	UNITY INOVA
20.0500 usec	49875.3 Hz	UNITY INOVA, UNITYplus
20.0625 usec	49844.2 Hz	UNITY INOVA
20.0750 usec	49813.2 Hz	UNITY INOVA, UNITYplus
20.0875 usec	49782.2 Hz	UNITY INOVA
20.1000 usec	49751.2 Hz	UNITY INOVA, UNITYplus, MERCURY
...	...	
200.0000 usec	5000.000 Hz	UNITY INOVA, UNITYplus, MERCURY
200.0125 usec	4999.688 Hz	UNITY INOVA
200.0250 usec	4999.375 Hz	UNITY INOVA, UNITYplus
200.0375 usec	4999.063 Hz	UNITY INOVA
200.0500 usec	4998.750 Hz	UNITY INOVA, UNITYplus
200.0625 usec	4998.438 Hz	UNITY INOVA
200.0750 usec	4998.126 Hz	UNITY INOVA, UNITYplus
200.0875 usec	4997.813 Hz	UNITY INOVA
200.1000 usec	4997.501 Hz	UNITY INOVA, UNITYplus, MERCURY
...	...	
1000.0000 usec	1000.000 Hz	UNITY INOVA, UNITYplus, MERCURY
1000.0125 usec	999.988 Hz	UNITY INOVA
1000.0250 usec	999.975 Hz	UNITY INOVA, UNITYplus
1000.0375 usec	999.963 Hz	UNITY INOVA
1000.0500 usec	999.950 Hz	UNITY INOVA, UNITYplus
1000.0625 usec	999.938 Hz	UNITY INOVA
1000.0750 usec	999.925 Hz	UNITY INOVA, UNITYplus
1000.0875 usec	999.913 Hz	UNITY INOVA
1000.1000 usec	999.900 Hz	UNITY INOVA, UNITYplus, MERCURY
...	...	

Below "sw" values of 1000 Hz, the step size is below 0.1 Hz even on MERCURY (MERCURY-Vx, MERCURYplus) systems. See also Agilent MR News 1998-01-30 for information on how "sw", "sw1", etc. are stored in VnmrJ parameter files.

The use of DSP ("dsp='r'" on the UNITY INOVA, or "dsp='i'" on any system) complicates the issue insofar as now, the final spectral window ("sw") is one of the executable values (see the above table) DIVIDED BY THE OVERSAMPLING FACTOR, whereby the oversampling factor ("oversamp") is always an integer. Typically (in liquids NMR), people will run with the maximum sampling rate (400 KHz with "dsp='r'", 500 KHz with "dsp='i'" on UNITY INOVA, 100 KHz on the other systems), unless "sw" is very small. This means that in practice, with DSP, people are using a VERY SMALL SUBSET of the values for "sw" that would be available without downsampling. However, the advantages of DSP (better filter properties, better baselines, far less intensity distortions within the DSP filter passband) outweigh the restrictions in the possible "sw" values.

Now, with the DirectDigital receiver (DDR) architecture of our current NMR spectrometers, we are entering a totally different gamut: here, we are using a fixed sampling rate of 80 MHz, producing a data stream that is first downsampled to 10, 5, or 2.5 MHz. Currently, we are only using the option of downsampling to 2.5 MHz (unless you want to acquire spectral windows of 5 or 10 MHz): the 2.5 MHz intermediate spectral window is (currently) to be taken as a fixed value, from which the observed "sw" values are derived by applying one or two optional stages of downsampling, i.e., one or two divisions by an integer. This leads to a totally different set of values, compared to the UNITY INOVA without (or with) DSP:

sw	downsampling
500000.0 Hz	5
416666.7 Hz	6
357142.9 Hz	7
312500.0 Hz	8
277777.8 Hz	9
250000.0 Hz	10
...	

This table is valid up to downsampling rates of 25 (i.e., an "sw" of 100 KHz); beyond that (i.e., at smaller spectral windows) there are further restrictions because the digital filtering is then performed in two stages. A discussion of how the downsampling / digital filtering is distributed onto these two stages is beyond the scope of this article: this is handled inside the acquisition software and is currently not accessible to the user. Empirically, you will find that the step size of the allowed "sw" values decreases with "sw":

52083.3, 50000.0, 48076.9, ...
25510.2, 25000.0, 24509.8, ...

10080.6, 10000.0, 9920.6, ...
4006.4, 3980.9, 3955.7, ...
2510.0, 2500.0, 2490.0, ...
1001.6, 1000.0, 998.4, ...

Keep in mind that these spectral windows are associated with appropriate, accurate digital filtering with sharp filter cutoff (i.e., no folded signals) while maintaining excellent baseline characteristics. You cannot compare this set of values with the first list above for instruments without the use of DSP, for several reasons:

- What really matters on conventional systems (without DSP) is the step size in the analog filter settings ("fb") - and "fb" typically has a much larger step size than "sw" (particularly at the lower end), while with the DirectDigital receiver, the digital filters are ALWAYS and ACCURATELY matched to the actual "sw" value.
- Related to that: due to the excellent filter cutoff characteristics, you don't need to consider cases where you would need to fine-tune "sw" to move folded signals to non-overlapping positions!

The settability of "sw" with the DirectDigital receiver is more than adequate for most or all applications. Considering that the above value set comes with all the advantages of the new architecture (see above and Agilent MR News 2005-03-14), you certainly would not want to revert to the largely unnecessary extra "sw" flexibility of the conventional spectrometer architectures!

[Agilent MR News 2006-10-26]

A POTENTIAL TRAP WITH "go" AFTER "df" OR "wft":

Whenever you call "rtp" to retrieve parameters, VnmrJ runs the macro "fixpar" which not only checks whether the parameter set is complete, but also ensures that the parameter limits / allowed values, as well as the parameter settings are compatible with the hardware for which VnmrJ is configured (note: this also applies to stand-alone workstations, see Agilent MR News 2003-08-18 for more information). This should avoid situations where you type "go" and the acquisition software then tries talking to non-existent hardware - at least in first approximation: after an "rtp" on imported parameter sets it is usually a good idea first to call "dps" to see whether the parameter settings make sense.

The same mechanism also plays when you recall a data set using "rt". At first, this may be viewed as irritating behavior, because after the "rt" command, the displayed parameters may NOT agree with the ones that are associated with the recalled data set - on the other hand, it DOES enable you to re-acquire a given spectrum, even if the original data were acquired on different spectrometer hardware (provided of course you have the appropriate pulse sequence, and provided that pulse sequence is compatible with your spectrometer configuration). The above (possible) divergence between the parameter settings and the ones associated with the original data is resolved immediately when you perform "df", "ft" or "wft" (or the equivalent nD commands), which copies the "processed" parameter tree into the "current" one: as soon as you look at acquired data, VnmrJ also displays the parameters that are associated with the data set that is being processed.

While these mechanisms appear to fulfill both the hardware compliance as well as GLP-related requirements (the latter by ensuring that parameters associated with a given data set are not easily manipulated), there is a potential trap in this functionality: after "df", "ft" and related commands you can NOT assume that the parameter settings are still compliant with your spectrometer! One example is discussed in the article below. Conclusion: if you want to re-acquire an imported spectrum, make sure you call "fixpar" prior to starting the experiment, and maybe double-check the final settings by calling "dps". On top of that: even if you have called "fixpar" explicitly, if you ever call "df" or "ft" etc. AGAIN prior to starting the experiment, you would also need to repeat the "fixpar" call!

[Agilent MR News 2006-10-26]

A MYSTERIOUS REFERENCING ISSUE AND THE "sw" PARAMETER:

One user recently reported having acquired two "identical" spectra in sequence, on the same sample, and referenced them both to the same signal - and yet, there was a referencing error of up to about 0.1 ppm with the low field signals. "sfrq" was identical, "sw" was slightly different, but that was compensated for by the referencing parameters ("rfl" / "rfp", "reffrq"); what happened?

The solution is in the article above: a 1D FID from a UNITY INOVA was imported onto a new spectrometer with DirectDrive architecture. Upon doing "rt", the software called "fixpar", which performed "sw=sw" to have the "sw" parameter re-calculated to a "legal" value for the DirectDigital receiver - but then the user called "wft" to re-check the original data, which restored the acquisition parameters used in the UNITY INOVA. One of the two spectra was "reacquired" by typing "go" after an "ft" - and then, "sw" had an "illegal" value that was rounded off to the nearest possible value by the acquisition software. The real issue with this is that the newly acquired spectrum now included an incorrect "sw" value in its parameter set: "go" does NOT correct "impossible" settings of "sw" (we certainly don't want "go" to alter "sw" / the dwell time, and with it the acquisition time, "at").

The conclusion is given in the previous article: when repeating experiments based on imported AND RETRANSFORMED data sets it is IMPERATIVE that you call "fixpar" prior to starting the experiment with "go" - or alternatively, use "rtp" rather than "rt", or DON'T retransform! This same issue could also arise upon importing spectra from a UNITY INOVA to an older or routine spectrometer with a lower timing resolution.

[Agilent MR News 2006-10-26]

2006-12-05:

CHANGES TO CONFIGURATION PARAMETERS:

In VnmrJ, configuration settings (i.e., "systemglobal" parameters that refer to and describe the hardware configuration of your system) are typically entered by the installer, by selecting "Edit" -> "System settings..." -> "System config" (or by typing "config" on the command line). Thereafter, there is rarely a need to alter or update these parameters, with a few exceptions:

- from time to time, the system lock frequency ("lockfreq") may need to be altered in order to compensate for magnet drift;
- a new accessory is added, or in a laboratory with multiple instruments an accessory is moved between instruments;
- a user or engineer may wish to disable an accessory while it is not used temporarily, or for testing purposes;
- on a processing workstation you may want to adjust the VnmrJ configuration parameters to match those of the system on which the bulk of the processed data were acquired (see also Agilent MR News 2003-08-18 and Agilent MR News 2000-03-25).

As these parameters are system-specific and apply to all accounts / users of the software, we only allow the system administrator (typically vnmr1) to alter such values permanently. This is controlled by the permissions of the associated parameter file, "/vnmr/conpar": vnmr1 has write permission, and for vnmr1 the above menu selection opens the configuration utility "config"; other users only see a checkbox "Display configuration" which (if checked) will cause the pop-up utility to display the hardware configuration in the text output pane ("Process" -> "Text output") once the utility is closed. Changing and viewing the system configuration through "config" (or via the main menu) is safe; whenever possible, this should be the ONLY mechanism in use for altering configuration parameters.

There are situations when the above mechanism is inconvenient (convenience and safety often don't go together!) - e.g., there may be situations when the system administrator is not around, but the operator needs to adjust the lock frequency, or any of the other examples above. Fortunately, the presence of the system administrator (or knowing the administrator password) is NOT required to change configuration parameters:

- any user can directly change "systemglobal" parameters, e.g., by typing
  ```
  lockfreq=new_value
  ```
  on the command line (this assumes that you know the name of the associated parameter and its "legal" values);
- on the other hand, also users other than vnmr1 can launch the interactive configuration utility, by typing "config" on the command line, and change ANY system configuration setting, even without knowing the name of the associated parameter.

Of course (and for good reasons) both these mechanisms will NOT alter the contents of "/vnmr/conpar" (unless the permissions of that file are altered, which is definitely NOT recommended):

- even for vnmr1, ONLY the interactive "config" utility saves changes in "/vnmr/conpar", though vnmr1 COULD use
  ```
  fsave('/vnmr/conpar','systemglobal')
  ```

to save changes from direct parameter entry; we recommend using the "fsave" mechanism ONLY in exceptional cases - the "lockfreq" parameter MAY be one of the few exceptions.
- for users other than vnmr1, ALL "systemglobal" changes are valid ONLY for the duration of the current VnmrJ (or VNMR) session, i.e., when you exit VnmrJ and later re-start it, the new process will start up by reading the contents of "/vnmr/conpar" (and hence revert to the standard settings). If a user other than vnmr1 uses "config", this is clearly indicated in the lower left corner of the "config" window.

This implies that you may need to repeat the alteration of such parameters for every session. There may be cases where one wishes to ENFORCE local settings for certain "systemglobal" parameters for specific accounts, e.g.:
- if an account is to be used in connection with a specific or a restricted hardware configuration ONLY, or
- if an account is to be used for data processing ONLY (in which case the "system" parameter could be set to "datastation" for that user);

This can be achieved by adding such parameter settings, e.g.,

```
system='datastation'
```

to a local "login" macro (see also Agilent MR News 1998-08-07 and articles referred to therein).

[Agilent MR News 2006-12-05]

CONFIGURATION PARAMETERS, VnmrJ VIEWPORTS, AND BACKGROUND PROCESSING:

There are a couple interesting quirks in relation to local / temporary configuration settings as described in the previous article, namely:
- if you are acquiring in an experiment that is NOT current to one of the VnmrJ viewports, the acquisition process will fork a "background copy" of "/vnmr/bin/Vnmrbg", and that background process will start by reading "/vnmr/conpar", i.e., it will use the default configuration settings.
- similarly, in automation, both the launching of the experiments AND the processing will be ALWAYS performed by such background processes and will therefore use the settings from "/vnmr/conpar". IN PARTICULAR, you can NOT change "traymax" (the "Sample changer" setting in "config") directly (or as user other than vnmr1) and then expect automation to operate on the specified sample changer type! See also Agilent MR News 1999-02-12 for a related note.
- when you are running VnmrJ with multiple open viewports, temporary and permanent changes to global and systemglobal (configuration) parameters ARE propagated from the active to the other viewports. HOWEVER, if you open any NEW viewports, each of these will be driven by a new "Vnmrbg" process that starts by reading "/vnmr/conpar" and will therefore again use the standard settings, i.e., temporary configuration changes are NOT active for viewports that are opened AFTER such parameter changes; see also bug report "viewports.j2101".

All these issues can be avoided by using "config" as vnmr1 to alter system configuration parameters.

[Agilent MR News 2006-12-05]

2007-02-02:

USING "setvalue" ON ARRAYED PARAMETERS:

In the vast majority of cases, VnmrJ parameter values are set using direct parameter entry, e.g.: "d1=2". This works for all user-enterable parameters in the "current" tree (i.e., the parameters that you view and work with in the current VnmrJ workspace / experiment), in the "global" tree (user-specific parameters that are NOT experiment-related and are stored in the user's "~/vnmrsys/global"), as well as in the "systemglobal" tree (stored in "/vnmr/conpar"), although in this last case changes by direct parameter entry are only temporary, see also Agilent MR News 2006-12-05.

Direct parameter entry is subject to a number of restrictions and "side effects", such as
- the entered values must fall into the range of "legal" values as defined in the parameter's limits, either as direct entries (maximum, minimum, step size), or using indirect references to limits stored in "parmax", "parmin", and "parstep", see also Agilent MR News 1998-08-28;
- parameter entry may be restricted by the parameter's internal protection (protection bits), see "man('setprotect')" - e.g., parameters may or may not be arrayable, enterable, etc.;

- there are numerous cases where the change of a parameter causes a macro ("underscore macro", see also Agilent MR News 1999-12-03) to be executed (this is triggered by one of the protection bits) - this typically serves to adjust other parameters which are linked to the one that is changed directly;
- a change to ANY acquisition parameter in the current tree will cause "ct" to be reset to zero;
- there is an entire tree (the "processed" tree, linked with the data in the experiment) that is NOT accessible to direct parameter entry (think of this as a "GLP ruling").

All these restrictions are there for good reasons and are key to VnmrJ's data and parameter handling functionality. However, there are some rare, but legitimate cases where a need to bypass these restrictions arises. One way to bypass any restrictions would be by directly editing a parameter (e.g., using "paramedit" or "paramvi") - however, because the VnmrJ parameter storage format is NOT primarily optimized for visual readability (for details see chapter 5 "Parameters and Data" in the VnmrJ User Programming Manual), this is NOT A RECOMMENDED OPTION (as it opens the possibility to screw a parameter's format, possibly corrupting entire parameter trees) - plus, it would not be applicable to such "special" parameter manipulations inside macros. The command to use here is "setvalue". The syntax for "setvalue" is

```
setvalue('parameter_name',parameter_value)
```

for the "current" tree (where the value can be a string or numeric),

```
setvalue('parameter_name',parameter_value,'tree_name')
```

for changes in other ("processed", "global", "systemglobal") trees, and

```
setvalue('parameter_name',parameter_value,array_index)
```

for arrayed parameters (again with an optional parameter tree argument). The array index can be the index for any of the defined array elements in that parameter, or the first element following the values defined so far: if you want to enter an entire array using "setvalue", you need to enter the values one by one, e.g.:

```
setvalue('d2', 0, 1, 'processed')
setvalue('d2', .1, 2, 'processed')
setvalue('d2', .2, 3, 'processed')
setvalue('d2', .4, 4, 'processed')
...
```

While defined array elements can be changed in any order, UNDEFINED elements must be added IN SEQUENCE, and again one by one. This permits EXPANDING an array by one element at a time. There are two cases which require special treatment:

- if you want to "unarray" a parameter, i.e., change an arrayed value into a non-arrayed value, you need to enter the new value with 0 as array index, such as in

```
setvalue('d2', .1, 0, 'processed')
```

- if you want to make the array size SMALLER, you MUST first eliminate the array using the above method, then enter the ENTIRE NEW array, e.g.:

```
setvalue('d2', .2, 0, 'processed')
setvalue('d2', .4, 2, 'processed')
```

This feature has been omitted in the current manuals - it has been corrected for future versions of the documentation. Just as a reminder: remember that the "setvalue" command will

- NOT check the parameter values against the parameter's built-in limits;
- NOT execute underscore macros: linked parameters are NOT automatically (co-)adjusted (which may lead to conflicting parameter settings: think of a case where a user changes "at", but NOT "np"!);
- if you change a parameter array size, "arraydim" is NOT adjusted;
- if you eliminate a parameter array, neither "arraydim" nor "array" are adjusted

So, "setvalue" should be used wisely; it may sometimes be better

- first to copy the entire "processed" tree to the current one (e.g., using "df", "wft", or "wft2d"), then
- to save "ct" in an temporary parameter:

```
$ct = ct
```

- to use standard parameter entry for proper / clean parameter settings, then
- to restore the value of "ct":

```
setvalue('ct', $ct)
```

- and finally to copy back the entire parameter tree, using

```
groupcopy('current','processed','acquisition')
```

See also Agilent MR News 2006-11-28 for an example (in that article the "$" is actually missing from the "setvalue" command!), or - for more examples - see

the FAQ document on "Handling Arrays in VNMR / VnmrJ" ("faq/vnmr_arrays") in the on-line User Library at

```
http://www.varianinc.com/products/nmr/apps/corner.html
```

Thanks to Maj Hedehus for bringing up this topic, and thanks to Dan Iverson for the information on eliminating arrays using "setvalue".

[Agilent MR News 2007-02-02]

2007-02-26:

DOCUMENTATION AMENDMENT FOR NMR SYSTEMS WITH 1 KWATT AMPLIFIERS:
(by Jim Frye, Varian)

On combined solids & liquids NMR systems with 1 KWatt high power amplifiers, you want to disable these amplifiers when they are not in use (such as when doing liquids NMR). This control is achieved through an optional, global flag parameter "hipwrampenable". For safety reasons, the high power amplifiers are disabled if the parameter "hipwrampenable" does not exist. The current Command and Parameter Reference Manual for VnmrJ 2.1B describes this parameter and its creation using the command

```
create('hipwrampenable', 'flag')
```

However, this instruction is WRONG, in that "hipwrampenable" must be created as a GLOBAL parameter, using

```
create('hipwrampenable', 'flag', 'global')
```

With the first command above would create "hipwrampenable" as LOCAL parameter (in the "current" tree), while the PSG software specifically looks for a global parameter and would therefore IGNORE any local parameter with that same name. As this parameter is optional, there would be no error message - and the high power amplifiers would remain disabled.

An addendum by the editor: There is an additional quirk with the presence of a local parameter "hipwrampenable": if BOTH a local AND a global parameter with that name exist, the parameter HANDLING (display, setting, parameter query using "hipwrampenable?") would refer to the LOCAL parameter (as local parameters have precedence over global ones with the same name), and hence you may THINK you are controlling the amplifier, while indeed the relays remain set as defined by the (setting or absence of the) global parameter. This may be detrimental in two ways:

- if the global parameter does not exist or defines the 1 KWatt amplifiers as disabled, solids NMR experiments may not work as expected;
- worse than that, if the global parameter enables the high power amplifiers, AND there is a local parameter "hipwrampenable", you may type

  ```
  hipwrampenable='nnn'
  ```

  to do liquids NMR, while the amplifiers remain activated through the global parameter, possibly causing serious damage to your liquids probe.

Considering potential detrimental effects of a local parameter overriding and hiding a global parameter with the same name, and considering that you might already have saved parameter sets with a local copy of that parameter, it may be worthwhile considering the addition the following construct to the macro "/vnmr/maclib/fixpar":

```
exists('hipwrampenable','parameter'):$e
if $e then
  destroy('hipwrampenable')
endif
```

This would remove any undesired local copy of this parameter, if present.

The commands "setvalue" and "display" permit setting and viewing (querying) parameters in SPECIFIC trees, irrespective of the presence of parameter duplicates in other trees.

[Agilent MR News 2007-02-26]

2007-10-06:

THE "probeConnect" PARAMETER (by Christine Hofstetter, Varian):

The Varian NMR System (DirectDrive) console now supports a new parameter called "probeConnect" that can be used in lieu of "rfchannel". The latter maps logical RF channels in the pulse sequence (as defined through "tn", "dn", "dn2", etc.) to physical RF channels (channel 1, channel 2, etc.) in the console, but with this parameter alone - e.g., with a setting of

```
rfchannel = '1324'
```

the PSG software has no "notion" of how the RF channels are connected with the NMR probe, and it is up to the operator to ensure a proper correspondence

between the nucleus settings in the local parameters, "rfchannel", and the probe connectivity. "rfchannel" is a local parameter that may need to have pulse sequence specific settings in the case of sequences with diverging RF channel assignments / usage - plus, the "rfchannel" setting may need to be altered to accommodate specific spectrometer / RF channel configurations.

The new parameter "probeConnect" is more user friendly, more versatile, and yet easier to understand. It simply and unambiguously maps nuclei (specified via "tn", "dn", "dn2", etc.) to physical RF channels, avoiding confusion and unnecessary complexity: with "probeConnect" you define how the probe RF ports are connected to the RF connectors of the preamplifier box. An example of a "probeConnect" setting is

```
probeConnect = 'H1 C13 F19 N15'
```

If your system is a four-channel system that is configured as HB (high band) - LB (low band) - HB - LB, but you would like to run a BioPack sequence that is written such that

```
tn='H1' dn='C13' dn2='N15'
```

Using the above setting for "probeConnect" will cause the 15N pulses to be created on RF channel 4, so that even though the pulse sequence is written with 15N on channel 3, channel 4 is used for 15N. If you then switch to an experiment that has

```
tn='H1' dn='N15' dn2='C13'
```

the RF channels will automatically be reassigned accordingly. Since "probeConnect" is a global parameter, the entire BioPack autocalibration routine can be run with 15N cabled on channel 4 through the above "probeConnect" setting.

Another useful feature of "probeConnect" is that is allows you to observe nuclei on the third or fourth RF channel, provided the extra hardware is installed on the system. If a three-channel system has

```
probeConnect = 'H1 C13 P31'
```

and "tn='P31'", the 31P pulses will be created on channel 3, and the signal will be observed through the third channel preamplifier. In this case, "probeConnect" could also be set to

```
probeConnect = 'H1 X P31'
```

where X is a "wild card" value that the software will set to any lowband nucleus (other than P31) in the parameter set. The possibilities are endless. Only ONE wild card token "X" may be used in the "probeConnect" value: multiple "X" tokens will generate an error message.

The number of "tokens" in the string MUST be equal to the number of RF channels. Parameter sets must not contain nuclei that are not part of the "probeConnect" value, unless the "wild card" (X) is present in the string. With the "X" token, one low-band nucleus that is not listed in the string is permitted. For example, a five-channel system has

```
probeConnect = 'H1 C13 F19 N15 H2'
```

and the parameter set has "tn='Si29'": in this case the system does not know how to deal with this nucleus, and the PSG software will abort with an error message indicating that the "rfchannel" index is not properly mapped. However, with a setting of

```
probeConnect = 'H1 X F19 N15 H2'
```

the system will run with Si29 on channel 2.

Unlike "rfchannel", "probeConnect" is a global parameter, so does not have to be re-created for every experiment. To create "probeConnect", type

```
create('probeConnect','string','global')
```

on the VnmrJ command line. Note that as soon as "probeConnect" exists, it overrides any "rfchannel" setting in the local parameters (see the note below for a potential issue with VnmrJ 2.1B).

If you ever want to stop using "probeConnect", you can destroy the parameter with the command

```
destroy('probeConnect','global').
```

On the other hand, you may want to consider adding "probeConnect" to your "saveglobal" parameter, such that its setting is logged / archived together with the data (in a local parameter "probeConnect_").

[Agilent MR News 2007-10-06]

USING THE "probeConnect" PARAMETER UNDER VnmrJ 2.1B:

If you are running VnmrJ 2.1B and want to use "probeConnect", you will also need to create an additional parameter "preAmpConfig", with

```
create('preAmpConfig','string','global')
```

The values for "preAmpConfig" are "H" for a high band preamplifier, "L" for a low band preamplifier, and "X" for no preamplifier on a given channel. For

example, a three-channel spectrometer with 1 HB preamp and 1 LB preamplifier (on channels 1 and 2) would use

```
preAmpConfig='HLX'
```

(no spaces). The "preAmpConfig" parameter is not needed in VnmrJ 2.2C.

As mentioned in the article above, starting with VnmrJ 2.2A, "probeConnect" overrides any "rfchannel" setting (if present). "probeConnect" was already supported in VnmrJ 2.1B - but that implementation was somewhat incomplete: in VnmrJ 2.1B you should avoid having defined BOTH "probeConnect" (global) and "rfchannel" (local) at the same time - the results would be unpredictable at best.

What is the best way to avoid the conflict between "probeConnect" and "rfchannel" in VnmrJ 2.1B? At first, one might think of adding a construct such as

```
exists('probeConnect','parameter','global'):$e
if $e then
  exists('rfchannel','parameter'):$e1
  if $e1 then destroy('rfchannel') endif
endif
```

to the "fixpar" (or the "userfixpar") macro, such that in the presence of "probeConnect" any "rfchannel" parameter is removed from retrieved parameter sets. However, this may cause certain pulse sequence macros to fail or abort when trying to retrieve "rfchannel" using "rtv" / "psgset", or when trying to set "rfchannel" to specific values. A better idea is to leave "rfchannel" alone AT FIRST, but rather to add the above construct to the "go" macro (e.g., above the line reading "if (traymax=96) then"), or by creating a macro "usergo" with the above construct. This way, setup macros can work as they used to (manipulating "rfchannel", if / as necessary on systems without the "probeConnect" parameter) and don't need to be altered, and if "probeConnect" exists, "rfchannel", will be deleted in the very last moment, just before the pulse sequence is launched.

Again, none of this is necessary with VnmrJ 2.2C.

[Agilent MR News 2007-10-06]

2009-07-17:

UNDOING VnmrJ COMMANDS / ACTIONS?

Many desktop productivity programs not only remember a (possibly large) number of past commands and user actions - they also permit moving "back in time" through this history and undoing an entire series of such actions with key combinations such as [Ctrl-z], often also re-doing actions just undone, with key combinations such as [Ctrl-y] (obviously, as such actions typically depend on the result of previous actions, one can only move back in the "event stack" sequentially). This is possible because the actions in the command stack are from a finite set of well-defined user actions (menu selections, mouse actions, etc.), and often the object of such actions is also well-defined and permits keeping a parallel stack of versions, reflecting the state of the data at past intermediate stages (so you don't need "inverse" versions of each possible action to move back in the history).

VnmrJ has a command stack / history, too - however, this stack only serves to memorize and possibly re-execute past command lines (with modifications by command line editing prior to the new call, as appropriate), but NOT UNDOING any actions: unlike in office productivity suites, VnmrJ commands / actions can be vast sets of sequential / nested macros of arbitrary complexity, and the object of such commands is of arbitrary size (e.g., 3D images, nD NMR data), hence undoing even just a single, generic command line or menu action is virtually impossible.

[Agilent MR News 2009-07-17]

UNDOING PARAMETER CHANGES?

Along the lines of the article above, VnmrJ does NOT keep backups of a given parameter, neither of its values, nor its properties such as parameter type, group, limits, protection bits, enumerals (enumerated allowed parameter values), etc. - in other words: once you change a parameter value, or, e.g., its limits, any previous values (arrayed or not) or properties are trashed immediately. This is rarely a problem at all because

- parameter changes typically happen in the "current" parameter tree, and in the case of acquisition parameters, re-processing an acquired data set will automatically bring back the original parameter value from the

"processed" (shadow) parameter tree, along with all of the parameter's built-in properties.
- when setting up parameters for an acquisition, one cannot resort to a "processed" parameter tree for recovery - but you will typically start off a stored parameter or data set, and so in the worst case you only need to redo the few steps from where you recall stored parameters.

In all these years we have not had complaints about a missing parameter (value or property) history: typical "incidents" are simply erroneous parameter entries (e.g., a typing error), and then you simply correct the last command line to re-enter the proper value. If you detect an erroneous parameter value in a stored parameter set, you can retrieve that parameter set, correct the value and re-save the parameters:

```
rtp('/vnmr/parlib/Ghsqc') d1=1 svp(file)
```

The same mechanism can be used to correct parameter limits, protection bits etc. in saved parameter sets - here, you would replace the parameter entry in the above example with the appropriate "setlimit", "setgroup", "setprotect", "setenumeral", etc. command (see the Command and Parameter Reference Manual for information on these commands).

That presumes that you know what value or property a given parameter should have - in the case of parameter values this is usually trivial (otherwise you would not detect that the current value is wrong). However, properties OTHER THAN the active parameter value may be somewhat non-trivial to restore - e.g., enumerals or parameter limits. In both these cases you will only find out about erroneous settings when you can't enter the desired and valid parameter value(s). You can use "display('parameter_name')" to see the current limits or enumerals, and

- you can remove the enumerals (i.e., allowed flag characters or string values) using the "setenumeral" command - but then the parameter entry will no longer be restricted to "legal" values
- erroneous (e.g., too restrictive) parameter limits can NOT simple be "removed", as limits are an inherent part of the parameter definition (while most parameters don't have enumerals).

If you have lost the enumerals for a given parameter and don't know what they should be, you will need to find a parameter set with an equivalent parameter that still has the enumerals, use "display('parameter_name')" to see them, then use

```
rtp('/vnmr/parlib/Ghsqc') setenumeral(...) svp(file)
```

to correct them in the faulty parameter set. For restoring parameter limits see the note below.

Note that this article and the one below are not really meant to present a concrete recipe for specific issues (even though they were triggered by actual inquiries from users), but are rather meant to present generic information on VnmrJ parameter handling and related issues.

[Agilent MR News 2009-07-17]

RESTORING PARAMETER LIMITS:

In the context of the above articles, one tricky case with a potentially larger scope was reported recently by a user: assume you have changed the limits of a parameter in one or several parameter sets - take for instance "oversamp" on a UNITY INOVA - and you want to revert this change. As stated in the article above, you cannot simply "remove" the parameter limits: at most, you could set the numeric limits to a very large range, e.g.:

```
setlimit('oversamp',32767,0,1)
```

("oversamp" is a positive integer) - this avoids any practical restriction, but also permits entering values which are "illegal" or without practical meaning, and which that could potentially cause havoc. You need to find out what the proper range for that parameter is. In the case of a standard parameter this range may be found in the description in the Command and Parameter Reference manual, or you can quickly screen a parameter directory in "/vnmr" for the proper values, e.g.:

```
cd /vnmr/parlib
grep '^oversamp ' *.par/procpar
```

which prints the header line for the parameter "oversamp" in all of the specified parameter sets (see chapter 5 "Parameters and Data" in the "User Programming Manual" for details on the VnmrJ / VNMR parameter format). Note the syntax in the above "grep" call:

- the "^" indicates the start of the line, so we don't extract occurrences of the specified parameter name inside a "dg" or "ap" template),
- the extra blank after the parameter name excludes other parameter names

   that start with the same string; in saved parameter sets ("*.par"),
 - for stored FIDs ("*.fid") and parameter sets ("*.par"), the actual
   parameters are contained in a subfile "procpar".
Here, this will yield output such as

```
        Apt.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        aptune.par/procpar:oversamp 7 1 20 0 1 2 1 9 1 64
        Carbon.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Cigar2j3j.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Cosy.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        COSY.par/procpar:oversamp 7 1 20 0 1 2 1 9 1 64
        cryo_burnin.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Dbppste_cc.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Dbppsteinept.par/procpar:oversamp 7 1 66 0 1 2 1 9 1 64
        Dbppste.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Default.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        Dept.par/procpar:oversamp 7 1 68 0 1 2 1 9 1 64
        ...
```

The limits (maximum, minimum, step size) are the numeric (integer) tokens 3, 4, and 5 after the parameter name, i.e., the minimum is always 0, the step size is 1, and the maximum is either 20 (in this case for "dsp='i'") or 68 (for "dsp='r'").

  If your system has been "infected" with erroneous parameter limits (i.e., you have altered the limits in numerous standard parameter sets), that method may not give clear results - then you may need to look this up on a system or a directory without these alterations (in the very worst case you could do a temporary install of VnmrJ from the original media into a new directory, in order to see "vanilla" parameter definitions).

  A recipe for fixing single parameter sets was given in the article above. For the case of "general parameter contamination" it would be very tedious to locate and correct all relevant stored parameter sets - in these cases it is better to tackle this with a different approach:

 - you can correct parameters upon retrieving data, either by adding a
   corrective construct such as

```
        exists('oversamp','parameter'):$e
        if $e then
          setlimit('oversamp',68,0,1)
        endif
```

   or, in this case, better even a construct such as

```
        exists('oversamp','parameter'):$e
        if $e then
          if dsp='r' then
            setlimit('oversamp',68,0,1)
          else
            setlimit('oversamp',20,0,1)
          endif
          on('oversamp'):$oson
          if $oson then
            oversamp=oversamp
          endif
        endif
```

   to the macro "/vnmr/maclib/fixpar", or - far better - to an independent macro "~/vnmrsys/maclib/userfixpar" which is automatically invoked by "/vnmr/maclib/fixpar". This way, you don't need to modify a standard VnmrJ macro. To cover all users, "userfixpar" can be placed in "/vnmr/maclib", provided no user has a local "userfixpar" macro.
 - alternatively (or even in addition) to the above, you could add a parameter check before the parameter is actually USED. "oversamp" is an acquisition parameter, i.e., it takes effect upon "go", "ga", or "au" (all of which are one and the same macro in "/vnmr/maclib"). But also here it is better NOT to alter the standard VnmrJ macro: instead, the above construct can be placed in a separate macro "usergo" which is automatically invoked if present. Again, "usergo" can be placed in every user's "~/vnmrsys/maclib", or in "/vnmr/maclib", provided there are no local copies which would override the global version. Or, if SOME users already have a local version of "usergo", they would also need to add the above construct, for complete coverage.

Note that calling "setlimit" JUST sets the parameter limits, but does NOT re-check the parameter value by itself - therefore, if a parameter is active, the value should be re-entered in order to have the value(s) checked against the modified parameter limits.

As an alternative to looking up existing parameter definitions in a VnmrJ directory, you can also check where in "/vnmr/maclib" such a parameter is created, e.g.:

```
cd /vnmr/maclib
grep oversamp * | grep create
```

which for this parameter points us to the "paros" macro, see below.

As mentioned above, this is NOT necessarily a recipe for you to follow as is: it is rather meant to explain general VnmrJ parameter handling principles (and the example does not apply to systems with DirectDrive architecture). On the other hand, if you have indeed "messed around" with parameter limits (rather than just setting parameter values), then the above becomes real: the point is that VnmrJ typically DOES set up its parameters with the appropriate limits ("oversamp", to stay with this example, is created when entering "dsp", which calls "_dsp", which in turn calls "paros" - and the "paros" macro creates "oversamp", if necessary:

```
exists('oversamp','parameter'):$e
if (not $e) then
  create('oversamp','integer')
  setlimit('oversamp',20,0,1)
  setprotect('oversamp','on',9)
  setgroup('oversamp','acquisition')
  oversamp=1
  ...
endif
```

However, if "oversamp" exists, the parameter is NOT altered (in order to preserve the current value), and at the same time its limits are NOT checked, updated or restored, and similarly, subsequent macros may check the value of "oversamp", but they should NOT be expected to check its limits, protection bits, etc.! This is pretty much a general principle in most of VnmrJ: "fixpar" (called automatically upon retrieving parameters or data) checks for the existence and the value of a number of parameters, and it will create them if necessary - but if a parameter exists, its limits are NOT checked, even though a MAGICAL tool to look at parameter limits ("getlimit" command) exists.

Talking about parameter limits: remember that parameter limits are ONLY active for DIRECT parameter entry ("parameter_name=value") - limits are BYPASSED when using "setvalue", or when setting parameter values through return argument mechanism, i.e.,

```
trunc($value,1,0):oversamp
```

will happily set "oversamp" to arbitrary values, no matter what the limits are: in such cases it is often a good idea to follow up with a parameter re-entry in order to have values checked against parameter limits:

```
trunc($value,1,0):oversamp
oversamp=oversamp
```

Also "setvalue" does NOT check parameter limits.

At a first glance, it makes more sense to check parameter properties upon retrieval (such that users realize what the parameter limits are BEFORE typing "go") - on the other hand, for crucial parameters, a check upon "go" (e.g., through "usergo") may be a valid and justified approach, too, given the existence of possible limit bypasses.

Finally, just to complete the example selected here, the above segment from "paros" indicates that a safe default limit setting for "oversamp" should be

```
setlimit('oversamp',20,0,1)
```

[Agilent MR News 2009-07-17]

2009-08-19:

PARAMETER HIERARCHY IN VnmrJ - REMINDER AND FOLLOW-UP:

From a MAGICAL (command line and macro) point-of-view, VnmrJ uses a well defined parameter organization - parameters are organized in trees:

- the "current" tree (the working parameters in the current experiment / VnmrJ workspace, stored in the the file "curpar")
- the "processed" tree (the parameters archived with an acquired data set, normally not handled directly by the user, stored in the file "procpar" in the VnmrJ experiment or with the archived data set)
- the "global" tree ("~/vnmrsys/global"), holding parameters that are user- but NOT experiment-specific
- the "systemglobal" tree ("/vnmr/conpar"), holding system-specific parameter values that can only be set and changed permanently by vnmr1 / the VnmrJ administrator.

In MAGICAL, these parameter trees are handled with a clear hierarchy, i.e., MAGICAL knows how to resolve occurrences of multiple instances of the same parameter name in several of these trees:

- for standard parameter handling, the hierarchy is
  current > global > systemglobal
  i.e., a parameter in the "current" tree will override any "global" or "systemglobal" parameter with the same name, and a "global" parameter will override a "systemglobal" parameter with the same name.
- parameters in the "processed" tree can only be manipulated by special commands, such as "setvalue" and related utilities, and apart from that, this tree is built, read and maintained transparently by VnmrJ commands such as "go", "df", "wft" and others - the "processed" tree is primarily meant to be used for GLP and for VnmrJ's "internal bookkeeping".

All this is explained in more detail in the FAQ document "VnmrJ / VNMR Parameter Handling" to be found in our "User Pages" at
http://www.varianinc.com/products/nmr/apps/corner.html
The associated ASCII file "faq/vnmr_parameters" can be downloaded from the User Library e-mail responder as explained in the article above. This document also includes a set of related articles from past issues of Agilent MR News.

These sources also discuss the specifics of the various parameter GROUPS ("acquisition", "processing", "display", etc.) from the point-of-view of VnmrJ parameter handling in parameter setup and data processing.

One aspect has not been covered in the sources / articles mentioned above: in ACQUISITION (VnmrJ pulse sequences, PSG software in general), parameters are (typically, at least) NOT handled hierarchically. In other words: there MAY be cases where a parameter is looked up in more than one of the active trees (this would be documented and would have to be programmed explicitly, on a SPECIFIC, case-by-case basis), but in general, in acquisition, parameters are fetched / looked up in SPECIFIC parameter trees ONLY. More precisely,

- if a parameter exists and is documented as a configuration (systemglobal) parameter, then you can NOT create a global or local (current) parameter with the same name and assume that this overrides the systemglobal value;
- if an acquisition parameter is a documented to be global, then again you can NOT create a local (current) parameter with the same name and assume that this will override the global value;
- similarly, you can NOT destroy a local parameter (or not create it if it were optional) and create a global parameter with the same name instead: PSG will ONLY look it up in the local (current) parameter tree.

There are several good reasons for doing things this way:

- From a GLP point-of-view, you ideally want ALL acquisition parameters (active / in use) to be archived with an acquired data set; at "go" time, the current tree is copied over to the "processed" tree (and then archived with the FID);
- the global tree is NOT copied over, nor is the "systemglobal" tree;
- an exception to this is through "saveglobal": at "go time", global and systemglobal parameters listed in the "saveglobal" parameter (a global parameter by itself) are transferred to the "processed" tree, whereby an underscore character is appended to the name (e.g.: "loc" gets archived as "loc_", "lockfreq" gets archived as "lockfreq_"). This permits archiving parameters which are NOT experiment-specific or related to the system configuration settings.
- there are a few additional exceptions, such as the systemglobal parameter "Console" (upper case "C") which is copied to a local parameter "console" (lower case "c") when "go", "ga", or "au" are called.
- the main reason for the above exceptions is that you want to be able to recall a parameter or data set and re-acquire the data under the same conditions - but this should of course NOT alter your configuration settings: the lock frequency might have changed, or the existing data might have been acquired on a different system (frequency or architecture).

There might be rare instances of situations where you want to "step outside" of the above scheme - see the article below.

[Agilent MR News 2009-08-19]

CAN PARAMETER VALUES BE ENFORCED?

As explained in the article above, VnmrJ acquisition does NOT apply any parameter "hierarchy", but looks up parameters in specific trees only. Some users felt that on their system they wanted to enforce specific settings for parameters that are described as local (say, "temp=20") and thought of defining a global parameter with that name instead. This idea is severely

flawed, because
 - if the MAGICAL hierarchy WOULD apply in acquisition (it does NOT!), the local parameter would have precedence anyway, overriding the global value;
 - to avoid this you would need to ensure that the local parameter (if it is defined) is destroyed, for the global parameter to be used;
 - unless that global parameter is added to "saveglobal", it would not be archived with the acquired data set, i.e., this opens a potential loophole for GLP non-compliance.

So, are there "legal" (feasible) ways to enforce parameter values? And if so, can that enforced value be made configurable? Indeed, there are such options:
 - the mechanism used most commonly for such tasks is to add a construct such as

```
exists('xyz','parameter'):$e
if $e then xyz='some_value' endif
```

   to a macro "userfixpar" ("/vnmr/maclib/userfixpar" for all users, or "~/vnmrsys/maclib/userfixpar" on a per-user basis, whereby the latter will override "/vnmr/maclib/userfixpar", if defined). With this, whenever a parameter set is recalled, that parameter (if defined) will be set to the given value. Users can then still change that value prior to typing "go".
 - if you do NOT want the users to change that value on a case-by-case basis, you could add the above to a macro "usergo" (again, in "/vnmr/maclib" or local, as desired) - then it will be set at "go" time, and user settings will be ignored.

Especially the second mechanism should be used WITH CAUTION ONLY: think of a system that is used for bio-NMR, and "all samples are always run at the same temperature" - and then, one day you have that odd user who wants to run a temperature array (say, for a kinetics study) - and will end up sweating blood in trying to change the temperature, unaware of the fact that a "usergo" macro enforces a specific "temp" value!

  In the above proposals, the enforced value is configurable ONLY by changing the relevant macro. There are more straightforward ways (from an operator point-of-view) to doing this, e.g., by expanding the above construct to

```
exists('Xyz','parameter','global'):$eg
if $eg then
  exists('xyz','parameter'):$e
  if not ($e) then
    create('xyz','flag')
  endif
  xyz=Xyz
  write('line3','usergo: value "xyz" set to %s from "Xyz", xyz)
endif
```

(assuming the value is a string value). This way, you CAN now create a (user settable) GLOBAL parameter (but with a DIFFERENT name!) from which the default (enforced) value will be set. The extra feedback on line3 avoids confusion by notifying the user / operator of the enforced setting. Still, such solutions need to be clearly documented for all users (as well as for Varian service personnel).

  NOTE: this is merely an example to indicate some of the possibilities that exist in VnmrJ; we do NOT suggest that you handle "temp" this way, nor is this a universal solution - e.g.: if the existing or the enforced value was an array, this would almost certainly cause complications or failures - you would not want to do this with parameters that can be arrayed.

  One option that one might consider in this context is to make a parameter "non-alterable" through "setprotect" in order to prevent changes by the user - however, this is NOT RECOMMENDED (at least not as a general recipe), as
 - there are several mechanisms (such as setting parameters through "setvalue" or by using the parameter as return argument for a VnmrJ command, see Agilent MR News 2009-07-17) that ignore / bypass parameter limits and protection bits;
 - if you protect parameters which some macro wants to alter, this will cause that macro to fail with an error message - such situations may be tricky to debug and fix.

Conclusion: you should not alter a parameter's protection bits unless you know what you are doing!

  Overall, keep in mind that we most likely had good reasons to handle parameter values the way we do - changing that philosophy is your personal responsibility and may affect our ability to provide support! Also, be VERY careful with enforcing parameter values relating to power handling (power values, gradient strengths, amplifier settings, etc.), especially on systems with high power gradient or RF amplifiers, or with Cold probes and/or dealing

sensitive samples: the PSG software in high resolution NMR spectrometers features power and duty cycle checking, but these checks may fail or not be accurate enough, and errors in power handling can have expensive consequences!
[Agilent MR News 2009-08-19]

2009-09-03:

POTENTIAL ISSUE WITH FETCHING PARAMETER VALUES INTO A PULSE SEQUENCE:

In Agilent MR News 2009-08-19 we posted a couple refresher / follow-up notes on VnmrJ parameters, parameter hierarchy and related issues. This information is now available on-line in a "FAQ" document "VnmrJ / VNMR Parameters" via our "Software Corner" at

http://www.varianinc.com/products/nmr/apps/corner.html

Bruce Adams (Merck Research Labs, Boston MA) reminded the editor of a potential issue with fetching parameter values into pulse sequences that was not covered in Agilent MR News so far: in Agilent MR News 2009-08-19 we covered the VnmrJ tree hierarchy for parameters. Besides this hierarchical structure, parameters are (independent of their position in the hierarchy) also divided into distinct parameter GROUPS (as also discussed in the above FAQ document), namely

- acquisition parameters
- processing parameters
- display parameters
- spin simulation parameters
- sample parameters
- "all" / "none" (not used)

Of these groups, the first three are used most frequently, and parameters in these groups are present in both the "processed" and the "current" trees. Technically, the "group membership" is just another parameter property that you can find out about using

```
display('parameter_name')
```

for parameters in the "current" tree, or using

```
display('parameter_name','tree_name')
```

for parameters in other trees - and you can set a parameter's group using the "setgroup" command.

The parameter group mainly determines how and when parameters are moved between VnmrJ parameter trees (see the aforementioned FAQ document). What Bruce Adams is referring to, however, is a peculiarity that may occasionally cause hiccups with using VnmrJ pulse sequences and has to do with differences in the handling of acquisition parameters compared to parameters in other groups (especially processing and display parameters):

- arraying acquisition parameters causes the "array" (acquisition) parameter to be set (for an exception to this see below) - and therefore implies an arrayed acquisition / experiment. You can of course remove a parameter name from the "array" string value, which will leave the arrayed parameter as is, but will prevent the acquisition from performing an arrayed experiment.
- arraying processing or display parameters has no effect on the "array" parameter and the acquisition in general.
- beyond that, processing and display parameters are NOT "seen" by the "getval" PSG utility, yielding a "value not found" error message.

The tricky thing about the last point is that in VnmrJ you can type

```
parameter_name?
```

and you WILL see the parameter's value (assuming it is in the current, global or systemglobal parameter tree, as discussed in Agilent MR News 2009-08-19) - but the pulse sequence will not read its value! In this situation you must remember to check the parameter details with

```
display('parameter_name')
```

as discussed above. This may sound exotic / convoluted - but it is rarely an issue because the default parameter group is "acquisition", i.e., you need to make a conscious decision to alter a new parameter's group, otherwise you are unlikely to run into this problem.

Apart from the effect this has on how a parameter is copied between trees (you definitely would NOT want acquisition parameters to be handled the way processing and display parameters are!), changing a parameter's group is NOT a method to prevent arraying to set the "array" parameter! To achieve this, you have two valid options:

- enter a parameter array, but thereafter REMOVE that parameter's name from the string value in the "array" parameter;
- alternatively and preferably, use "setprotect" to set protection but #8

(value 256) for that parameter, e.g., using
setprotect('parameter_name','on',256)
(see also Agilent MR News 2003-03-03 for an example).
Finally, a lesson to be learned from this: DON'T try avoiding to create new parameters by "abusing" an existing processing or display parameter for a new pulse sequence! In particular, do NOT try using any of the parameters "r1" up to "r7" or "n1" up to "n3", or a spin simulation parameter for acquisition purposes / within a pulse sequence: this simply won't work!

Thanks, Bruce, for suggesting that topic!

[Agilent MR News 2009-09-03]

2011-03-14:

LITTLE KNOWN TOOLS FOR ARRAY HANDLING IN MAGICAL:

In the last issue (Agilent MR News 2011-03-04) we showed how an improper panel definition syntax could cause havoc with the direct entry of parameter arrays in VnmrJ panel widgets - and we gave a recipe and examples for the proper syntax that avoids such potential issues. In this follow-up note we want to show you how you can use the associated MAGICAL syntax element on the VnmrJ command line and in your macros. The editor would like to thank Dan Iverson (Agilent, Santa Clara, CA) for this information.

Traditionally, entering arrayed values could be tedious on the command line; you had to enter the array by listing all values at once, e.g.:

```
pw=5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
```

or in batches:

```
pw=5,6,7,8,9,10,11,12,13,14
pw[11]=15,16,17,18,19,20,21,22,23,24
```

This latter syntax can also be used to alter individual elements or a set of consecutive elements in an array; note that using an array index on the left-hand side of such an assignment statement assumes that the specified array element or the preceding element already exist - otherwise an error will be issued.

At some point we added the "array" command in order to facilitate entering linear arrays: the above array definition can be specified in one command:

```
array('pw',20,5,1)
```

However, the "array" command only works for entering ENTIRE, linear arrays at once, you cannot expand or amend existing arrays - and the direct entry of array values with very small or very big numbers can be tedious, e.g.:

```
gt1=0.0002,0.0004,0.0008,0.0016,0.0032,0.0064,0.00128,...
```

or

```
gt1=2e-4,4e-4,8e-4,16e-4,32e-4,64e-4,128e-4,...
```

An additional MAGICAL utility for entering array values was introduced with VnmrJ 1.1D in the context of VnmrJ panel layout definitions (see Agilent MR News 2011-03-04): the "square bracket syntax" (NOT available with any VNMR software version, nor with VnmrJ 1.1C or earlier releases) facilitates working with arrays. The "core element" can be illustrated with the following sample syntax

```
val=[a,b,c,d]/e
```

yielding the same as

```
val= a/e, b/e, c/e, d/e
```

Here, "val" is the name of a variable (e.g., "d2", or "$value"), "a" .. "d" are numeric values, and "e" is either a numeric value or a numeric parameter. The "[]" can enclose a single value or expression or an array of values or expressions. Any mathematics applied to the "[]" element will be applied individually to each element within the "[]". Some examples.

Entry	Result
nt=[1]	nt=1
nt=[1,2,3]	nt=1,2,3
nt=[1,2,3]*10	nt=10,20,30
nt=22*[2*3,r2+6,trunc(r3)]+2	nt=22*2*3+2,22*(r2+6)+2,22*trunc(r3)+2

As a side effect of this implementation, you can also use "[]" to specify the precedence in expressions, just like "()". For instance,

```
nt=[2*[3+4]]
```

yields "nt=14".

Note that there are limitations if the "[]" element is used as part of a mathematical expression, e.g.:

- only a single "[]" element is allowed; an entry such as

```
nt=[1,2]*[3,4]
```

is not allowed - you would get an error message

        No more than one [--.--]
 - when used in expressions, the "[]" element cannot be mixed with the
   standard comma (,) arraying element, e.g.:
        nt=1,[2,3,4]*10
   is not allowed. You will get the error message
        Cannot combine , with [--.--]
These restrictions only occur if mathematical operators are used and the "[]"
element itself contains a comma. Simply listing multiple "[]" elements, or
combining them with the comma element is OK:

Entry	Result
nt=[1,2],3	nt=1,2,3
nt=[1,2],[3,4]	nt=1,2,3,4

What also does NOT work is the following example:
        pw=8,9,10,11,12 p1=[pw]*2
or
        $val1=8,9,10,11,12 $val2=[$val1]*2
i.e., you can NOT place an arrayed variable inside the square brackets and
expect this to be taken for its arrayed values: in the above case "[pw]" will
simply be interpreted as "[pw[1]]" (and "[$val1]" as "[$val1[1]]"), hence "p1"
and $val2 in these examples will be set to 16. This appears to contradict the
syntax used in VnmrJ panels, as indicated in Agilent MR News 2011-03-04:
        Vnmr command: gt1=[$VALUE]/1e3
However, this is not the same environment, as in the context of VnmrJ panels,
the parameter "$VALUE" is substituted by its value BEFORE the expression is
evaluated, while in macros and on the command line, the square bracket syntax
is resolved / interpreted ALONG WITH (or maybe even before) variable values
are looked up. What WOULD work is
        pw=8,9,10,11,12
        p1=[pw[1],pw[2],pw[3],pw[4],pw[5]]*2
but that obviously defeats the idea of having a simplified syntax for array
handling. Of course, you can still use the "array" utility to REdefine the
complete parameter array rather than manipulating its values - as long as the
array is a linear sequence of numeric values. If you would like a similar
utility for entering / defining arrays with exponentially spaced values, there
is help: the editor has just (re-)posted his "maclib/xarray" contribution
which years ago used to exist inside a larger User Library package; for
details please visit the on-line User Library via our NMR Software Corner at
        http://www.chem.agilent.com/en-US/Support/Pages/default.aspx
                                                [Agilent MR News 2011-03-14]

==========