Open NMR Software Manual

Revision : 1.3/Software Release
Date : 28th July 2013
Author : TBB.

Errors and omissions should be reported to :

AMR Limited
Culham Innovation Centre
D5 Culham Science Centre
Abingdon
Oxon
OX149DB.

Email : tbb@admagres.com
Tel +44 (0) 1865 408375.

1. Introduction and Scope

This manual contains information regarding the OpenNMR software suite, which is used to perform NMR experiments with Advanced Magnetic Resonance Limited NMR Spectrometers.

The OpenNMR software suite consists of the following components :

OpenBox : A simple command line interface to allow data acquisition and control. OpenBox is designed for NMR experts only, and allows simple control of the spectrometers for testing and application development.

ThinBasic : A BASIC programming environment that allows users to write data acquisition scripts, new data processing commands and perform macros with OpenBox.

Codeblocks : A C programming environment that allows users to edit and compile pulse sequences for use with OpenNMR.

Problems with software operation should be reported to tbb@admagres.com, quoting the software version number. This can be found by typing OPENNMR in OpenBox.

2. Installing OpenNMR and Development Components

To use OpenNMR the following components are required :

OpenNMR : OpenNMR_setup.exe provided by AMR Ltd. This should be installed to the default directory (c:\OpenNMR).

Pulse Sequences and Libraries : PulseSequence_setup.exe provided by AMR Ltd. This should be installed to the default directory (C:\PulseSequences).

Code::Blocks : An IDE (integrated development environment) for writing and compiling NMR pulse sequences. This can be found at :

www.codeblocks.org

Version 10.05 is recommended. Installation of Code::Blocks is optional, if users do not want to use the pulse programming features of OpenNMR

ThinBasic : An IDE for developing commands and scripts/macros. This can be found at :

www.thinbasic.com

Version 1.8.9. is recommended (and required for the use of AMRForms). The installation of ThinBasic is not optional for OpenNMR to run correctly, as many commands used by the system are based on ThinBasic.

FTDI USB Drivers : The R3/R4 spectrometers communicate with the host PC via an FTDI USB interface. The drivers can be found at :

www.ftdichip.com

Version 2.08.02 is recommended. Note that the particular driver depends on the OS being used (32 or 64 bit).

OpenNMR Directory Structure

The c:\OpenNMR directory structure has the following sub directories :

C:\OpenNMR\!!commands : This directory contains commands that can be executed by either the OpenBox console or via other control interfaces (ie Excel, Matlab). Commands may be thinbasic scripts, (\basic) or binary executable files (\exe). Other types of files may be added at a later stage,.

C:\OpenNMR\temp : This directory contains temporary files used by the OpenNMR software
C:\OpenNMR\bin : This directory contains various executable files and dlls used by the OpenNMR software.

C:\OpenNMR\filters : This directory contains information regarding the digital filters that are used with NMR data acquisitions.

C:\OpenNMR\shapes : This directory contains shape data for use with shaped gradient and shaped RF experiments.

C:\OpenNMR\system : This directory contains system specific configuration files.

C:\OpenNMR\!!sequences : This directory contains compiled pulse sequence files and parameter files associated with the sequences.

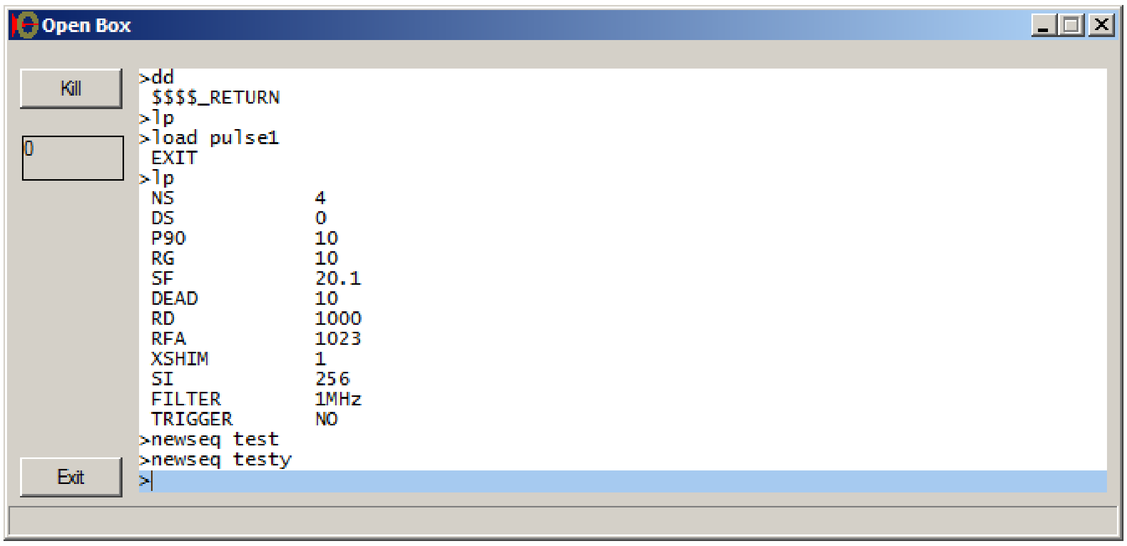
C:\OpenNMR\!!data : The default data directory. The SAVEASCII command saves data to this directory as default.

3. OpenBox

3.1 Introduction

OpenBox provides a simple command line interface console for running NMR experiments and allowing users to perform application development. It is designed for use by NMR experts and engineers.

OpenBox is installed as part of the OpenNMR software suite and can be executed by clicking on the OpenNMR shortcut on the desktop when openNMR is installed.



```
>dd
 $$$$_RETURN
>lp
>load pulse1
 EXIT
>lp
 NS            4
 DS            0
 P90           10
 RG            10
 SF            20.1
 DEAD          10
 RD            1000
 RFA           1023
 XSHIM         1
 SI            256
 FILTER        1MHz
 TRIGGER       NO
>newseq test
>newseq testy
>
```

OpenBox has the following features :

Kill : Stops a currently running pulse sequence or thinbasic script.

Exit : Closes down the current open box

Command Line : This in the command line where instructions are entered. It has a simple editing function (press the up arrow to recall previous commands).

3.2 OpenBox Commands

A list of OpenBox commands can be found by typing help at the OpenBox command line. Commands may come from a variety of sources. For example the NP command (which returns the number of points) is a ThinBasic program, which can be found in the c:\OpenNMR\!!commands\basic directory. All the ThinBasic commands that are shipped with the system can be found in this directory. Refer to the section on ThinBasic for more information about writing new commands. Another directory c:\OpenNMR\!!commands\exe contains exe files that may be executed from the command line. For example the ft.exe file performs a fourier transform on the current data set.

3.3 Running a Pulse Sequence Via OpenBox

To run a pulse sequence via OpenBox you should first bring up a data display window to allow data visualisation. This can be done by typing DD at the OpenBox command line. A data display window should appear.

Next load in a pulse sequence via the command line. Pulse1 is a simple FID sequence. To load in type :

```
LOAD PULSE1
```

Next check the parameters of the sequence. This can be done via typing LP. A list of the parameters defined for use with the sequence is displayed :

NS
DS
P90
RG
SF
DEAD
RD
RFA
XSHIM
SI
FILTER
TRIGGER

Set the parameters by typing the parameter name and the value, eg :

P90 10

Which sets the P90 parameter to 10 microseconds.

A reasonable start set of parameters for the PULSE1 sequence might be :

NS 4
DS 0
P90 10
RG 5
SF (Magnet Frequency in MHz)
DEAD 10
RD 1000 (note RD is measured in milliseconds).
RFA 1023
XSHIM 0
SI 1024
FILTER 1MHz
TRIGGER NO

Note these parameters are not optimised for a particular system. SF should be set to the magnet frequency in MHz, eg 3.0 or 21.6.

Type GO to run the sequence and display interactive data in the data display window.

4. Open NMR Pulse Programming

4.1 Introduction

Pulse programming with OpenNMR is performed via the Code::Blocks development environment. It is first necessary to install this environment, plus the pulse sequence libraries before pulse programming can take place.

Contact Advanced Magnetic Resonance Limited for information on the version of Code::Blocks recommended for use with OpenNMR.

The pulse sequence libraries are provided as a separate .exe file, setup_sequences.exe. On running this executable the pulse sequences and the compilation libraries are installed in the c:\PulseSequences directory. Note that without this directory installed it is impossible to compile pulse sequences.

4.2 Creating a New Pulse Sequence

To create a new sequence, start the OpenBox program and type :

```
Newseq Sequence_name
```

Where sequence_name is the name of the new pulse sequence. Note that this creates the entire Code::Blocks project and directory structure for the new sequence in the directory C:\PulseSequences, so it is advisable to create new sequences like this rather than trying copy files manually.

The Code::Blocks editor will start automatically with a skeleton FID pulse sequence in place.

4.3 Skeleton OpenNMR Pulse Sequence

This section describes the main features of the OpenNMR pulse sequence. OpenNMR pulse sequences are written in C, with special libraries and functions to handle the particular requirements of NMR data acquisition. It is important to remember C is case sensitive, this causes the majority of errors at compilation time.

```
//Sequence Library Header

#include "definitions.h"
#include <stdio.h>
#include <stdlib.h>
#include "eventlib.h"

//Sequence CLI header and start definition

int main(int argc, char *argv[])
{

//Define sequence internal variables here

START_PAR;

//Define accessible parameters here.

END_PAR;

//Assign Accessible Parameters to Spectrometer Properties here.

//Perform pulse sequence internal calculations here.

BeginSequence(number_of_scans);

//Marks the beginning of the event list for the sequence.

//Pulse Sequence Event instructions here

// Marks the end of the event list for the sequence

EndEvents(End_Time,0);

//Defines the end of the sequence code for the C compiler

return(0);
}
```

4.4 Sequence Library Header

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

#include "eventlib.h"
```

These libraries must be included at the beginning of the sequence for the sequence to compile correctly.

4.5 Sequence CLI header and Start Definition

```
int main(int argc, char *argv[])
{
```

This instruction allows command line arguments to be passed to the sequence via the argc argv calling convention. The open bracket { marks the start of the sequence

4.6 Sequence Internal Variable Declarations

The Variable Declaration section should define any variables that are used in the sequence but are not required to be accessible (see below for accessible parameters description). Examples of these might be loop labels and calculated pulse sequence delays :

```
Int
Double
Double
```

Note that these variables are case sensitive. Typical uses are to define a delay based on other parameters such as accessible parameters and to define labels for loops.

4.7 Accessible Parameters Declaration

The Start_PAR instruction marks the start of the accessible parameter definition section. Accessible parameters may be specified via the OpenBox console, or by the external control program. Parameters may be defined as NMR_int (integer), NMR_double (double) or NMR_String (string). Note that the C language is case sensitive. For an FID sequence some examples might be :

```
START_PAR;

NMR_int(NS);
NMR_int(DS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_string(Trigger);
END_PAR;
```

The end of the accessible parameter definition is marked by the END_PAR instruction. Accessible parameters can be given any names, but if used to control a specific spectrometer property (for example the frequency) must be linked to that property (see the section below). For example NMR_double(SF) or NMR_Double(RFFreq) could be defined to hold the RF frequency.

4.8 Assigning Accessible Parameters to Spectrometer Properties

It is necessary to assign certain accessible parameters with specific spectrometer properties. In order to do this the accessible parameters defined above have to be linked to the relevant property. The next block assigns accessible parameters to different spectrometer properties :

```
SetFrequency(SF); //Assign SF to Channel 1 RF frequency
SetRFA(RFA);      //Assign RFA to Channel 1 Hard RF pulse amplitude
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0213");
SetAcquisition(Filter);
SetTrigger(Trigger);
SetDS(DS);
```

A complete list of properties can be found in the Appendix.

Note that without the correct assignment, OpenNMR will not be able to assign an accessible parameter with a particular property, so if the SetRG(accessible_parameter) statement is not present, it will not be possible to control the system receiver gain.

4.9 Sequence Internal Calculations

Calculations internal to the sequence should be performed here. For example a tau value in a CPMG experiment might be defined as :

```
Tau_delay=tau-(p90/2+P180/2);
```

Where tau_delay is the first pulse gap between the 90 and 180 degree pulses. Tau,P90 and P180 could be accessible variables or internal sequence variables.

4.10 Sequence Events Section

The BeginEvents instruction marks the beginning of the pulse sequence scans loop. BeginEvents takes a single argument (the number of scans to be executed, usually defined as NS) :

```
BeginEvents(NS);
```

Timed events in the sequence are specified by the Event instruction. Each instruction takes two arguments, a time and an action to be performed during that time. The default time unit is microseconds. It is recommended that all times are stated explicitly in microseconds by adding the multiplier *us to the time value. *ms and *s can also be used. Note that the multiplier *ms on RD means that this parameter takes effect in ms rather than us, so a value of 1000 specified on the Openbox commandline leads to an RD value of 1 second.

```
Event(RD*ms,0);
```

The instruction below shows multiple actions being performed during an event instruction. In this event instruction transmit enables on RF channels 1 and 2 are performed, and the phase of the RF pulse on channel 1 is set to that specified in PH1 (phase list 1 **.**

```
Event(2*us,TX1+TX2+SetPhase(RF,PH1)); // Enable TX and set RF Phase
```

In this event an RF pulse of length P90 is output on RF channel 1 :

```
Event(P90*us,RF);                         // RF Pulse
```

The following instruction shows the ringdown time following the RF pulse. During this time period the phase of the receiver is set to phase list 1.

```
Event(DEAD*us,SetPhase(REC,PH1));  //  Probe  ring  down  and  set  REC
Phase
```

Phase lists are incremented using the NextPhase action :

```
Event(10*us,NextPhase(PH1));      // Increment Phase List
```

A full list of actions can be found in section 4.12/4.13/4.14.

4.11 EndEvents Instruction and Defining the End of the Sequence

The EndEvents instruction marks the end of the timed pulse sequence events and must be present :

```
EndEvents(END_TIME,0);
```

Following this a final return statement and a close bracket completes the C code :

```
return(0);
}
```

4.12 List of Pulse Sequence Actions

Two types of pulse sequence actions can be performed. System actions may be performed simultaneously with other system actions and generally perform tasks such as performing hard and shaped RF pulses, transmitter enable pulses and data acquisition. Controls actions may be performed in combination with system actions, but not simultaneously with other control actions. Control actions generally perform tasks not associated with spectrometer output input and include such tasks as setting the phase/amplitude/frequency of RF pulses and incrementing phase lists. There is no requirement to perform an action during an Event instruction, no action can be taken by providing the action value 0 are an argument to the Event instruction.

4.13 Summary of System Actions

Event Action	Description	Notes	Minimum Time		
RF	Hard RF Pulse Channel 1		Na		
RF2	Hard RF Pulse Channel 2		Na		
RF3	Hard RF Pulse Channel 3		Na		
RF4	Hard RF Pulse Channel 4		Na		
TX1	Tx Gate 1		Na		
TX2	Tx Gate 2		Na		
TX3	Tx Gate 3		Na		
TX4	Tx Gate 4		Na		

4.14 Summary of Control Actions

Control Action	Notes	Minimum Time
SetPhase(RF/RecChannel,PhaseList)		
NextPhase(PhaseList)		
Loop(LoopLabel)		
EventControl(Control Code, Data Value)		

In general the minimum possible time for a system/control action is 0.2*us, which is also the minimum event time. Many control functions require longer times than this in order to function correctly.

Actions may take effect at different points during the execution of an Event instruction (but will always occur somewhere between the start of the Event instruction and the minimum execution time), so performing system and control actions simultaneously may lead to confusing behaviour. For example if an RF system action and a SetPhase action are performed simultaneously, the system action will output an RF pulse of the previous phase until the phase update is completed by the control action, sometime within the first few nanoseconds of the Event instruction.

4.15 Acquire and WaitTrigger Instructions

There are two special instructions that are not used as part of the Event instruction. Acquire acquires data and takes the argument 0 :

```
Acquire(0);
```

The exact time the Acquire instruction takes is a function of the number of points, the dead time of the receiver and the filter used. The duration of the Acquire instruction is held in the global variable AcqTime, which can be used in sequence internal calculations.

The WaitTrigger() instruction places a wait for a trigger signal at a specific point in the pulse sequence.

The behaviour is as follows :

Define a Trigger accessible parameter as NMR_string :

```
NMR_string(Trigger);
```

Link the parameter to the Trigger property using

```
SetTrigger(Trigger);
```

If the accessible parameter Trigger is set to YES then the pulse sequence will automatically wait before the first event of the sequence by default (the first event after the BeginSequence instruction for a trigger signal.

If the instruction WaitTrigger() is added the trigger event is moved to before that particular event in the sequence.

Setting Trigger<>YES inhibits the trigger signal, the pulse sequence will run without triggering being necessary.

4.16 Loops

Loops can be performed by defining an internal variable to hold the loop label :

```
int             LoopLabel;
```

An accessible parameter should be defined to hold the number of times to loop :

```
START_PAR;
NMR_int(Number_of_loops);
END_PAR;
```

Define the start of the loop in the required part of the sequence :

```
LoopLabel=StartLoop(Number_of_loops);

`Internal Loop events here

Event(1*us,Loop(LoopLable));
```

The end of the loop is defined with the Loop control action, which takes the loop label as an argument.

4.17 Shaped RF

The following steps are required to perform a shaped RF pulse in a pulse sequence :

i) Define a variable of type 'RFShape'.

ii) Load the shape from disk using the 'LoadRFShape' instruction

iii) Define the event in the actual sequence using the 'ShapedRF' instruction.

An example of the use of shapedRF pulses can be seen below :

```
#include "definitions.h"

#include <stdio.h>
#include <stdlib.h>
#include "eventlib.h"

int main(int argc, char *argv[])

{
RFShape RFS1;                           // Declare RF Shape variable

START_PAR;

NMR_int(NS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_double(P1);

END_PAR;

SetFrequency(SF);
SetRFA(RFA);
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0123");
SetPhaseList(PH2,"0123");
SetAcquisition(Filter);

LoadRFShape(&RFS1,"sinc256");           // Load shape from disk

BeginEvents(NS);
Event(RD*ms,0);

Event(2*us,TX1+TX2+SetPhase(RF,PH1));   // Enable TX and set RF Phase
Event(P90*us,RF);                       // Square Pulse
Event(10*us,SetPhase(RF,PH2));          // Set RF Phase
Event(P1*us,ShapedRF(RFS1));            // Shaped RF Pulse
Event(1*MS,0);
Event(DEAD,SetPhase(REC,PH1));          // Probe ring down and set
REC Phase
Acquire(0);
Event(10*us,NextPhase(PH2));            // Increment Phase List
Event(10*us,NextPhase(PH1));            // Increment Phase List
EndEvents(END_TIME,0);

return(0);
}
```

4.18 Spectrometer Properties

The spectrometer properties are certain system properties that need to be set in order to perform NMR experiments. As the accessible parameters can vary in name, the system needs a way of linking the accessible parameters to the spectrometer property. Properties are set using the SetPropertyName instructions. Usually these instructions take single accessible parameters as arguments, although it is possible to use internal variables if the programmer wishes to stop the end user altering parameters. Some instructions such as SetPhaseList require more than one argument.

4.19 List of Spectrometer Properties

Property	Description			
SetFrequency(nmr_double)	Sets the RF Frequency of RF Channel 1			
SetRFA(nmr_double)	Sets the RF amplitude of RF channel 1			
SetSI(nmr_int)	Sets the number of points in the acquisition			
SetRG(nmr_double)	Sets the receiver gain			
SetPhaseList(phaselist,nmr_string)	Sets the specified phaselist (PH1-PH6) to the value specified			
SetAcquisition(nmr_string)	Sets the acquisition filter			
SetTrigger(nmr_string)	Sets the trigger			
SetDS(nmr_int)	Sets the number of dummy scans			

4.19 Compiling Pulse Sequences

Pulse sequences can be compiled from within Code::Blocks by Selecting the Build menu and then selecting Re-Build (or by pressing CNTL-F11). Errors are reported in the console window at the base. Note that if more than one sequence/project is loaded into Code::Blocks it is not always obvious which sequence will be compiled. Programmers should pay careful attention if more than one sequence is loaded that the correct sequence is being compiled.

New sequence modifications only take effect after pulse sequences are re-loaded in OpenBox or other control software, so after a re-compilation a :

```
LOAD new_sequence
```

(where new_sequence is the name of the new pulse sequence) instruction must be issued at the OpenBox command line in order for the changes to take effect.

5. Command and Macro/Script Programming via ThinBasic

5.1 Introduction

Users may create their own commands or scripts/macros that can be executed from either the OpenBox console or other OpenNMR controlling software using the ThinBasic development environment.

5.2 ThinBasic and Editing Commands

Run the ThinBasic IDE (known as ThinAir) from the Windows Start menu. Using the File menu select Open and navigate to :

```
C:\OpenNMR\!!commands\basic
```

And select the command NP.tbasic.

This command returns the number of points to the OpenBox command line :

```
'Skeleton

USES "CONSOLE"

#INCLUDE "..\..\BIN\NMR.INC"

GetNMRData
{Insert Program here}
CALL CloseNMRDataLink
```

The USES and #INCLUDE statements must be present.

The GetNMRData command makes the last acquired data set accessible to the ThinBasic command.

The Call CloseNMRDataLink closes the ThinBasic link with OpenNMR and must be present at the end of the script.

5.3 Variables defined in NMR_inc

The following global variables are defined in NMR.inc and are accessible from commands/scripts :

```
GLOBAL DataDir   AS STRING  ‘Current Data Directory
GLOBAL RootDir   AS STRING  ‘Current Roor Directory
GLOBAL TempDir   AS STRING ‘Current Temporary Directory
GLOBAL SeqDir    AS STRING   ‘Current Pulse Sequence Directory
GLOBAL QueueDir  AS STRING  ‘Current Queue Directory
GLOBAL SystemDir AS STRING  ‘Current System Directory

GLOBAL DataA()  AS DOUBLE  ‘Data array channel A
GLOBAL DataB()  AS DOUBLE  ‘Data array channel B
GLOBAL Argv()   AS STRING     ‘Array of command line arguments
GLOBAL Argc     AS INTEGER   ‘Number of command line arguments
GLOBAL Points   AS LONG         ‘Number of points in the data set
GLOBAL CmdReturn AS STRING    ‘String holding the OpenBox commandline
return value
GLOBAL ApplicationType AS INTEGER ‘Application type
```

5.4 Simple ThinBasic Command : NP

This command gets the number of points (held in the global variable points) and writes the number to the OpenBox console via the Console_Writeline command:

```
USES "CONSOLE"

#INCLUDE "..\..\BIN\NMR.INC"

GetNMRData
Console_Writeline(Points)    ' Print result for NMRCore to pick up
CALL CloseNMRDataLink
```

5.5 Simple ThinBasic Command : MAXAPOS

This command searches the data array for channel A and returns the point number than contains the maximum value :

```
'MaxAPos - The position of the maximum value if Channel A

USES "CONSOLE"

#INCLUDE "..\..\BIN\NMR.INC"


GLOBAL n         AS LONG
GLOBAL a         AS DOUBLE
GLOBAL MaxV      AS DOUBLE
GLOBAL MaxAPos   AS LONG

GetNMRData

MaxAPos=-1
MaxV=0
FOR n=1 TO Points
  a=DataA(n)
  IF ABS(a)>ABS(MaxV) THEN
    MaxV=a
    MaxAPos=n
  END IF
NEXT

Console_Writeline(MaxAPos)
CALL CloseNMRDataLink
```

5.6 Command Naming Convention

New OpenNMR installations will overwrite the existing \!!commands directory source code. This will cause user modified versions of the original commands to be lost. AMR Ltd recommends that if users wish to modify existing instructions they are saved with a new filename.

6. Gradient Control

6.1 Simple Gradient Control

The simplest form of gradient control is as follows :

```
#include "definitions.h"

#include <stdio.h>
#include <stdlib.h>
#include "eventlib.h"

int main(int argc, char *argv[])

{
START_PAR;

NMR_int(NS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_int(G1);
END_PAR;

SetFrequency(SF);
SetRFA(RFA);
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0");
SetPhaseList(PH2,"0");
SetAcquisition(Filter);

BeginEvents(NS);
Event(RD*ms,0);
Gradient(2*ms,GradA(SQUARE(G1)));     // Gradient on
Gradient(1*ms,GradA(SQUARE(0)));      // Gradient off
Event(P90,RF);                        // 90 Pulse
Event(DEAD,SetPhase(REC,PH1));        // Probe ring down and set REC
Phase
Acquire(0);
Event(10*us,NextPhase(PH1));          // Increment Phase List
Event(10*us,NextPhase(PH2));          // Increment Phase List
EndEvents(END_TIME,0);

return(0);
}
```

The two important lines are :

```
Gradient(2*ms,GradA(SQUARE(G1)));      // Gradient on
Gradient(1*ms,GradA(SQUARE(0)));       // Gradient off
```

The first argument defines how long the gradient command will take, so the sequence above gives a 2ms gradient pulse followed by a 1ms delay. The gradients are implemented as shapes with just one point in them. The hardware plays out this single point at the start of the allocated time.

GradA defines which channel to use, this could also be GradB or GradC.

SQUARE is used to indicate this should be a simple gradient control and takes one integer argument which defines the strength of the gradient.

6.2 More Flexible Gradient Control

To provide more control the gradients have to be broken down into two parts. Firstly there are the ***Gradient Properties***. These are the gradient shape and whether the strength is fixed or is taken from a table. Secondly there are the ***Sequence Properties***. These are the position that the gradients occur within a sequence and the time they take to execute.

```
#include "definitions.h"

#include <stdio.h>
#include <stdlib.h>
#include "eventlib.h"

int main(int argc, char *argv[])

{
GradControl     GC0;
GradControl     GC1;

START_PAR;

NMR_int(NS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_int(G1);
NMR_double(D1);
END_PAR;

DefineFixedGradient(&GC0,&RAMP_DOWN,G1);
DefineFixedGradient(&GC1,&RAMP_UP,G1);

SetFrequency(SF);
SetRFA(RFA);
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0");
SetPhaseList(PH2,"0");
SetAcquisition(Filter);

BeginEvents(NS);
Event(RD*ms,0);
Gradient(1*ms,GradA(&GC1));           // Gradient on
Event(D1*us,0);
Gradient(1*ms,GradA(&GC0));           // Gradient off
Event(P90,RF);                        // 90 Pulse
Event(DEAD,SetPhase(REC,PH1));        // Probe ring down and set REC
Phase
Acquire(0);
Event(10*us,NextPhase(PH1));          // Increment Phase List
Event(10*us,NextPhase(PH2));          // Increment Phase List
EndEvents(END_TIME,0);

return(0);
}
```

These following lines are used to define some gradient control variables :

```
GradControl     GC0;
GradControl     GC1;
```

The variables then have to be initialised. In this example they are fixed values and use predefined shapes called RAMP_UP and RAMP_DOWN :-

```
DefineFixedGradient(&GC0,&RAMP_DOWN,G1);
DefineFixedGradient(&GC1,&RAMP_UP,G1);
```

Note that both gradient controls use the same amplitude value. It is not immediately obvious that this needs to be done for the second gradient, but it is important to remember that the amplitude value is used to define the gradient strength of the highest part of the shape, which is the first point. If an amplitude of zero was used then all points in the shape would be multiplied by zero and gradient shape would simply step down like a square pulse.

The predefined shapes are SQUARE_UP, SQUARE_DOWN, RAMP_UP and RAMP_DOWN. Other shapes need to be handled within the sequence. The easiest way to do this is to load it from a text file on disk.

In order to do this it is first necessary to define some Gradient Shape variables, such as :

```
GradShape       GRS1;
GradShape       GRS2;
```

This is done at the same place within the C program as the GradControl variables.

Next the shapes have to be loaded from disk like this:-

```
LoadGradShape(&GRS1,GSH1);
LoadGradShape(&GRS2,"GAUSS256");
```

Note that you can either use a fixed name like “GAUSS256” or an NMR_string variable. In either case the shape must be stored in C:\OpenNMR\shapes and have a .TXT extension.

Finally the DefineFixedGradient call would look like this:-

```
DefineFixedGradient(&GC1,&GRS1,G1);
```

Within the sequence itself the syntax is the same as for the predefined shapes.

6.3 Multiple Shapes within a Pulse Sequence

It is possible to create a sequence where an OpenNMR variable is used to choose what type of gradient shape should be used :

```
#include "definitions.h"

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

#include "eventlib.h"

int main(int argc, char *argv[])

{

GradShape       GRS1;
GradShape       GRS2;

GradControl     GA;
GradControl     GB;

START_PAR;
NMR_int(NS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_string(GSH1);
NMR_string(GSH2);
NMR_int(G1);
NMR_int(SType);
END_PAR;

SetFrequency(SF);
SetRFA(RFA);
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0213");
SetAcquisition(Filter);

LoadGradShape(&GRS1,GSH1);
LoadGradShape(&GRS2,GSH2);


if(0==SType)
  {
  DefineFixedGradient(&GA,&SQUARE_UP,G1);
  DefineFixedGradient(&GB,&SQUARE_DOWN,G1);
  }
else if (1==SType)
  {
  DefineFixedGradient(&GA,&RAMP_UP,G1);
  DefineFixedGradient(&GB,&RAMP_DOWN,G1);
  }
else
```

```
  {
  DefineFixedGradient(&GA,&GRS1,G1);
  DefineFixedGradient(&GB,&GRS2,G1);
  }

BeginEvents(NS);

Event(RD*ms,0);
Event(2,TX1+TX2+SetPhase(RF,PH1));       // Enable TX and set RF Phase
Event(P90,RF);                           // Pulse

Gradient(5*ms,GradA(&GA));
Event(10*ms,0);
Gradient(5*ms,GradA(&GB));

Event(DEAD,SetPhase(REC,PH1));              // Probe ring down and set
REC Phase
Acquire(0);
Event(10*us,NextPhase(PH1));             // Increment Phase List
EndEvents(END_TIME,0);

return(0);
}
```

In the above sequence the variable **Stype** is used to define either square, ramped, or user defined gradients. It is important to note that the sequence proper, i.e. between BeginEvents and EndEvents, does not change.

6.4 Calculating Shapes within a Pulse Sequence

It is possible to calculate the gradient shape from within a pulse sequence instead of loading it from disk. This is how RAMP_UP and RAMP_DOWN gradient shapes are generated. There are a number of steps involved but they are built upon the examples above. In order to do this, first Define a GradShape variable. This variable is in fact a C struct with two field; GradBuffer and Size. Next initialise the fields. For example the sequence libraries initialise RAMP_UP like this:-

```
Grad_Shape_Array  tempgrad;

        for(i=0;i<128;i++)
          {
          tempgrad[i]=i;
          }
        ConvertGradText(tempgrad,RAMP_UP.GradBuffer,128);
        RAMP_UP.Size=128;
```

In the above example, tempgrad is an array of floating point numbers. ConvertGradText scales the numbers and converts them to the correct hardware format.

6.5 Table Gradients

Below is a simple example of a sequence using a Table Gradient:

```
#include "definitions.h"

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

#include "eventlib.h"
```

```
#define INC_TABLE 1


int             TableI[4096];
int             TableIStart;

int main(int argc, char *argv[])

{
GradControl     GC1;

START_PAR;

NMR_int(NS);
NMR_double(P90);
NMR_double(RG);
NMR_double(SF);
NMR_double(DEAD);
NMR_double(RD);
NMR_double(RFA);
NMR_int(SI);
NMR_string(Filter);
NMR_string(Trigger);
NMR_int(G1);

END_PAR;

MakeStepTable(TableI,NS,G1);
TableIStart=FRAMLoadList(&TableI[0],NS);
InitList(INC_TABLE,TableIStart);

DefineTableGradient(&GC1,&SQUARE_UP,INC_TABLE);

SetFrequency(SF);
SetRFA(RFA);
SetSI(SI);
SetRG(RG);
SetPhaseList(PH1,"0");
SetPhaseList(PH2,"0");
SetAcquisition(Filter);
SetTrigger(Trigger);

BeginEvents(NS);
Event(RD*ms,0);
Gradient(1*ms,GradA(SQUARE(10000)));    // Square gradient pulse
Gradient(1*ms,GradA(SQUARE(0)));        // to trigger 'scope
Event(2*ms,0);
Gradient(2*ms,GradA(&GC1));             // Table gradient on
Gradient(1*ms,GradA(SQUARE(0)));        // gradient off
Event(P90,RF);                          // 90 Pulse
Event(DEAD,SetPhase(REC,PH1));          // Probe ring down and set
REC Phase
Acquire(0);
Event(10*us,NextPhase(PH1));            // Increment Phase List
Event(10*us,NextPhase(PH2));            // Increment Phase List
StepList(100*us,INC_TABLE,1);           // Next value from table
EndEvents(END_TIME,0);

return(0);
}
```

An array should be defined to hold the table values and an integer which will be the tables start address within the gradient waveform memory

```
int             TableI[4096];
int             TableIStart;
```

A gradient control variable should also be defined as before :

```
GradControl     GC1;
```

The table array can be populated by many different methods (including direct specification of the gradient values). In this particular example the MakeStepTable function used :

```
MakeStepTable(TableI,NS,G1);
```

NS is the size of table to generate and G1 is the step size.

Once the table has been populated with values it needs to be loaded into gradient wavedform memory as follows :

```
TableIStart=FRAMLoadList(&TableI[0],NS);
```

The value returned by FRAMLoadList is the gradient waveform memory address that the table will to loaded to. Finally the table needs to be initialised by associating it with one of the list pointers :

```
InitList(INC_TABLE,TableIStart);
```

The first argument is simply a number between 1 and 7 which defines which list pointer to use.

Once the list has been completely defined and initialised then it can be associated with a Gradient Control variable :

```
DefineTableGradient(&GC1,&SQUARE_UP,INC_TABLE);
```

The line in the sequence to execute the Table Gradient is the same as before:

```
Gradient(2*ms,GradA(&GC1));             // Table gradient on
```

6.6 Pulse Sequence User Errors

Users may generate their own errors from within pulse sequences via the SequenceError command :

```
SequenceError(user_error_number,message);
```

User_error_number should be in the range 901-999. Message is a string that appears in the pop up dialog box reporting the error. The pop up dialog box appears for 10 seconds then disappears, or can be removed by clicking on the OK button.

For example :

```
if(RFA>512)
{SequenceError(901,"RFA too high");
}
```

7. Shim Control

OpenNMR has two separate methods of setting shims.

Firstly, in the C:\OpenNMR\system\system.cfg file the [Gradients] section specifies limits for the shim values and default values. These default values will be present for ALL pulse sequences and will be active when gradient amplifiers are turned on :

```
[Gradients]
ShimXMax=100
ShimYMax=100
ShimZMax=100
ShimX=10
ShimY=15
ShimZ=30
```

In the above specification the values of ShimX, ShimY and ShimZ cannot be greater than 100, and the default values (present for all pulse sequences) are 10,15 and 30 for the X,Y and Z shims respectively.

Shims can also be controlled from within a pulse sequence by defining variables XShim,YShim and ZShim and using the SetShims command, eg :

```
NMR_int(ShimX);
NMR_int(ShimY);
NMR_int(ShimZ);

SetShims(ShimX,ShimY,ShimZ);
```

In this scenario ShimX, ShimY and ShimZ can be controlled from the OpenNMR command line, but still cannot exceed the maximum allowed value for the shim specified in the system.cfg file. If values are not specified in the system.cfg file the shim limits default to zero for protection purposes, and thus ShimX,ShimY and ShimZ must be zero or an error will be reported.

8. OpenNMR Supervisor Board Commands

The supervisor board associated with many types of AMR Ltd equipment is used to supervise and control various functions that are not needed to be performed synchronously with the pulse sequence.

These include, but are not limited to :

Setting and reading the magnet box temperature via Eurotherm controllers.

Reading the temperature of the gradient set.

Reading temperature sensors associated with other functions.

Reading and setting the temperature of the variable temperature control unit.

Setting external relays.

Reading Tomco RF amplifier status interfaces.

Monitoring system voltage rails.

Controlling further hardware elements via the I2C interface.

The exact meaning of the OpenNMR commands and what they report depends on the configuration of the supervisor board and the system it is used on.

8.1 ADC Command

The ADC's can be read from OpenNMR using the ADC command. This will report the eight adc values or an error if the ADC's cannot be read (normally this is because the supervisor board USB cable is not connected).

```
ADC
```

The eight ADC values are reported in sequence and can be accessed via the CmdReturn parameter. ADC will report an error if the ADC's cannot be read. Because the ADC's are part of the supervisor microcontroller this will always mean there is a problem with the supervisor to host USB interface rather than the ADCs themselves.

ADC values vary from 0-1023. The first 3 values are typically (although not always) associated with temperature sensors. A value of 260 indicates 35 degrees from a PT100 sensor. The ADC's have a resolution of 2 degrees C per bit when used with the on board PT100 sensors.

The remaining values are often (although not always) associated with voltage rail monitoring. The inputs vary from 0 (0V) to 1023 (2.5V).

8.2 Fan Command

The fan command reads the dedicated fan controller which can be linked to monitor the various system fans. A value of 255 indicates a fan is stationary, any other value indicates a fan is operating.

```
SYSTEM FAN1
```

returns the value of the fan1 input.

8.3 DAC Command

The DAC command sets one of the 4 supervisor board DAC channels. The DAC's can be set as follows :

```
DAC Channel_Number Value
```

Channels are numbered from 0 to 3. Values may vary from -32768 to +32767.

On certain systems DAC channels may be mapped to shim power supply channels. The DAC command allows the user to shim the system with these power supply channels.

Once a shim value has been established the channel value can be updated in the supervisor board by typing :

```
DAC Channel_Number Store
```

The DAC channel with then always be set to the last specified value once the system is started up. This allows the correct shim values to be recalled when the system is powered back on.

Users should not repeated store values to DAC channels, as this makes use of EEPROM memory and takes time, as well as having a finite read/write cycle life. Users should vary the DAC channel values, establish the correct value for the DAC channel before finally writing the DAC channel value to disk.
The EEPROM stored value for each DAC channel can be retrieved by typing :

```
DAC Channel_Number Query
```

Note that this may not necessarily be equal to the current DAC channel value, as this value may not have been stored.

8.4 SYSTEM Command

The system command recalls all configurable information stored on the supervisor board.

```
SYSTEM VERSION     Print control system software version
SYSTEM CPU         Print ID code of control system CPU
SYSTEM DACS        Returns status of DACS. None zero values are errors
SYSTEM FAN1 (or 2) Returns fan spin rate. The lower the value the faster
                   the fan is turning. 255 indicates fan has stopped.
SYSTEM USB         Checks if USB to spectrometer is working
SYSTEM GRAD        Reports whether the gradients have been disabled because
                   of over temperature
```

```
 1  USB communication status. 0=OK
 2  Embedded micro software version
 3  Embedded micro signature 1
 4  Embedded micro signature 2
 5  Embedded micro signature 3
 6  I2C Temperature Status
 7  I2C DAC status
 8  Debug1
 9  Debug2
10  PORTC
11  Tach1
12  Tach2
13  PORTE
14  PORTD
15  PORTA
16  I2C Temperature High Byte
17  I2C Temperature Low Byte
18  ADT7410 status byte
19  Not used
```

9. Running OpenNMR Via Microsoft Excel Sheets

9.1 Introduction

OpenNMR can be directly controlled via Microsoft Excel Spreadsheets. Because of various issues regarding the interaction of OpenNMR and Excel, OpenNMR has to be run in a particular way via a separate sub program called OpenAndShut.

9.2 Excel Sheet Example

An Excel skeleton sheet exists which provides an example for controlling OpenNMR via Excel. This sheet runs the basic FID pulse sequence (pulse1), allows the user to change parameters and returns the data and the size of the data from OpenNMR.

The skeleton sheet contains the following code :

Module1 :

This contains the OpenNMR functions and global variables.

Workbook Open :

Contains code that is executed when the workbook opens or closes. This includes starting the data display window and populating the window with dummy data.

Sheet1 :

Contains the code for running the sequence via the GO button. This includes code for clearing the sheet, loading the pulse1 sequence, setting the parameters, running the sequence, getting the data, processing the data and writing it to the sheet and getting the size of the data via the OpenNMR “SIZE” command.

9.3 Setting Up the Skeleton for Execution Using the OpenAndShut Program

It is recommended that the skeleton.xls file is copied to a folder in the root directory designated to contain all Excel sheets for use with OpenNMR, for example c:\OpenNMRSheets.

The sheet must not be run directly, but via a shortcut which calls the OpenAndShut program, Excel and the sheet in question.

The shortcut parameters should be :

Target :

```
C:\OpenNMR\bin\OpenAndShut.exe "C:\ProgramFiles\Microsoft
Office\Office11\excel.exe" "c:\OpenNMRSheets\Skeleton"
```

Note that different versions of Microsoft Office have different locations for the excel.exe binary.
This example shows the location for Microsoft Office 2003.

Replace "c:\OpenNMRSheets\Skeleton" with another name in order to run another excel sheet. Note that directories specified with spaces need quotes surrounding their names.

Start In :

```
C:\OpenNMR\bin
```

This is the directory where OpenAndShut.exe, the OpenNMR sub program is located.

10. APP Wrappers

An APP wrapper provides a convenient way of wrapping up a number of OpenNMR elements into an application or APP. Elements might be a pulse sequence, some parameters and some scripts or executables.

An APP consists of a text file placed in the OpenNMR \!!APP directory.

A typical APP might look as follows :

Parameters : A list of Parameters used in the APP. These parameters will be added to the parameters when data acquired by the APP is saved (using LOADASCII or XLS). Parameters will be recalled when the APP is reloaded as they are stored in the APP directory. These parameters can be thought of as specific to the APP, rather than specific to the pulse sequence. They can be changed in an identical way to pulse sequence parameters.

Init : An initialisation script.

Run : A run script.

Sequence : A sequence script.

AN example APP might be as follows :

```
LONG VALUE1
STRING NAME
INIT _STARTSCRIPT
RUN _MAINSCRIPT
SEQ PULSE1
```

The user specifies the APP at the OpenNMR command line by typing :

```
APP Appname
```

The APP will automatically load the SEQ sequence (in the above case pulse1. An LP command (list parameters) shows both the sequence parameters and the APP parameters. The APP parameters are stored in Appname.par in the OpenNMR !!APP directory and will be exported with the pulse sequence parameter data if for example an XLS command is used to export data.

When the user types GO the APP runs the init executable (which may be a thin basic script or another type of exe file in the OpenNMR !!Commands directory). Following this the Run executable runs the main executable/script.

APPs have specific pulse sequences tied to them, so it is impossible to change a pulse sequence while an APP is running.

An APP can be stopped by typing :

```
APP none
```

At the OpenNMR command line.

11. AMRForms

AMRForms allows users to rapidly create a series of forms and simple windows features such as checkboxes to facilitate user input and display information from within ThinBasic scripts.

Users should note that AMRForms requires ThinBasic v1.8.9 in order to function correctly.

Forms lack the configurability of usual Windows forms components, but have the advantage of greater ease of use. Users wishing to create more complex forms always have the option of using the ThinBasic windows user interface which contains a much higher level of configurability and features, but is much more difficult to use.

To enable the AMRForms component the AMRForm library must be specified at the beginning of the thinbasic script :

```
#INCLUDE "..\..\BIN\NMR.INC"
#INCLUDE "..\..\BIN\AMRForm.INC"
```

Parameters can then be defined for use with the form :

```
Global SampleID      As String
Global Scans         As Long
Global RG            As Double
Global r             As Long
Global v             As Variant
Global MagData       As Long
Global SW            As String
```

The following function allows files to be created in the OpenNMR file numbering format :

```
' If filename ends in 2 or more dogits then this function
' returns the next filename in the sequence.

Function NextFileName(CurrentName As String) As String

Local i       As Long
Local s       As String
Local fs      As String
Local NewName As String

NewName=CurrentName
i=1
s=""
While i<Len(CurrentName)
  s=Repeat$(i,"#")
  fs=Repeat$(i,"0")
  If IsLike(RIGHT$(CurrentName,i),s,%TRUE)=FALSE Then
    Exit While
  End If
  i=i+1

Wend

If Len(s)>=3 Then
  i=i-1
  fs=LEFT$(fs,i)
  NewName=LEFT$(CurrentName,Len(CurrentName)-i)
  s=RIGHT$(CurrentName,i)
  i=Val(s)+1
  s=Format$(i,fs)
  NewName=NewName & s
End If
Return NewName
End Function
```

The following code gets values for the parameters displayed in the form :

```
GetNMRData
GetNMRParameter("NS",Scans)
GetNMRParameter("RG",RG)
GetNMRParameter("Filter",SW)
GetNMREnvironment("FILENAME",SampleID)
GetNMREnvironment("MAGDATA",MagData)
SampleID=NextFileName(SampleID)
```

The following code constructs the form. AMRFormStart contains the form label. AMRFormString,AMRFormLong and AMRFormDouble produce parameter entry boxes in order to allow the user to enter data. AMRFormCheckBox produces a check box and AMRFormList a drop down list :

```
AMRFormStart("Test1")
AMRFormString("Filename",SampleID,16)
AMRFormLong("Scans",Scans,5)
AMRFormDouble("Receiver Gain",RG,6,1)
AMRFormCheckBox("Save Magnitude Data",MagData)
AMRFormList("Sweep Width",SW,"1MHz|500KHz|100KHz")
r=AMRFormEnd
```

The variable r contains the result of pressing the OK button, either %IDOK or %IDCANCEL :

```
If r=%IDOK Then
  AMRFormRetrieve("Filename",SampleID)
  AMRFormRetrieve("Scans",Scans)
  AMRFormRetrieve("Receiver Gain",RG)
  AMRFormRetrieve("Save Magnitude Data",MagData)
  AMRFormRetrieve("Sweep Width",SW)
  UpdateNMRParameter("NS",Scans)
  UpdateNMRParameter("RG",RG)
  UpdateNMRParameter("Filter",SW)
  SetNMREnvironment("FILENAME",SampleID)
  SetNMREnvironment("MAGDATA",MagData)

  NMR("GO")
  If MagData<>0 Then
    NMR("MAG")
  End If
  NMR("XLS " & SampleID)
End If
```

In the above code the values for Scans etc are retrieved from the form into the relevant parameters. The sweep width will take one of the values of the drop down list (1MHz, 500KHz or 100KHz). MagData will take on the value 1 or 0 depending on the status of the check box (0 not checked, 1 checked).

The CloseNMRDataLink Statement finishes the script.

```
CloseNMRDataLink
```

This example script can be found in the OpenNMR \!!Commands directory (formdemo.tbasic).

12. OpenNMR Configuration File

The OpenNMR Configuration File contains system specific information for OpenNMR to use. The file may be found in the c:\OpenNMR\system directory and is named system.cfg.

The file contains the following text :

```
[VT]
Port=7
Type=EUROTHERM

[Sequence]
MaxPulse=1000
RF1=1
RF2=3
RF3=4

[Gradients]

ShimXMax=1000
ShimYMax=1000
ShimZMax=1000
```

The [VT] section contains information on the AMR Thermal Control Unit. This unit communicates with the host PC via an RS232 link. The user must specify the type of controller (EUROTHERM) and the RS232 port number that the controller uses. This can be found in the Windows Device Manager application if necessary.

The [Sequence] section contains information specific to pulse sequences, including the maximum hard RF pulse length and the RF channel mapping.

The [Gradients] section contains limits for the shim values. No shim parameter is allowed to exceed these values.

13. Environment Variables

Environment varibles store information that is used by OpenNMR for various tasks. The environment variables can be displayed with the listenv command :

```
1 STATUS_BAR=
```

The information displayed in the status bar at the bottom of the OpenBox window.

```
2 CLIPPED=
```

Whether the data acquisition has clipped or not. CLIPPED=0 (not clipped) or 1 (clipped).

```
3 SEQ=
```

The current sequence loaded.

```
4 CHIP_ID=
```

The ID of the USB chip in the Spectrometer.

```
5 SEQ_ERROR=0
```

An error code that is set if the sequence fails. Is set to zero otherwise.

6 SCAN=DONE

The current scan number or DONE if the sequence is complete.

7 TOTAL_SCANS=1

The total number of scans last performed by the sequence.

8 SNR=0

The signal to noise ratio if in an SNR acquisition.

9 APP =

The last App used.

Environment variables can be used by controlling software to interrogate the current status of the system. For example the system can be started using the GORI command and the current scan number being performed can be found using the SCAN parameter. This can be observed by running a pulse sequence using OpenBox using GORI and typing LISTENV to see the parameters as they update.

14. Use of the GO Command and Variations

The GO command as used from the OpenBox command line can be used in several different ways :

GO

GO NZ : This method starts a pulse sequence without zeroing the data memory that is currently in the spectrometer.

GO SNR n : This method starts a pulse sequence and acquires until the data achieves a specific signal to noise ratio (specified with n).

GORI : This method starts a pulse sequence and then returns control immediately back to the control program (be it OpenBox, an Excel spreadsheet or any other control method). The control program must poll one of the environment variables continuously in order to determine when the sequence is complete. A VBA example is as follows :

```
NMR ("GORI")

Do
DoEvents
Sleep (100)
n = GetNMREnvironment("SCAN", s)
Scancountstring = "[" & s & "]"
Loop Until InStr(UCase$(s), "DONE") <> 0
```