AMR Database Programming Manual

Revision : 0.3/Software Release
Date : 14th June 2015
Author : TBB.

Errors and omissions should be reported to :

AMR Limited
Culham Innovation Centre
D5 Culham Science Centre
Abingdon
Oxon
OX143DB.

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

Introduction

This manual describes the functionality of the AMR database. It contains information for programmers wishing to use the AMR database with their own software for end user applications.

What is the AMR Database ?

The AMR Database is an SQL Lite database which can be used to organise scientific data. In particular it can hold any data that is organised in three column format with an associated parameter set, so this allows it to be used with additional data modalities as well as NMR (nuclear magnetic resonance) data. Parameters may be saved in one of three formats, STRING, LONG or DOUBLE. Each parameter has a name, a value and a units field. Notes may be associated with each dataset, and fields such as DATATYPE allow processing software to establish the difference between varying types of data. Processing/Environment history lists allow a record of data acquisition and data processing history to be recored.

In addition to the numerical data described above, the database can also hold references to non numerical data files such as images and documents (pdf, .doc). This functionality allows the user to create an entire database consisting of data sets from multiple modalities on a single sample, along with additional documentation files containing information such as descriptions and images.

Data saved into the database may be linked together as series or collections to facilitate the loading and processing of data in applications. Data sets may be members of multiple series and multiple collections. Collections of collections may be formed.

The database itself is free format, but certain conventions are adhered to for use with the OpenNMR software and other OpenNMR data processing packages (for example DEEPER). If users wish to use AMR applications with the database, they should adhere to the AMR conventions, which are outlined in other documents.

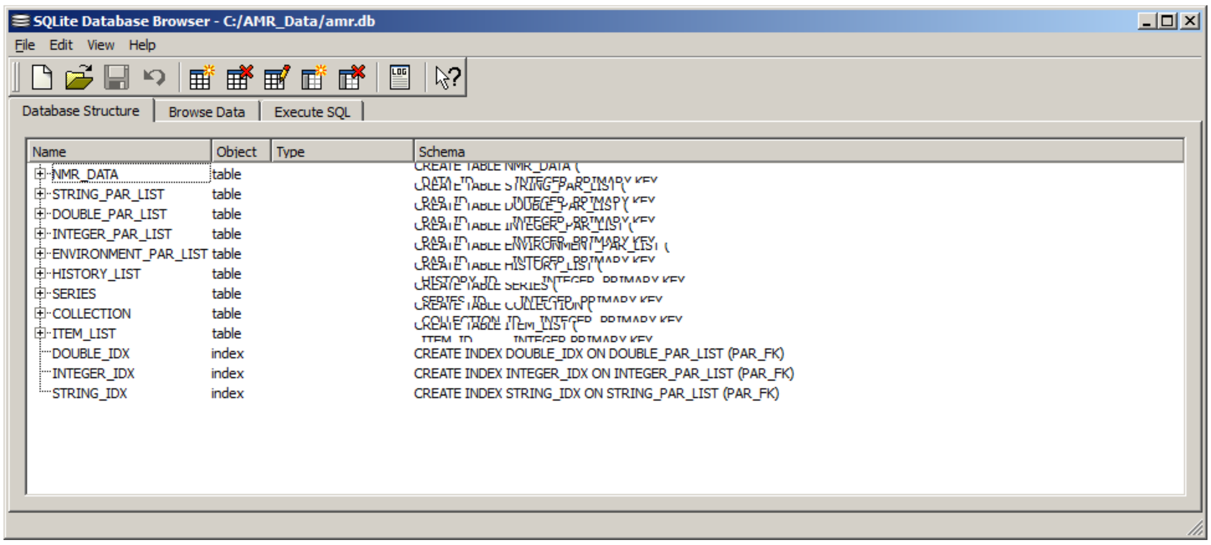
Installation

To install the AMR Database, download the OpenNMR installation setup file and execute.

The file installs both the AMR database under the default directory (c:\AMR_Data) and various DLL control libraries for use with OpenNMR and ThinBasic. In addition users may interface to the database directly using the sqllink.dll. The API for this DLL is appended to the end of this document. Specifying the calling functions to the DLL can be problematic - AMR Ltd has extensive experience in writing calling functions for the DLL, so many function calls have already been written in the most popular languages. Contact AMR Ltd for further advice.

Viewing the Database with a DataBase Browser

The AMR Database and elements contained within it can be viewed using a standard SQL Lite data base browser. The database contains a number of structures which serve as methods of organising data. The structure containing references to the data and parameters is called the data structure. To view the database, load the database file (c:\AMR_DATA\amr.db) into a suitable database browser.



Name	Object	Type	Schema
NMR_DATA	table		CREATE TABLE NMR_DATA (DATA_ID INTEGER PRIMARY KEY
STRING_PAR_LIST	table		CREATE TABLE STRING_PAR_LIST (PAR_ID INTEGER PRIMARY KEY
DOUBLE_PAR_LIST	table		CREATE TABLE DOUBLE_PAR_LIST (PAR_ID INTEGER PRIMARY KEY
INTEGER_PAR_LIST	table		CREATE TABLE INTEGER_PAR_LIST (PAR_ID INTEGER PRIMARY KEY
ENVIRONMENT_PAR_LIST	table		CREATE TABLE ENVIRONMENT_PAR_LIST (PAR_ID INTEGER PRIMARY KEY
HISTORY_LIST	table		CREATE TABLE HISTORY_LIST (HISTORY_ID INTEGER PRIMARY KEY
SERIES	table		CREATE TABLE SERIES (SERIES_ID INTEGER PRIMARY KEY
COLLECTION	table		CREATE TABLE COLLECTION (COLLECTION_ID INTEGER PRIMARY KEY
ITEM_LIST	table		CREATE TABLE ITEM_LIST (ITEM_ID INTEGER PRIMARY KEY
DOUBLE_IDX	index		CREATE INDEX DOUBLE_IDX ON DOUBLE_PAR_LIST (PAR_FK)
INTEGER_IDX	index		CREATE INDEX INTEGER_IDX ON INTEGER_PAR_LIST (PAR_FK)
STRING_IDX	index		CREATE INDEX STRING_IDX ON STRING_PAR_LIST (PAR_FK)

Data Structure

Data Table : Contains information regarding the data, including a reference to the three column data file, user notes on the data, data checksums and file types.

String Par Table : A table containing string parameters. The string table has fields for the name of the parameter (32 chars) the parameter value (1024 chars) and the parameter units (32 chars).

Double Par Table : A table containing double parameters. The table has fields for the name of the parameter, the parameter value and units. In applications this table is used to store NMR parameters such as P90, the ninety degree pulse length.

Integer Par Table : A table containing integer parameters. The table has fields for the name of the parameter, the parameter value and units. In NMR applications, this table is used to store integer parameters associated with the data set. An example might be SI, the number of data points, or NS, the number of scans.

Environment Par Table : A table containing environment parameters. This is used in NMR terms to hold parameters from the OpenNMR environment. Refer to the OpenNMR manual for more information regarding this. A parameter example is TOTALSCANS, the number of scans performed in the NMR experiment.

History Par Table : A table containing history parameters. These parameters contain historical information regarding how the data were processed following acquisition. OpenNMR maintains this historical information and writes it to the database along with other parameters and the data.

Series Structure

The series structure provides a method of grouping data structures together. A series structure can hold a number of data structures. An example might be an NMR T1 experiment, where data is recorded for a number of different inversion values and for organisation purposes this data should be interlinked. A series can reference an arbitrary number of data structures, but cannot reference collections (see below). It is not possible to create a series of a series.

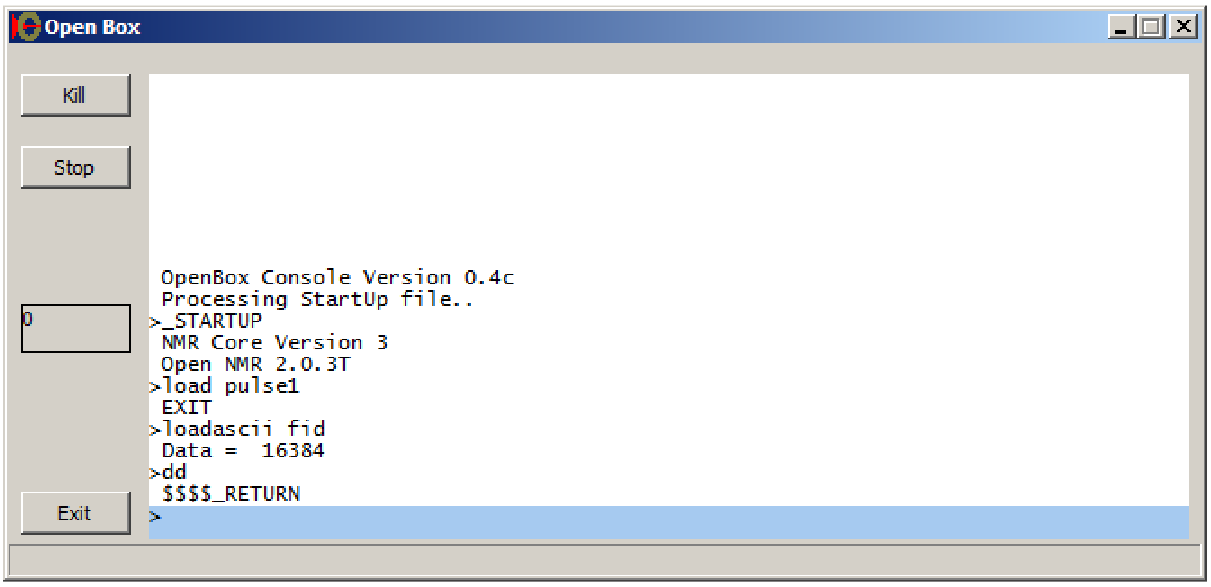
Collection Structure

The collection structure provides a method of grouping data structures, series structures and collection structures. Thus it is possible to generate collections of collections, and collections of data and series together. The collection structure allows a higher level of organisation, and can be used to group data such as NMR T1-T2 data or three dimensional data, or different data modalities for a single sample.

Saving the Data to the Data Base using OpenNMR

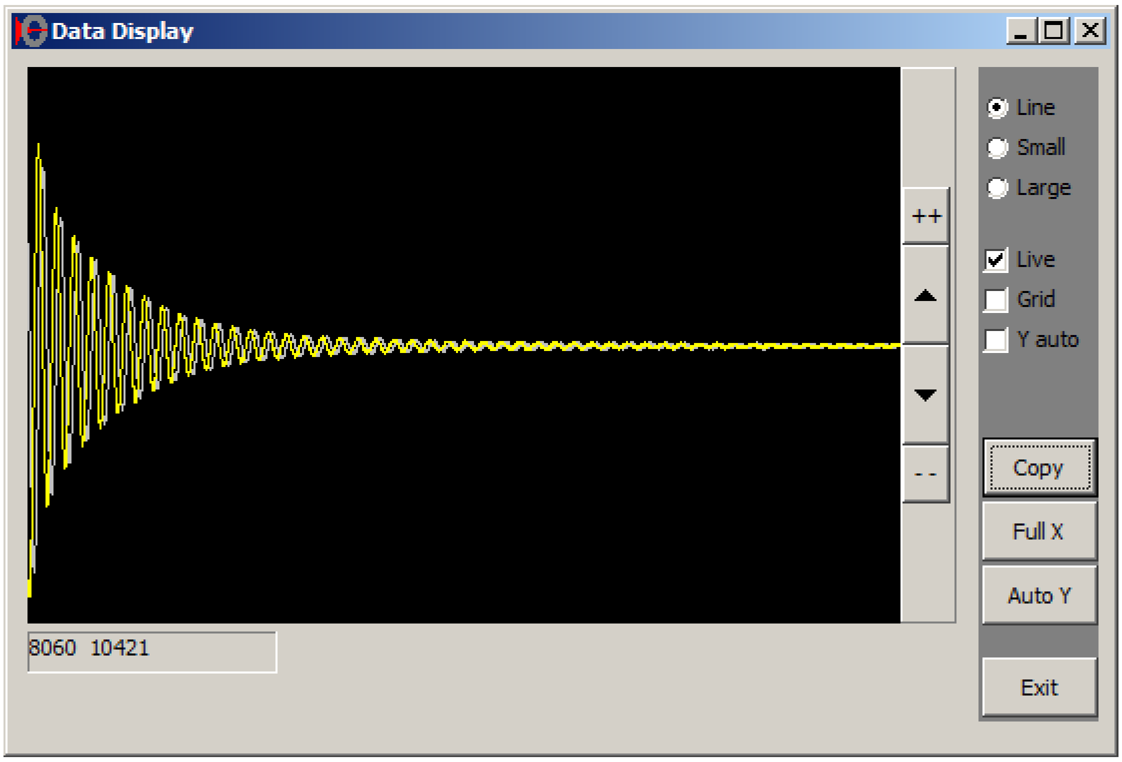
To save a single file to the database using OpenNMR, run the OpenNMR command box and type :

LOAD PULSE1
LOADASCII FID.



```
 OpenBox Console Version 0.4c
 Processing StartUp file..
>_STARTUP
 NMR Core Version 3
 Open NMR 2.0.3T
>load pulse1
 EXIT
>loadascii fid
 Data =  16384
>dd
 $$$$_RETURN
>
```

This should display a demonstration data FID data set (type DD at the command prompt to view the data).



```
8060  10421
```

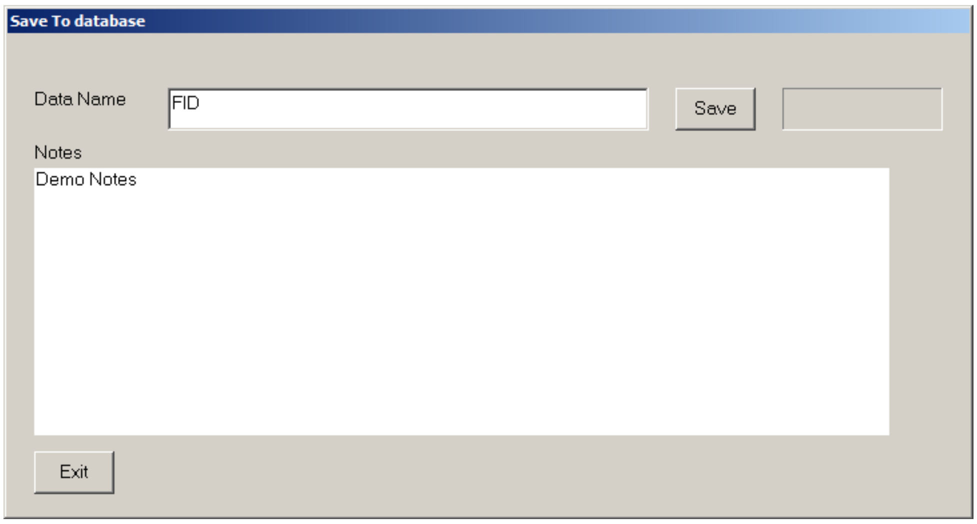
Chosen settings: Line [✓], Live [✓].

The current parameters for this data can be viewed by typing LP.

To save this data to the database, type SAVE at the OpenBox command line.

SAVE

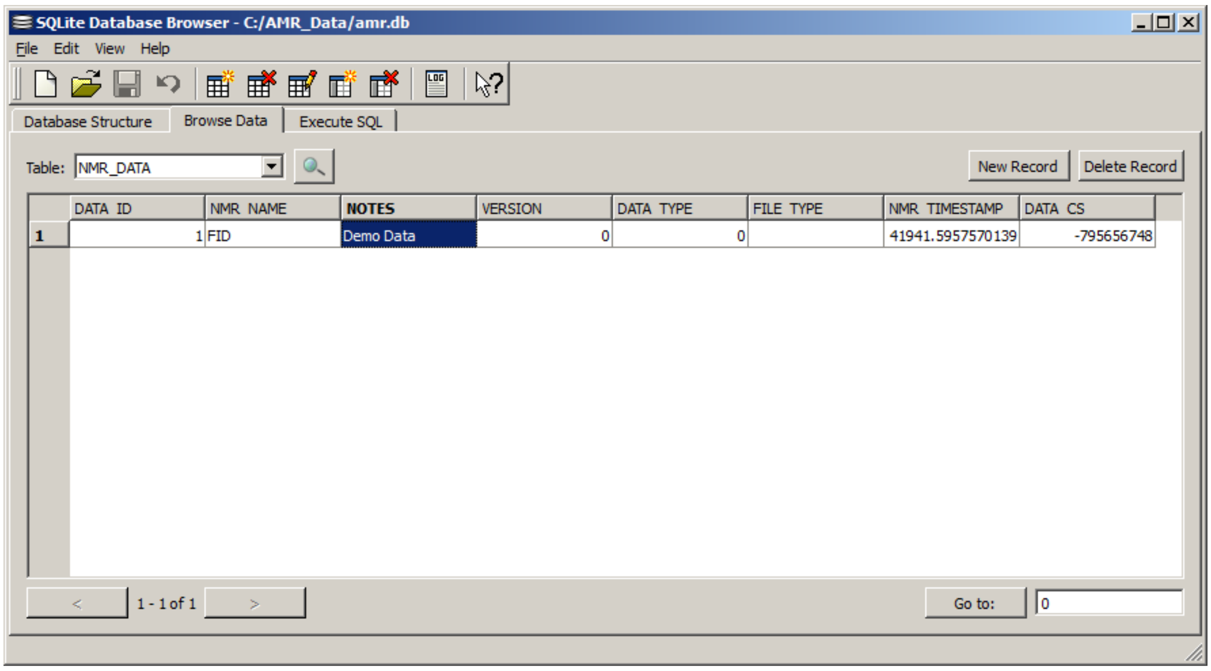
The SAVE dialog prompts for both a data file name, plus notes to be associated with the data file. Type FID for the filename and “Demo Notes” in the dialog box before saving the data.



Data Name: FID

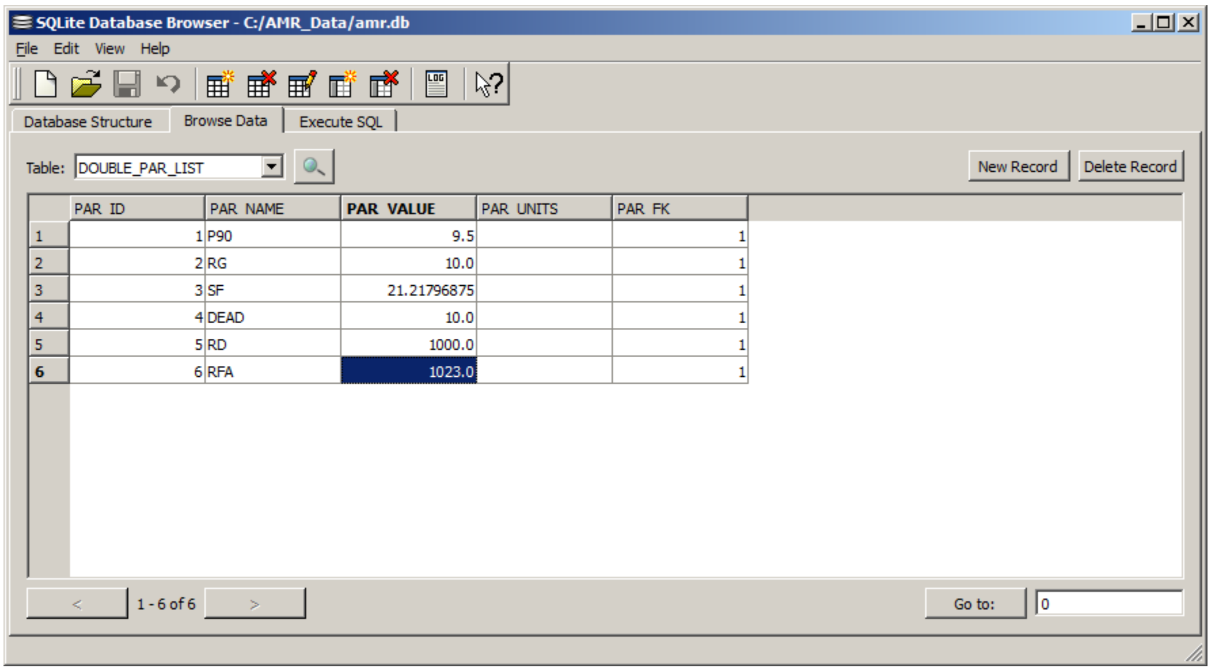
Notes: Demo Notes

Once the data has been saved to the database, it can be viewed with the browser. Select the NMR_Data table in the database structure and click on the browse data tab to view the contents of the NMR_Data table. The FID data that was saved earlier can be seen, along with the notes :



	DATA_ID	NMR_NAME	NOTES	VERSION	DATA_TYPE	FILE_TYPE	NMR_TIMESTAMP	DATA_CS
1	1	FID	Demo Data	0	0		41941.5957570139	-795656748

Click on the Double_Par_List data table to view the double parameters associated with the data set :



	PAR_ID	PAR_NAME	PAR_VALUE	PAR_UNITS	PAR_FK
1	1	P90	9.5		1
2	2	RG	10.0		1
3	3	SF	21.21796875		1
4	4	DEAD	10.0		1
5	5	RD	1000.0		1
6	6	RFA	1023.0		1

The double parameters saved include the P90, the receiver gain RG, the Spectrometer frequency, the dead time, the repetition/relaxation delay and the RF pulse amplitude. Each parameter may have units associated with it. The Par FK field holds the data set ID that these parameters belong to. This can be seen in the data ID table. Explore the other tables to see doubles, longs, environment and history parameters associated with the FID data set.

Repeat the process with the CPMG data set (LOAD CPMG, LOADASCII CPMG) to load the CPMG data into the data base and SAVE to save it. Check the data is present (DATA_ID 2) using the database browser.

Accessing the DataBase Directly Using ThinBasic

The database operates via a DLL, which any program capable of accessing a DLL can use. A description of the DLL API is attached to the end of this document. In addition a ThinBasic include file (sqllink.inc) contains the routines for calling the database from ThinBasic.

AMR Ltd has a number of example files that show how to call the database DLL from a wide range of languages, including C++, Python, Excel and ThinBasic. Contact AMR Ltd for more advice regarding calling the NMR database from these applications.

DEEPER Demonstration Database

The DEEPER Demonstration Database contains a demonstration database holding data from multiple modalities for 20 rock core samples. Modalities available include BSEI images, NMR data, XRD data, MICP data and XRF data. Download the deeper demonstration file and install to view this database. The demonstration database installs to the C:\AMR_Demo directory.

Saving Database Files for Use With DEEPER

The DEEPER data processing and analysis software can read files from the data base for processing. The DEEPER data browser works at a collection level, ie only collections of data are displayed. Therefore it is necessary to save the data to a database data file, then add that data to a collection in order for it to be visible in the browser.

The COLLSAVE command can be used to perform this task from the OpenBox command prompt, replacing the SAVE command described earlier.

Database API Description

1. **SQLNMR.DLL**

This DLL must be used together with OpenNMR. It accesses all the NMR data and parameters to simplify saving NMR data to the database. SQLNMR is normally located in the c:\OpenNMR\bin directory.

FUNCTION AMR_SimpleSave(Name:PChar):LONGINT; STDCALL;

> The current NMR data and parameters are stored in the database and "Name" is used to label the entry. Note that no check is done to see if "Name" has already been used. It is perfectly legal to have multiple entries with the same name.
>
> The function returns a unique Database ID which can later be used to access the data.
>
> Currently the database has to be "c:\AMR_Data\AMR.DB"

```
FUNCTION AMR_SaveWithNotes(Name:PChar;
                           RawNotes:PChar)
                           :LONGINT; STDCALL;
```

> This function works the same as AMR_SimpleSave except that it allows notes to be added.

2. **SQLLINK.DLL**

This DLL allows data and parameters to be read from the Database. To do so you must use the unique ID that defines a particular dataset. This ID is either obtained by storing it's value when the data was originally stored (for example by "AMR_SimpleSave") or by searching the Database.

```
FUNCTION  AMR_SearchByName(Name:PChar;
             Match:LONGINT;
             Results:PChar;
             IDs:PLongint;
             Timestamps:PDouble;
             Limit:LONGINT)
             :LONGINT; STDCALL;
```

Name	Name given to dataset (NOT case sensitive)
Match	NAME_EXACT = 0
	NAME_STARTS = 1
	NAME_CONTAINS = 2
Results	Pointer to character buffer
IDs	Pointer to array of 32 bit integers
Timestamps	Pointer to array of doubles
Limit	Maximum results allowed.

When calling this function "Limit" must be set to the maximum number of results the calling software can handle. The "IDs" and "Timestamps" arrays must then have a dimension of at least "Limit".

Determining the required size of "Results" is more difficult. This array is used to receive all the matching data names found, with each name separated by CR/LF and the complete list terminated with a NULL character. As a rule of thumb setting the size of the buffer to 128*Limit should be sufficient.

The function returns the number of matching entries found. If too many entries are found then a negative value is returned.

```
FUNCTION  AMR_SearchNotes(NoteText:PChar;
              Results:PChar;
              IDs:PLongint;
              Timestamps:PDouble;
              Limit:LONGINT)
              :LONGINT; STDCALL;
```

This works in the same way as AMR_SearchByName except that there is no "Match" parameter because the test is always if the Notes "contain" the given string.

```
PROCEDURE AMR_SetDateLimits(DateStart,DateEnd:DOUBLE); STDCALL;
```

This sets two variables in the DLL called "DateLimitStart" and "DateLimitEnd". When these have been set then calls to "AMR_SearchByName" will only return results when the timestamp of the dataset is between the two limits. To turn off this matching set both dates to -1.

```
FUNCTION AMR_GetDataSize(ID:LONGINT;
                VAR Size:LONGINT)
                :LONGINT; STDCALL;
```

Returns the number of NMR data points of dataset “ID” in the “Size” parameter. The function returns -1 (ERR_ILLEGAL_ID) if the dataset does not exist.

```
FUNCTION AMR_GetData(ID:LONGINT;
              A,B,X:PDouble;
              Max:LONGINT)
              :LONGINT; STDCALL;
```

Returns the NMR data in “A” and “B” and the time values in “X”. These variables are pointers to arrays of doubles and “Max” defines how big these arrays are. The function returns the number of points actually read, or a negative value for errors. Possible values are -1 (ERR_ILLEGAL_ID) or -15 (ERR_CRC_ERROR).

```
FUNCTION AMR_GetName(ID:LONGINT;
              Name:PChar)
              :LONGINT; STDCALL;
```

Returns the name of the dataset in “Name” given it’s ID. Returns -1 (ERR_ILLEGAL_ID) id there is an error.

```
FUNCTION AMR_GetNameSeries(ID:LONGINT;
                 Name:PChar)
                :LONGINT; STDCALL;
```

This works the same as AMR_GetName except that it applies to a series.

```
FUNCTION AMR_GetNameCollection(ID:LONGINT;
                   Name:PChar)
                  :LONGINT; STDCALL;
```

This works the same as AMR_GetName except that it applies to a collection.

```
FUNCTION AMR_GetNMRParameter(ID:LONGINT;
                 Index:LONGINT;
                 Name:PChar;
                 Value:PChar;
                 VAR ParType:CHAR
                 ):LONGINT; STDCALL;
```

Returns an NMR parameter from dataset “ID”. “Index” is used to select each parameter in turn, starting with Index=1. The function returns either the Index or -14 (ERR_END_LIST).

```
FUNCTION  AMR_AddNotes(ID:LONGINT;
             Notes:PChar)
                         :LONGINT; STDCALL;
```

This adds notes to dataset “ID”

```
FUNCTION  AMR_GetNotes(ID:LONGINT;
             Notes:PChar)
                         :LONGINT; STDCALL;
```

This returns the notes associated with dataset “ID”

```
FUNCTION  AMR_Delete(ID:LONGINT):LONGINT; STDCALL;
```

Deletes dataset “ID”

```
FUNCTION AMR_GetNMRParameterByName(ID:LONGINT;
                    Name:PChar;
                    Value:PChar;
                    VAR ParType:CHAR
                    ):LONGINT; STDCALL;
```

The name of the parameter required is passed to this function in “Name”. The value of the parameter and it’s type are returned in “Value” and “ParType”. If the parameter does not exist then -13 (ERR_NOT_FOUND) is returned.

```
FUNCTION AMR_GetNMREnvironment(ID:LONGINT;
                    Index:LONGINT;
                    Name:PChar;
                    Value:PChar
                    ):LONGINT; STDCALL;
```

This works in exactly the same way as AMR_GetNMRParameter except that it works with the NMR Environment variables

---

```
FUNCTION AMR_GetNMRHistory(ID:LONGINT;
                 Index:LONGINT;
                 Value:PChar
                 ):LONGINT; STDCALL;
```

Works in a similar way to AMR_GetNMRParameter and returns the processing history one line at a time.

```
FUNCTION AMR_Search:LONGINT;  STDCALL;
```

This launches a new program called "datasearch.exe". This allows the user to interact with the database and select a dataset. The function returns the ID of the selected data set, or 0 (zero) if no dataset was selected.

```
FUNCTION AMR_NewSeries(Name:PChar;
               SeriesType:PChar)
               : LONGINT STDCALL;
```

Creates a new series. Name can be up to 128 characters and SeriesType up to 32 characters. The function returns the ID of the series created.

```
FUNCTION AMR_AddToSeries(SeriesID,DataID:LONGINT;
                Value:DOUBLE)
                :LONGINT STDCALL;
```

Adds a dataset to a series. Use the SeriesID returned by a call to AMR_Newseries, and DataID returned by the call to AMR_SimpleSave or AMR_SaveWithNotes.

Value is a parameter which varies for each entry in the series.

This function returns the series index of the added dataset.

**FUNCTION AMR_GetSeriesIndex(SeriesID,Index:LONGINT;
Value: PDouble)
:LONGINT STDCALL;**

This function is used to step through all the entries in a series. SeriesID identifies which series to access and Index which item in the list. The fuction returns the ID of the dataset associated with element "Index" in the list. In addition Value is updated with double precision number associated with the returned dataset.

If Index is too high then -14 (ERR_END_LIST) is returned.

**FUNCTION AMR_AddSeriesNotes(SeriesID:LONGINT;
Notes:PChar)
: LONGINT STDCALL;**

Allow notes to be added to a series.

**FUNCTION AMR_SeriesSearchByName(Name:PChar;
Match:LONGINT;
Results:PChar;
IDs:PLongint;
Timestamps:PDouble;
Types:PChar;
Limit:LONGINT)
:LONGINT; STDCALL;**

This works in the same way as AMR_SearchByName except that the series type is returned in Types.

**FUNCTION AMR_SeriesGetNotes(ID:LONGINT;
Notes:PChar)
:LONGINT; STDCALL;**

This works in the same way as AMR_GetNotes.

FUNCTION AMR_SeriesSearch:LONGINT; STDCALL;

This works in the same way as AMR_Search. A zero is retuned if something other than a series is selected.

FUNCTION AMR_DeleteSeries(ID:LONGINT;
DeleteData:LONGINT)
:LONGINT; STDCALL;

This function deletes the series “ID” . If “DeleteData” is non-zero then the datasets listed in the series are also deleted.

FUNCTION AMR_GetAllDoubles(ID:LONGINT;
Results:PChar;
DoublePars:PDouble)
: LONGINT; STDCALL;

This function returns all the double precision floating point variables stored with a dataset. ID refers to the required dataset. If either Results or DoublePars is NIL then this function simply returns the number of variables found. To actually receive the names and values of the parameters pointers to 2 arrays have to be passed in Results and DoublesPars. The calling software must make sure the two arrays are large enough.

Normally this routine is called twice. The first time with NIL arguments so that the number of parameters is obtained. The calling software will then allocate an array of double precision numbers at least as large as the value returned by the first call. The size needed for Results is more complicated since the name for each parameter may have a different length. However parameter names are limited to 32 characters so allowing 34*FunctionReturn will be more than enough. (32 + CR + LF).

The list of parameter names is returned as one long string with each name terminated by a CR/LF pair.

FUNCTION AMR_GetAllDoublesAndUnits(ID:LONGINT;
Results:PChar;
DoublePars:PDouble;
Units:PChar)
: LONGINT; STDCALL;

This works in the same was as AMR_GetAllDoubles except that it takes an extra string argument. All the units associated with each parameter are returned as one long string with each unit separated by a CR/LF pair.

```
FUNCTION  AMR_GetAllLongs(ID:LONGINT;
                 Results:PChar;
                 LongPars:PLongint)
                 : LONGINT; STDCALL;
```

This works in the same way as AMR_GetAllDoubles except that the Integer parameters are returned.

```
FUNCTION  AMR_GetAllLongsAndUnits(ID:LONGINT;
                      Results:PChar;
                      LongPars:PLongint;
                      Units:PChar)
                     : LONGINT; STDCALL;
```

See AMR_GetAllDoublesAndUnits

```
FUNCTION  AMR_GetAllStringsAndUnits(ID:LONGINT;
                        Results:PChar;
                        LongPars:PLongint;
                        Units:PChar)
                      : LONGINT; STDCALL;
```

See AMR_GetAllDoublesAndUnits

```
FUNCTION  AMR_GetAllStrings(ID:LONGINT;
                  Results:PChar;
                  StringPars:PChar)
                  : LONGINT; STDCALL;
```

This works in a similar way to GetAllDoubles. The difference is that both Results and StringPars are returned as single strings with each value separated by CR/LF pairs.

FUNCTION AMR_GetDoubleParFromList(IDList:PLongint;
ListSize:LONGINT;
Par:PChar;
FoundIDs:PLongint;
FoundDoubles:PDouble
):LONGINT; STDCALL;

This routine is used to retrieve all the values of a double precision parameter associated with a list of Dataset IDs. IDList is an array of IDs and ListSize defines the size of the list. Par is the name of the parameter to search for.

The results are returned in two more arrays. FoundIDs is the list of Dataset IDs found and the value for each DataSet is returned in FoundDoubles. Note that the number of values returned may ***not*** be the same as the size of the list passed to this routine. It will be smaller if a parameter named in Par does not occur in all the Datasets, or in the same ID occurs more than once in the list. The function returns the number of values actually found, or a negative error code.

FUNCTION AMR_GetLongParFromList(IDList:PLongint;
ListSize:LONGINT;
Par:PChar;
FoundIDs:PLongint;
FoundIntegers:PLongint
):LONGINT; STDCALL;

This works he same way as AMR_GetDoubleParFromList except that it works with the integer parameters.

FUNCTION AMR_GetStringParFromList(IDList:PLongint;
ListSize:LONGINT;
Par:PChar;
FoundIDs:PLongint;
FoundStrings:PChar
):LONGINT; STDCALL;

This works he similar way to AMR_GetDoubleParFromList. The strings are returned as one long string with each string value terminated by a CR/LF pair. The software calling this routine needs to make sure that the buffer pointed to by FoundStrings is large enough.

FUNCTION AMR_DefineDatabase(DBDir:PChar):LONGINT; STDCALL;

This function allows the default database (c:\AMR_Data) to be overridden. DBDir must contain the full path of the directory that contains the actual database file *amr.db* .

FUNCTION AMR_SaveDataToDatabase(Name:PChar;
Notes:PChar;
DataType:LONGINT;
DataA:PDouble;
DataB:PDouble;
DataX:PDouble;
DataSize:LONGINT):LONGINT;STDCALL;

This saves data to the main table in a database. Name can be up to 128 characters and Notes up to 1024. The actual data is pointed to by the three variables DataA, DataB and DataX, and the size of the data must be passed in DataSize. DataType can be used to define what sort of data is being saved. Currently this is always zero.

As well as the parameters defined above there is other information that is always saved automatically. These are a version number, a timestamp and a checksum for DataA and DataB.
The function returns an ID which can be used in other calls to SQLLINK.

FUNCTION AMR_AddNMRParameter(ID:LONGINT;
Name:PChar;
Value:PChar;
ParType:CHAR
):LONGINT; STDCALL;

This allows a parameter to be added to the database. ID is obtained from the call to AMR_SaveDataToDatabase. Name is the name of the parameter to be added. Each parameter is passed as a string in Value. ParType must be “D”, “S” or “L” for *double*, *string* or *longint.* ERR_ADDING_PAR (-17) is returned if an error occurs.

FUNCTION AMR_AddNMREnvironment(ID:LONGINT;
Name:PChar;
Value:PChar
):LONGINT; STDCALL;

This function works is a similar way to AMR_AddNMRParameter except that ParType is not needed. ERR_ADDING_ENV (-18) is returned if an error occurs.

FUNCTION AMR_AddNMRHistory(ID:LONGINT;

Value:PChar
):LONGINT; STDCALL;

This adds one line of processing history to the database. ERR_ADDING_HIST (-19) is returned if an error occurs.

FUNCTION AMR_DatabaseName(DBDir:PChar):LONGINT; STDCALL;

This allows a different database to be accessed. By default C:\AMR_Data\ is used. This function can be used in one of 3 ways.

If DBDir is an empty string then the current database directory is returned in DBDir. Make sure that there is enough space for the returned string. 1024 bytes should be more than enough.

If DBDir ends in ".DB" then a new directory path and filename is defined.

If DBDir does ***NOT*** end in ".DB" then just a new directory path is defined and the actual database file will be "AMR.DB"

Calling this function defines a new root folder for the database.
ERR_DB_NOT_FOUND (-16) is returned if the database does not exist.

FUNCTION AMR_NewCollection(Name:PChar;
CollectionType:PChar):
LONGINT STDCALL;

This works in the same way as AMR_NewSeries.

FUNCTION AMR_AddCollectionNotes(CollectionID:LONGINT;
Notes:PChar)
: LONGINT STDCALL;

This works in the same way as AMR_AddSeriesNotes.

```
FUNCTION AMR_AddToCollection(CollectionID,
                                          ItemID,
                  ItemType:LONGINT)
                  : LONGINT STDCALL;
```

This is used to add one item to a collection. “CollectionID” is the value obtained when “AMR_NewCollection” is called. “ItemID” can be the ID of a DataSet, Series, or another Collection. “ItemType” can be:-

ITEM_DATA	= 1
ITEM_SERIES	= 2
ITEM_COLLECTION	= 3

Items are added to the “Item_List” table and the function returns the ID of the item in that table. A value <=0 indicates an error.

FUNCTION AMR_CollectionSearch:LONGINT; STDCALL;

This works in the same way as AMR_Search. A zero is retuned if something other than a collection is selected.

```
FUNCTION AMR_CollectionSearchByName(Name:PChar;
                       Match:LONGINT;
                       Results:PChar;
                       IDs:PLongint;
                       Timestamps:PDouble;
                       Types:PChar;
                       Limit:LONGINT)
                      :LONGINT; STDCALL;
```

This works in the same way as “AMR_SeriesSearchByName”

```
FUNCTION AMR_GetCollectionList(ID:LONGINT;
                 ItemLinks:PLongInt;
                 ItemTypes:PLongInt;
                 Limit:LONGINT):LONGINT; STDCALL;
```

This function is used to retrieve all the items in a collection. “ID” is the value returned by “AMR_CollectionSearchByName”. “ItemLinks” and “ItemsTypes are pointers to arrays of LONGINTS. “Limit” defines how large these array are.

Each value in “ItemsLinks” is a value used to access the tables NMR_DATA, SERIES or COLLECTION. “ItemTypes” defines which of the tables should be accessed, with the constants being the same as used with “AMR_AddToCollection”. The two arrays should be considered as providing pairs of numbers which together allow each item of the collection to be retrieved.

The function returns the number of items in the collection.

```
FUNCTION  AMR_CollectionGetNotes(ID:LONGINT;
                  Notes:PChar)
                  :LONGINT; STDCALL;
```

This works in the same way as AMR_GetNotes.

```
FUNCTION AMR_DeleteCollection(ID:LONGINT):LONGINT; STDCALL;
```

This is used to delete a Collection. Note that none of the items that make up the collection are deleted; just the container.

```
FUNCTION AMR_SQLCommand(Command:PChar;
                        Reply:PChar;
                        Limit:LONGINT):
         LONGINT; STDCALL;
```

This function allows any SQL command to be executed. The command is passed in"Command" and the results are returned in "Reply". The size of the Reply buffer is passed in "Limit".

All the results are returned as one long string which each record separated by <CR/LF>. Each field within a record is delimited by a comma.

For example if "Command" is set to:-

```
select par_value,par_fk from double_par_list where par_name like "tau"
```

then the string returned in "Reply" will be like this:-

```
600.0,12
600.0,13
600.0,14
600.0,15
1000.0,120
1000.0,121
1000.0,122
2000.0,123
2000.0,124
10000.0,125
10000.0,126
```

Each line contains two values. The first is the value of TAU and the second is the NMR dataset ID.

The value returned by this function is the size of the string if there were no errors. A negative value indicates than an error occurred.

```
ERR_SQL_QUERY  ( -2)
ERR_TOO_MANY_RESULTS ( -104)
```

Other error codes may be added so the check that the call worked OK should be that the result is a number >0 .

```
FUNCTION AMR_ConvertTimestamp(VAR ts      : DOUBLE;
                              VAR Year    : LONGINT;
                              VAR Month   : LONGINT;
                              VAR Day     : LONGINT;
                              VAR Hour    : LONGINT;
                              VAR Minute  : LONGINT;
                              VAR Sec     : LONGINT;
                              VAR MSec    : LONGINT
                              ):LONGINT;
```

This function converts between a floating point version of time and date, and a representation that uses 7 32 bit integers. The floating point version gives the time in days from 1900.

To convert ***from*** floating point to 7 integer values set “ts” to a positive number >=1900. On return the 7 variables will be set.

To convert ***to*** the floating pint version set “ts” to a –ve value and the 7 integer variables to the date and time you wish to set. On return “ts” will be set.

An error is indicated by a –ve result. Possible values are

ERR_SQL_QUERY (-2)
ERR_TIME_DATE (-21)

FUNCTION AMR_GetTimestamp(ID:LONGINT; VAR ts:DOUBLE):LONGINT;

This gets the timestamp for NMR dataset “ID”.

FUNCTION AMR_SetTimestamp(ID:LONGINT; ts:DOUBLE):LONGINT;

This sets the timestamp for NMR dataset “ID”. If “ts” is set to a –ve value then the current date and time are used.

```
FUNCTION AMR_AddNMRParameterAndUnits(ID:LONGINT;
                                     Name:PChar;
                                     Value:PChar;
                                     ParType:CHAR;
                                     Units:PChar
                                     ):LONGINT; STDCALL;
```

This works in the same way as AddNMRParameter except that it also allows Units to be defined.

```
FUNCTION AMR_AddUnitsToNMRParameter(ID:LONGINT;
                                    Name:PChar;
                                    ParType:Char;
                                    Units:PChar
                                    ):LONGINT; STDCALL;
```

This function allows units to be added or changed for an existing dataset. “ID” referes to the dataset. “Name” and “ParType” refer to the name and type of the parameter to be updated with new “Units”

```
FUNCTION AMR_UpdateNMR(ID:LONGINT;
                       Item:PChar;
                       Value:PChar)
                       :LONGINT; STDCALL;
```

Also:-

AMR_UpdateSeries
AMR_UpdateCollection
AMR_UpdateDouble
AMR_UpdateInteger
AMR_UpdateString
AMR_UpdateEnvironment
AMR_UpdateHistory

These functions all work in the same way. “ID” is the unique ID which identifies one entry in a table in the database. “Name” is the name of one field as defined by the database. These fields can be obtained by using the SQLite Database Browser. The updated value is passed in “Value”.

Examples:-

```
AMR_UpdateNMR(23,’DATA_TYPE’,’CPMG’);
AMR_UpdateSeries(4,’SERIES_NAME’,’Retest’);
AMR_UpdateDouble(327,’PAR_VALUE’,’23.78’);
```

Note that both “Item” and “Value” are strings, even when updating numeric values. The *Update** routines are simple layers on top of the underlying SQL commands. Any positive return value indicates that the function executed OK. Possible error return is ERR_SQL_QUERY (-2).

FUNCTION AMR_RetrieveNMR(ID:LONGINT;
Item:PChar;
Value:PChar;
Limit:LONGINT)
:LONGINT; STDCALL;

Also:-

AMR_RetrieveSeries
AMR_RetrieveCollection
AMR_RetrieveDouble
AMR_RetrieveInteger
AMR_RetrieveString
AMR_RetrieveEnvironment
AMR_RetrieveHistory

These functions all work in the same way. “ID” is the unique ID which identifies one entry in a table in the database. “Name” is the name of one field as defined by the database. The associated value currently stored in the database is retuned in “Value”. “Limit” defines the maximum size of the buffer that has been allocated receive “Value”. This must be less than 16K.

Examples:-

AMR_RetrieveCollection(223,’COLLECTION_TYPE’,ct,32);
AMR_RetrieveDouble(734,’PAR_UNITS’,MHz,32);

The function returns the length of the “Value” string. If an error occurs then a negative result is returned. Possible values are ERR_SQL_QUERY (-2), ERR_LIMIT (-4),ERR_TOO_BIG(-5).

FUNCTION AMR_DeleteNMRParameter(ID:LONGINT;Name:PChar)
:LONGINT; STDCALL;

This function deletes one parameter. “ID” refers to the dataset that the parameter is associated with, and “Name” is the name of the parameter, and is ***not*** case sensitive. A positive value is returned if there is no error. Note that if the ID/Name pair does not exist then this is ***not*** considered to be an error.

```
FUNCTION AMR_SaveFileToDatabase(Name:PChar;
                        Notes:PChar;
                        DataType:PChar;
                        Filename:PChar):LONGINT; STDCALL;
```

This function allows any file to be saved to the database. “Name” is the name used to identify the dataset. “Filename” should contain the complete path and name of the file to be save. The function automatically extracts the file extension and stores this in the FILE_TYPE field. If there is no file extension then “.” Is stored.

The function returns the unique DATA_ID if successful. Possible errors include ERR_FILE_LOAD (-6)

```
FUNCTION AMR_GetFileFromDatabase(ID:LONGINT;
                        Filename:PChar):LONGINT; STDCALL;
```

This retrieves the file stored in stored in the database. It will create the file defined by “Filename” and copy the original file into it. If “Filename” has an extension then it will be used. If no extension is given then the value in the “FILE_TYPE” field is used.

Possible errors include ERR_ILLEGAL_ID (-1),ERR_FILE_LOAD (-6), ERR_CRC_ERROR (-15).

For ERR_CRC_ERROR the file is created anyway and the front end software should decide what should be done with it.

```
FUNCTION AMR_SelectiveSearch(Selection:PChar;
                      VAR SelectedType:CHAR):LONGINT;
```

This allows a selective search of the database. “Selection” can contain the letters ‘D’, ‘S’ and ‘C’. Other letters are ignored. If the string contains a ‘D’ then searches on data sets are allowed. Similarly ‘S’ and ‘C’ permit searches on series and collections. If “Selection” is empty then the the function behaves as if the argument “DSC” was given, i.e. all searches are allowed.

The function returns the ID of the database element selected. A zero indicates no selection was made. If the returned value is non-zero then SelectedType is set to ‘D’, ‘S’ or ‘C’ to indicate which table was selected.

FUNCTION AMR_CreateDatabase(Name:PChar):LONGINT; STDCALL;

This function enables new, empty, databases to be created. The single argument is the name of the database to be created. There are some rules which must be followed.

Name = 'c:\temp\database_1'

The folder 'c:\temp' must already exist but the subfolder 'database_1' must NOT exist.
The function will create 'database_1' and 'database_1\files' and generate a new database file named 'amr.db'

Name = 'c:\temp\database_2\newdatabase.db'

The folder 'c:\temp' must already exist but the subfolder 'database_2' must NOT exist.
The function will create ' database_2' and ' database_2\files' and generate a new database file named 'newdatabase.db'

The function returns a -ve number if there is an error. Likely values are :

ERR_ALREADY_EXISTS	-23
ERR_CREATE_DIR	-24

; Example Excel Function Calls for Accessing SQLLINK.DLL. note that the “_” underscore character will be required to signify to the VBA development environment that the text continues to another line.

More information on using the AMR DataBase from Excel and examples are available from AMR Ltd.

```
--

Public Declare Function AMR_DatabaseName Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal databasename As String) As Long

--

Public Declare Function AMR_SaveDataToDatabase Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal Name As String, ByVal Notes As
String, ByVal datatype As String, ByRef DataA As Double, ByRef DataB
As Double, ByRef DataX As Double, ByVal Datasize As Long) As Long

‘Note call as below

Dim Name As String
Dim Notes As String
Dim datatype As String
Dim Total_points As Long

Dim DataA(65535) As Double
Dim DataB(65535) As Double
Dim DataX(65535) As Double

Dim Data_ID As Long

Data_ID = AMR_SaveDataToDatabase(Name, Notes, datatype, DataA(1),
DataB(1), DataX(1), Total_points)

--

Public Declare Function AMR_AddNMRParameterAndUnits Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal ID As Long, ByVal Name As String,
ByVal Value As String, ByVal ParType As Long, ByVal Units As String)
As Long

‘Note call as below

Dim addpar_error as Long
Dim data_ID As Long
Dim par_name As String
Dim par_value As String
Dim lvalue As Long
Dim par_units As String

‘lvalue (type of parameter to add) must be established by the
following function (where “S”=string, ”D”=double and “L”=long)

Lvalue=ASC(“S”)

addpar_error = AMR_AddNMRParameterAndUnits(Data_ID, par_name,
par_value, lvalue, par_units)

--
```

```
Public Declare Function AMR_NewCollection Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal Name As String, ByVal
collection_type As String) As Long

--

Public Declare Function AMR_AddToCollection Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal CollectionID As Long, ByVal ItemID
As Long, ByVal ItemType As Long) As Long

--

Public Declare Function AMR_AddCollectionNotes Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal CollectionID As Long, ByVal Notes
As String) As Long

--

Public Declare Function AMR_GetData Lib "c:\OpenNMR\bin\sqllink.dll"
(ByVal ID As Long, ByRef DataA As Double, ByRef DataB As Double, ByRef
DataX As Double, Max As Long) As Long

--

Public Declare Function AMR_GetDataSize Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal ID As Long, ByRef size As Long) As
Long

--

Public Declare Function AMR_UpdateNMR Lib "c:\OpenNMR\bin\sqllink.dll"
(ByVal DataID As Long, ByVal Field As String, ByVal field_value As
String) As Long

--

Public Declare Function AMR_SQLCommand Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal SQL_Command As String, ByVal Reply
As String, ByVal Limit As Long) As Long

--

Public Declare Function AMR_GetFileFromDataBase Lib
"c:\OpenNMR\bin\sqllink.dll" (ByVal ID As Long, ByVal Filename As
String) As Long
```

; Thinbasic function calls for accessing SQLLINK.DLL. note that the “_” underscore character is required to signify to thinair that the text continues to another line.

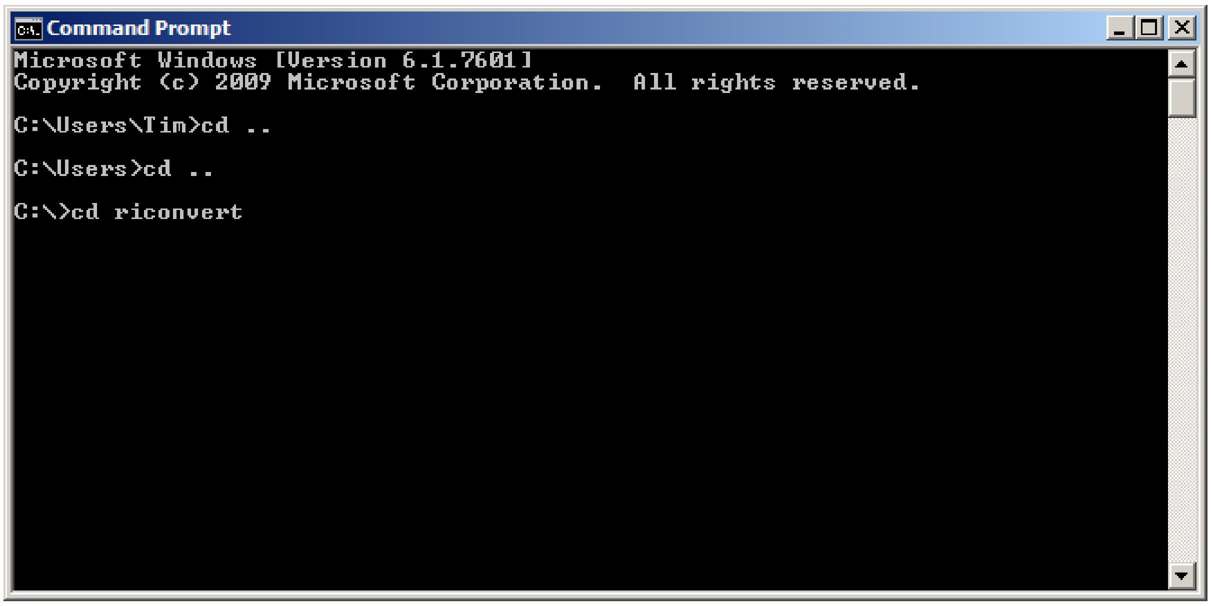
```
'Functions for accessing SQLLINK.DLL.

$LIBSQLNMR = "c:\OpenNMR\bin\sqllink.DLL"
$LIBSQLNMR2 = "c:\OpenNMR\bin\SQLNMR.DLL"

Declare Function AMR_AddNMRParameter Lib $LIBSQLNMR Alias "AMR_AddNMRParameter" (ByVal
ID As Long, ByVal ParName As Asciiz, ByVal ParValue As Asciiz, ByVal partype As Long) As
Long

Declare Function AMR_NewSeries Lib $LIBSQLNMR Alias "AMR_NewSeries" (ByVal SeriesName As
Asciiz,ByVal SeriesType As Asciiz) As Long

Declare Function AMR_AddToSeries Lib $LIBSQLNMR Alias "AMR_AddToSeries" (ByVal Series_ID
As Long, ByVal Data_ID As Long, ByVal Series_Value As Double) As Long

Declare Function AMR_AddNMREnvironment Lib $LIBSQLNMR Alias "AMR_AddNMREnvironment"
(ByVal ID As Long, ByVal ParName As Asciiz, ByVal ParValue As Asciiz) As Long

Declare Function AMR_GetAllDoubles Lib $LIBSQLNMR Alias "AMR_GetAllDoubles" (ByVal ID As
Long, ByRef StringParameters As Asciiz, ByRef FoundDoubles As Double) As Long

Declare Function AMR_GetAllLongs Lib $LIBSQLNMR Alias "AMR_GetAllLongs" (ByVal ID As
Long, ByRef StringParameters As Asciiz, ByRef FoundLongs As Long) As Long

Declare Function AMR_GetAllStrings Lib $LIBSQLNMR Alias "AMR_GetAllStrings" (ByVal ID As
Long, ByRef StringParameters As Asciiz, ByRef FoundStrings_comp As Asciiz) As Long

Declare Function AMR_GetDataSize Lib $LIBSQLNMR Alias "AMR_GetDataSize" (ByVal ID As
Long, ByRef Size As Long) As Long

Declare Function AMR_GetData Lib $LIBSQLNMR Alias "AMR_GetData" (ByVal ID As Long, ByRef
DataA As Double, ByRef DataB As Double, ByRef DataX As Double, ByVal Maxpoints As Long)
As Long

Declare Function AMR_NewCollection Lib $LIBSQLNMR Alias "AMR_NewCollection" (ByVal
CollectionName As Asciiz,ByVal CollectionType As Asciiz) As Long

Declare Function AMR_AddToCollection Lib $LIBSQLNMR Alias "AMR_AddToCollection" (ByVal
Collection_ID As Long, ByVal Data_ID As Long, ByVal Collection_Value As Long) As Long

Declare Function AMR_CollectionGetNotes Lib $LIBSQLNMR Alias "AMR_CollectionGetNotes"
(ByVal ID As Long, ByRef Collection_notes As Asciiz) As Long

Declare Function AMR_SaveWithNotes Lib  $LIBSQLNMR2 Alias "AMR_SaveWithNotes" (ByRef
AName As Asciiz, ByRef Notes As Asciiz) As Long

Declare Function AMR_AddCollectionNotes Lib $LIBSQLNMR Alias "AMR_AddCollectionNotes"
(ByVal CollectionID As Long, ByRef Notes As Asciiz) As Long

Declare Function AMR_CollectionGetNotes Lib $LIBSQLNMR Alias "AMR_CollectionGetNotes"
(ByVal CollectionID As Long, ByRef Notes As Asciiz) As Long

Declare Function AMR_DeleteNMRParameter Lib $LIBSQLNMR Alias
“AMR_DeleteNMRParameter"(ByVal DATA_ID As Long, ByVal ParName As Asciiz) As Long

Declare Function AMR_AddNMRHistory Lib $LIBSQLNMR Alias "AMR_AddNMRHistory" (ByVal ID As
Long, ByVal SeriesName As Asciiz) As Long

Declare Function AMR_GetDoubleParFromList Lib $LIBSQLNMR Alias
"AMR_GetDoubleParFromList" (ByRef IDList As Long, ByVal ListSize As Long, ByVal Par_name
As Asciiz, ByRef FoundIDS As Long, ByRef FoundDoubles As Double) As Long


Declare Function AMR_GetNMRParameter Lib $LIBSQLNMR Alias "AMR_GetNMRParameter" (ByVal
ID As Long,ByVal Index As Long, ByRef par_name As Asciiz, ByRef par_value As Asciiz,
ByRef par_type As Long) As Long

Declare Function AMR_SaveDataToDatabase Lib $LIBSQLNMR Alias "AMR_SaveDataToDatabase"
(ByVal DataName As Asciiz, ByRef Notes As Asciiz, ByVal Datatype As Long, ByRef DataA As
Double, ByRef DataB As Double, ByRef dataX As Double, ByVal Data_size As Long) As Long
```

```
Declare Function AMR_SearchByName Lib $LIBSQLNMR Alias "AMR_SearchByName" (ByRef PName
As Asciiz, ByVal Match As Long, ByRef Results As Asciiz, ByRef IDs As Long, ByRef
TimeStamps As Double, ByVal limit As Long)As Long

Declare Function AMR_SaveWithNotes Lib $LIBSQLNMR2 Alias "AMR_SaveWithNotes" (ByRef
PName As Asciiz, ByRef PNotes As Asciiz ) As Long

Declare Function AMR_AddCollectionNotes Lib $LIBSQLNMR Alias "AMR_AddCollectionNotes"
(ByVal CollectionID As Long, ByRef Notes As Asciiz) As Long

Declare Function AMR_Search Lib $LIBSQLNMR Alias "AMR_Search" () As Long

Declare Function AMR_GetNMREnvironment Lib $LIBSQLNMR Alias "AMR_GetNMREnvironment(ByVal
ID As Long, ByVal Index As Long, ByRef AName As Asciiz, ByRef AValue As Asciiz) As Long

Declare Function AMR_GetName Lib $LIBSQLNMR Alias "AMR_GetName" (ByVal ID As Long, ByRef
Aname As Asciiz) As Long

Declare Function AMR_GetNameCollection Lib $LIBSQLNMR Alias "AMR_GetNameCollection"
(ByVal ID As Long, ByRef Aname As Asciiz) As Long

Declare Function AMR_GetCollectionList Lib $LIBSQLNMR Alias "AMR_GetCollectionList"
(ByVal ID As Long, ByRef ItemLinks As Long, ByRef ItemTypes As Long, ByVal Limit As
Long) As Long

Declare Function AMR_AddNMRParameterAndUnits Lib $LIBSQLNMR Alias
"AMR_AddNMRParameterAndUnits"(ByVal ID As Long, ByVal Parname As Asciiz, ByVal ParValue
As Asciiz, ByVal partype As Long, ByVal UnitsName As Asciiz) As Long

Declare Function AMR_AddUnitsToNMRParameter Lib $LIBSQLNMR Alias
"AMR_AddUnitsToNMRParameter" (ByVal ID As Long, ByVal Parname As Asciiz, ByVal partype
As Long, ByVal UnitsName As Asciiz) As Long

Declare Function AMR_GetNMRParameterByName Lib $LIBSQLNMR Alias
"AMR_GetNMRParameterByName" (ByVal ID As Long, ByVal parname As String, ByRef Parvalue
As Asciiz, ByRef Partype As Asciiz) As Long

Declare Function AMR_UpdateNMR Lib $LIBSQLNMR Alias "AMR_UpdateNMR" (ByVal ID As Long,
ByVal Field As String, ByVal value_string As String) As Long
```

RIDE

RIDE (Resonance Instruments Data Extraction) is a conversion tool which allows the user to convert and add .RiDat files to the AMR database.

In order to use RIDE, AMR Ltd recommends the user creates a directory C:\RICONVERT and copies the RIDE executable to this directory.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Tim>cd ..

C:\Users>cd ..

C:\>cd riconvert
```

Files can be converted using the command prompt. After selecting the RIDE directory the following options can be used :

```
ride filename
```

This appends .RIDAT to "filename" and creates "filename.amr"

Example:-

```
        ride fid.00001
```

```
ride filename outname
```

This appends .RIDAT to "filename" and creates "outname.amr", the AMR Ltd binary format.

Example:-

```
ride ultra.00001 olddatafile
```

```
ride filename /d
```

This appends .RIDAT to "filename" and creates a new entry in the default AMR database. A full NMR data entry is created together with parameters etc. but no collection. The input filename is used as the dataname in the database. Either /d or /D can be used.

Example:-

```
ride toluene.00001 /d
```

```
ride filename newname /d
```

This works as the example above except that "newname" is used when adding to the database.

Example:-

```
ride toluene.00001 H1_toluene /d
```

```
ride filename newname /dpath
```

This defines a different database to add the data to.

Example:-

```
ride drx.00001 sandstone /dc:\amr_rockcore
ride filename dataname  /d /c
```

In addition to creating a new NMR data entry a Collection is also created. "dataname" is used both the NMR data set and the Collection. This should be the format used for data that needs to be viewed with the DEEPER data analysis software

Example:-

```
ride ultra.00001  water /d /c
```

```
Ride filename dataname /d /cColName
```

This allows the collection name to be defined.

Example:-

```
ride ultra.00001  water /d /cCPMG_Collection
```

A default set of NMR parameters is always added to both the .AMR files and the database. An extra option /I can be used to define a folder which contains RINMR .info files. If this is done then ride will extract the sequence name from the .RIDAT file and then try and locate the corresponding .INFO file and use it to add extra parameters.

Example:-

```
ride ultra.00001  water /d /cCPMG_Collection /ic:\opennmr\riinfo
```

Known issues :

Spaces are not allowed in paths or names.

The timestamp added to the database is the current time, not the time that the data was acquired.