The QRZDLL Specification

Contents

About QRZDLL

QRZDLL was written by Fred Lloyd, AA7BQ, and introduced into the third volume of the QRZ! Ham Radio CDROM (June 1994). The code was developed to enable the use of higher level GUI interface development tools, such as Microsofts Visual Basic. QRZDLL itself is written in C and was compiled with Microsoft C version 7.0. The DLL can be used along with both C and Visual Basic applications. It is likely that it works with Visual C++, but this has not been tested.

If you are using Microsoft's Visual Basic and would like to get started right away, just have a look at the QRZDEMO.BAS application. This sample application, which took only about 10 minutes to write, shows just how easy it is to use QRZDLL. Be sure and include a copy of the GLOBAL.BAS file which contains all of the constants and declarations used by QRZDLL.

The goal of QRZDLL is to provide a consistent interface to the callsign database while maintaining some freedom to change its implementation. Future versions and enhancements of QRZDLL will be downward compatible with this specification..

If you have any comments about the spec, and/or suggestions for future enhancements please feel free to email me at: **aa7bq@qrz.com**

73 from Scottsdale,

-fred

Licensing Information

As the owner of this copy of the QRZ! Ham Radio CDROM and as an independent software developer, you are hereby granted a royalty-free, unlimited license to redistribute the QRZDLL.DLL library along with programs that you write provided that for each program that uses QRZDLL.DLL a message appears in the "About" box on your program's main menu which states:

Portions of this software courtesy of QRZ, Copyright (c) 1994

QRZSearch

int QRZSearch (*Mode, Key1, Key2, Key3, Found, Format*)

```
int Mode        /* Search Mode (see below) */
LPSTR Key1      /* Search Key 1 - depends on Mode */
LPSTR Key2      /* Search Key 2 - depends on Mode */
LPSTR Key3      /* Search Key 3 - depends on Mode */
LPSTR Found     /* Address of return buffer for result */
int Format      /* Return record format type (see below) */
```

The **QRZSearch** function is the primary entry to the callsign database. QRZSearch will locate the first occurence of a given key sequence and return a result in the buffer pointed to by *Found*. QRZSearch returns the number of bytes written to the *Found* buffer or 0 if no record matching the specified key(s) was found.

In addition to returning the data in the buffer Found, QRZDLL maintains a copy of the current record in memory which can be retrieved one field at a time using QRZField.

Parameter	Description

Mode Specifies the type of lookup to be performed. Must be one of:

QRZCALL	Do Callsign lookup
QRZNAME	Do Name lookup
QRZCITY	Do City/State lookup
QRZZIP	Do Zip Code lookup
QRZSTREET	Do Street/City/State lookup

Key1
Key2
Key3 Depending on ***Mode***, the keys are used for varying purposes:

For **QRZCALL**, Key1 is the callsign or callsign suffix, Key2 and Key3 are unused.

For **QRZNAME**, Key1 is the Last name, Key2 is the First name (with an optional space separated middle initial) and Key3 is unused. Key2 is optional but if used is considered wild. For example, when Key2 = "F", then all first names beginning with the letter "F" are returned. Two word first names (such as "John Paul") will fail as the database attempts to find a John P. Somebody. A quoting method will be introduced in a future release.

For **QRZCITY**, Key1 is the two-letter state code and Key2 is the city name. Key1 is mandatory. Key2 is optional, and may contain a trailing '*' or wildcard. Key3 is unused.

For **QRZZIP**, Key1 is the zip code to start with and Key2 is the zip code to end with. Key1 is mandatory and Key2 is optional. When only Key1 is specified, only those zip codes matching Key1 are returned. Key1 may not be greater than Key2 are equal to or greater than Key1 up to and including all zip codes which are equal to Key2. Key3 is not used.

For **QRZSTREET**, Key1 is the two-letter state code and Key2 is the city name (same rules as for QRZCITY above). Key3 contains the substring of the desired street name. For each record matching the QRZCITY criteria, a string search is performed on the

street address field, returning any records which contain Key3 in them. QRZSTREET is not indexed on the street names and so its performance will be noticably slower.

Found When a match is found, it is formatted according to *Fmt* and placed into the buffer pointed to by *Found*.

Format The supported format types which are returned in the *Found* buffer are:

DISP_FMT	Display Format (all data)
MAIL_FMT	Mailing Label Format (partial data)
BOOK_FMT	Callbook Style Format (partial data)
RAW_FMT	Raw Format (full data)
DBF_FMT	DBF Format - Quote/comma delimited full data

Notes: The ***Mode*** parameter selects one of the 4 presorted databases making it the default database for all other functions in the library (the Street and City/State modes use the same database). The system-wide ***Mode*** setting can only be changed by another call to **QRZSearch**.

Constant Definitions: (as used in Visual Basic)

```
Const QRZCALL = 1     ' Modes
Const QRZNAME = 2
Const QRZCITY = 3
Const QRZZIP = 4
Const QRZSTREET = 5

Const DISP_FMT = 1    ' Formats
Const MAIL_FMT = 2
Const BOOK_FMT = 3
Const RAW_FMT = 4
Const DBF_FMT = 5
```

Visual Basic Declaration:

Declare Function QRZSearch Lib "qrzdll.dll" (**ByVal** *Mode* as Integer, **ByVal** *Key1* As String, **ByVal** *Key2* As String, **ByVal** *Key3* As String, **ByVal** *Found* As String, **ByVal** *Format* as Integer) As Integer

C Declaration:

int FAR PASCAL QRZSearch(int mode, LPSTR key1, LPSTR key2, LPSTR key3, LPSTR found, int format);

QRZGetNext

int QRZGetNext (*Found, Format*)

LPSTR ***Found*** /* Address of return buffer for result */
int ***Format*** /* Return record format type (see QRZSearch) */

The **QRZGetNext** function retrieves the next logical record matching the key sequence and **Mode** specified by a previous call to QRZSearch . The result is returned in the buffer pointed to by *Found*. **QRZGetNext** returns the number of bytes written to the buffer or 0 if no more records were found.

The *Found* and *Format* parameters are identical to those used by QRZSearch.

Visual Basic Declaration:

Declare Function QRZGetNext Lib "qrzdll.dll" (**ByVal** *Found* As String, **ByVal** *Format* as Integer) As Integer

C Declaration:

int FAR PASCAL QRZGetNext(LPSTR *Found*, int *Format*);

QRZInit

int QRZInit (***Drive***)

LPSTR ***Drive*** /* Address of string pointing to CDROM drive */

The **QRZInit** function is optionally called to initialize the callbook index and to prevent the DLL from searching for the desired CDROM drive. If **QRZInit** is not called, QRZDLL will start searching from drive C: upward until it finds a x:\callbk\callbkc.dat file. Using QRZInit merely shortens the startup time, and directs the program to a particular drive if multiple drives are available.

QRZinit returns either -1, 0 or a drive letter on success.

Return Values

> 0	Drive letter where database was found
0	No database was found
-1	QRZDLL Already in use

Visual Basic Declaration:

Declare Function QRZInit Lib "qrzdll.dll" (**ByVal** *Drive* As String) As Integer

C Declaration:

int FAR PASCAL QRZInit(LPSTR drive);

See also:QRZExit

QRZAdvance

int QRZAdvance (*Amount*, *Found, Format*)

int ***Amount*** /* Number of Records to Advance (seek forward) */
LPSTR ***Found*** /* Address of return buffer for result */
int ***Format*** /* Return record format type (see QRZSearch) */

The **QRZAdvance** function moves the current database record pointer forward in an unqualified manner. It is chiefly used to randomly browse through the database as might be done with a scroll bar or arrow button. The ***Amount*** parameter can be either 1 or 2, in which case the pointer will advance one record or 100 records, respectively. To move the pointer to a more specific location, use the QRZAdvanceToTo function. The inverse function, QRZBack, moves the pointer back either 1 or 100 records.

QRZAdvance returns the number of bytes written to the buffer *Found*.

Note that the actual pointer movement will be an inexact number of records when moving at distances of greater than 1 record. The actual movement is an approximation based on the size of an average record (currently 84 bytes per record). Thus, a jump of 100 equals 8,400 bytes forward plus the distance to the start of the next record.

The *Found* and *Format* parameters are identical to those used by QRZSearch.

Visual Basic Declaration:

Declare Function QRZAdvance Lib "qrzdll.dll" (**ByVal** *Amount* as Integer, **ByVal** *Found* As String, **ByVal** *Format* as Integer) As Integer

C Declaration:

int FAR PASCAL QRZAdvance(int *Amount*, LPSTR found, int *Format*);

QRZAdvanceTo

int QRZAdvanceTo (*Position*, *Found, Format*)

int ***Position*** /* Position in Selected Datafile (1/1000) */
LPSTR ***Found*** /* Address of return buffer for result */
int ***Format*** /* Return record format type (see QRZSearch) */

The **QRZAdvanceTo** function was created specifically support fast random tabbing through the database at the rate of 1/1000th of the database per jump. The *Position* argument specifies the absolute offset into the selected database file in the range of 0 to 1000. The same caviats regarding random pointer positioning as mentioned in QRZAdvance apply. **QRZAdvanceTo** returns the number of bytes written to the buffer *Found*.

The *Found* and *Format* parameters are identical to those used by QRZSearch.

Visual Basic Declaration:

Declare Function QRZAdvanceTo Lib "qrzdll.dll" (**ByVal** *Position* as Integer, **ByVal** *Found* As String,
ByVal *Format* as Integer) As Integer

C Declaration:

int FAR PASCAL QRZAdvanceTo (int Amount, LPSTR found, int Format);

QRZBack

int QRZBack (*Amount*, *Found, Format*)

int ***Amount*** /* Number of Records to Back up (seek backwards) */
LPSTR ***Found*** /* Address of return buffer for result */
int ***Format*** /* Return record format type (see QRZSearch) */

The **QRZBack** function is the logical inverse of the QRZAdvance function. QRZBack moves the current database record pointer backward by either 1 or 100 records. The exact position may vary due to the approximation used for record size (84 bytes). For example, to move back by 1 record, the routine will seek the current pointer back by 300 (84 + 84 + 42) bytes, or two and a half logical records. It will then seek forward twice, first to the end of the half record and then to read in the new current record.

The *Found* and *Format* parameters are identical to those used by QRZSearch.

Visual Basic Declaration:

Declare Function QRZBack Lib "qrzdll.dll" (**ByVal** *Amount* as Integer, **ByVal** *Found* As String, **ByVal** *Format* as Integer) As Integer

C Declaration:

int FAR PASCAL QRZBack(int *Amount*, LPSTR found, int *Format*);

QRZGetCount

long QRZGetCount ()

The **QRZGetCount** function returns the number of records which matched the most recent Search and/or GetNext / Count activity. QRZGetCount simply returns an internal variable, no record positioning or file I/O takes place. The result is returned as a long integer.

Visual Basic Declaration:

Declare Function QRZGetCount Lib "qrzdll.dll" () As Single

C Declaration:

long FAR PASCAL QRZGetCount();

QRZGetSbPos

int QRZGetSbPos ()

The **QRZGetSbPos** function was implemented as an adjunct to the QRZAdvanceTo function in that it returns an integer in the range of 0 to 1000 indicating the relative current position of the record pointer in the current data file. This value can be used to update a 0-1000 scrollbar or gauge after a search has been performed.

Visual Basic Declaration:

Declare Function QRZGetSbPos Lib "qrzdll.dll" () As Integer

C Declaration:

int FAR PASCAL QRZGetSbPos();

QRZCount

long QRZCount (*More*)

int * ***More*** /* Flag indicating whether the count is complete */

The **QRZCount** function returns the number of records which the most recent Key sequence and **Mode** used in QRZSearch . would return if QRZGetNext were used. Internally, **QRZCount** repetitively calls QRZGetNext but saves time by not formatting the records. The database position pointer is modified as a result of this call and is left pointing at the start of the second record which failed to match the Key sequence. The result is returned as a long integer. QRZBack can be used to back the pointer up to the last record which matched.

QRZCount must be called repetitively until the ***More*** flag returns false. The scanning mechanism returns after each 100 records to give the user interface program a chance to abort the operation in the event that it becomes excessively long. The value returned by **QRZCount** grows larger with each call and only the value returned when ***More*** is false is correct.

Visual Basic Declaration:

Declare Function QRZCount Lib "qrzdll.dll" (More as Integer) As Long

C Declaration:

long FAR PASCAL QRZCount(int *More);

QRZReformat

int QRZReformat (*Format, Found)*

int ***Format*** /* Return record format type (see QRZSearch) */
LPSTR ***Found*** /* Address of return buffer for result */

The **QRZReformat** function reformats the current record stored in memory to the indicated *Format* and returns it in the buffer pointed to by *Found*. No record pointer or file I/O takes place. The current record count remains unchanged. QRZReformat returns the number of bytes written to the buffer *Found*.

Visual Basic Declaration:

Declare Function QRZReformat Lib "qrzdll.dll" (ByVal Format as Integer, ByVal Found As String) as Integer

C Declaration:

int FAR PASCAL QRZReformat(int *Format,* LPSTR *Found*);

QRZField

void QRZField (*Field, Found, ReturnLen)*

int ***Field*** /* Return record format type (see QRZSearch) */
LPSTR ***Found*** /* Address of return buffer for result */
int ****ReturnLen*** /* Address of Return Length variable */

The **QRZField** function fetches an individual record field from the current record in memory. The value ***Field*** is set to indicate which field is desired. The result is returned in the buffer pointed to by ***Found***. No record pointer or file I/O takes place. The current record count remains unchanged. The variable ***ReturnLen***, passed as a pointer, is set to indicate the length of the field copied to Found.

Field Values:	**Description**:
CALLS	Callsign
LNAME	Last Name
JR	Jr / Sr / II / etc.
Fname	First Name
MI	Middle Initial
DOB	Date of Birth as mm/dd/yy
EFDATE	License Effective Date as mm/dd/yy
EXPDATE	License Expiration Date as mm/dd/yy
MAIL_STR	Street Address
MAIL_CITY	City
MAIL_ST	State
MAIL_ZIP	zip code
CLASS	License Class
P_CALL	Previous Callsign
P_CLASS	Previous Class
NUM_FIELDS	Number of Fields in record
FULLNAME	Full Name as JOHN P. SMITH JR
FULLCITY	Full City as PHOENIX, AZ 85008

Visual Basic Declarations:

Declare Sub QRZField Lib "qrzdll.dll" (**ByVal** *Field* as Integer, **ByVal** *Found* As String, *ReturnLen* as Integer)

Visual Basic Constants:

```
Const CALLS = 0
Const LNAME = 1
Const JR = 2
Const Fname = 3
Const MI = 4
Const DOB = 5
Const EFDATE = 6
Const EXPDATE = 7
Const MAIL_STR = 8
Const MAIL_CITY = 9
Const MAIL_ST = 10
Const MAIL_ZIP = 11
```

```
Const CLASS = 12
Const P_CALL = 13
Const P_CLASS = 14
Const NUM_FIELDS = 15
Const FULLNAME = 100
Const FULLCITY = 101
```

C Declaration:

void FAR PASCAL QRZField(int *Field,* LPSTR *Found,* int **ReturnLen*);

QRZSetFilter

void QRZSetFilter (*Filter*)

LPSTR ***Filter*** /* String containing single character filters */

The **QRZSetFilter** function is provided to tell QRZDLL which initial prefix characters to **exclude** from the callsign match routines. The filter only affects the **QRZCALL** search **Mode**. For example, passing the string "VG" would exclude all callsigns beginning with the letters 'V' or 'G', and in our case, all Canadian and UK callsigns. This filtering will be expanded to include multicharacter prefixes in a future edition of the library.

Visual Basic Declaration:

Declare Sub QRZSetFilter Lib "qrzdll.dll" (ByVal Filtstr As String)

C Declaration:

void FAR PASCAL QRZSetFilter(LPSTR filt)

QRZGetCbInfo

int QRZGetCbInfo (*Found*)

LPSTR ***Found***

> The **QRZGetCbInfo** function returns a string with information about the currently selected database. This information includes the name of the file in use, the size in bytes and the revision ID.
>
> **QRZGetCbInfo** returns the number of bytes written to **Found**.

Visual Basic Declaration:

Declare Function QRZGetCbInfo Lib "qrzdll.dll" (Found As String) as Integer

C Declaration:

int FAR PASCAL QRZGetCbInfo(LPSTR Found)

QRZExit

int QRZExit ()

The QRZExit function serves to release QRZDLL by giving it an opportunity to release the open file handles which it has borrowed from the calling application. This will allow another application to attach to QRZDLL before your program exits.

QRZExit returns the number of file handles which were open, which can be up to 4 in the current implementation.

Once you QRZExit has been called, all QRZDLL internal variables are reset to zero and no other calls to QRZDLL may be made until another QRZInit call is made.

It is recommended that you always call QRZExit just before your program ends.

Visual Basic Declaration:

Declare Function QRZExit Lib "qrzdll.dll" () as Integer

C Declaration:

int FAR PASCAL QRZExit(void)