Understanding the DLCALL Function

R:BASE Technologies, Inc.

Understanding the DLCALL Function

by R:BASE Technologies, Inc.

Special thanks to:

Mike Byerley (Fort Wayne, Indiana), an Authorized R:BASE Developer, for his contribution to the introduction, implementation and testing of the DLCALL Function in R:BASE.

Table of Contents

Part I

1 Introducing the DLCALL Function

The DLCALL Function calls any Windows dynamic link library (DLL) and loads it into memory for use with R:BASE.

Syntax for External DLL:

(DLCALL('libraryname.ext','FunctionOrProcedureName',[arg],[arg],[.....]))

Syntax for Windows API:

(DLCALL('libraryname','FunctionOrProcedureName',[arg],[arg],[.....]))

External DLLs must have the file name listed with the extension, such as 'MyLibrary.dll'.

Windows APIs require only the name of the Library, without the extension, such as: 'Kernel32' or 'User32'.

The DLLs must be created as Standard Windows 32-bit DLLs. Any number of functions or procedures can be used in a single DLL. Functions or Procedures to be used with DLCALL must be Exported in the DLL. No special code is necessary in the DLL for it to be used by R:BASE.

Part II

2 DLL Location

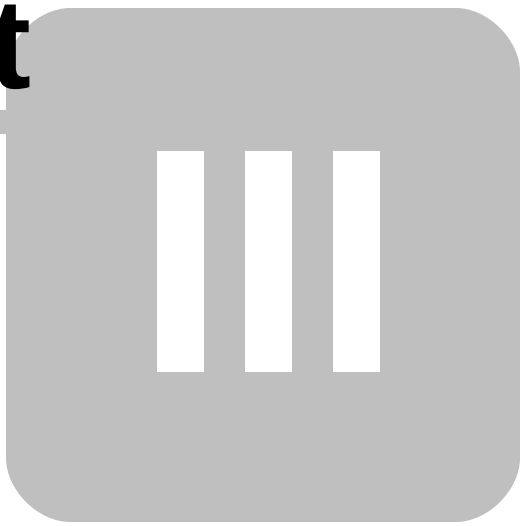
DLLs can be located in the Legal Windows Search Path, and If elsewhere, then specify the Full Path Name in DLLoad.

Search Path Used by Windows to Locate a DLL

With both implicit and explicit linking, Windows first searches for "known DLLs", such as Kernel32.dll and User32.dll. Windows then searches for the DLLs in the following sequence:

1. The directory where the executable module for the current process is located.
2. The current directory.
3. The Windows system directory. The GetSystemDirectory function retrieves the path of this directory.
4. The Windows directory. The GetWindowsDirectory function retrieves the path of this directory.
5. The directories listed in the PATH environment variable.

Part



3 When or If DLLOAD is Used

DLLOAD may be called "anytime" during the Session to Load the Library into Memory OR as a subsequent call to determine if the Library is Loaded.

In any event, DLLOAD is called internally when the Library is first referenced by DLCALL, and THEN the Library remains in memory until either the R:BASE session is ended OR DLFREE is called.

Part IV

4 Data Type Rules

Data Types in the functions must be of the Same Storage Size as the corresponding R:BASE Data Types.

Example: 32-bit Win32 Integer = 4 bytes of storage vs 32-bit R:BASE Integer = 4 bytes of storage

Part V

5 Declaration Logic

Calls to a function OR procedure from ANY DLL must be DECLARED at Least ONCE in the session in which they will be referenced.

Two Calling Conventions are supported, using the STDCALL and CDECL Keyword in the Declaration.

STDCALL

Function Declaration using STDCALL:

```
STDCALL FUNCTION 'functionName' (PTR VARCHAR (SIZE), ..... ) : VARCHAR (SIZE)
STDCALL FUNCTION 'functionName' ALIAS 'functionAliasName' (PTR TEXT
(SIZE), ..... ) : TEXT (SIZE)
```

Procedure Declaration using STDCALL:

```
STDCALL VOID FunctionOrProcedureThatHasNoReturnValue (PTR TEXT (SIZE))
```

Windows API Declaration using STDCALL:

```
STDCALL FUNCTION 'GetCurrentDirectoryA' ALIAS 'GetCurrentDir' (PTR TEXT, INTEGER )
: INTEGER
```

CDECL

Function Declaration using CDECL:

```
CDECL FUNCTION 'functionName' (INTEGER) : INTEGER
CDECL FUNCTION 'functionName' ALIAS 'functionAliasName' (PTR DOUBLE) : DOUBLE
```

Procedure Declaration using CDECL

```
CDECL VOID FunctionOrProcedureThatHasNoReturnValue (PTR TEXT (SIZE))
```

Important Notes:

- 'SIZE' applies to TEXT and VARCHAR Data Types.
- Parameters in the Declaration are in the REVERSE order from the Actual Function or Procedure in the DLL.
- Parameters can be 0 to *n* Parameters of any legal R:BASE Data type.
- Function Names ARE CASE SENSITIVE in the Declaration ONLY.
- The Case must match the casing used in the DLL.
- Case is INSENSITIVE when used in DLCALL.

Part VI

6 Remarks

- For best results, TEXT and VARCHAR data should be passed with the PTR (Pointer) Attribute and ANY VARCHAR data type Larger than 32K as a Parameter, MUST be passed as PTR.
- VARCHAR data type as a Return Value restricted to 32K, but Any SIZE up to 256MB can be passed as a Pointer to the R:BASE Variable. Modification of the Data Passed as Pointer must be on the data pointed to.
- It is important that the SIZE parameter on TEXT and VARCHAR be Specified to avoid creating excess buffer space that has to be created from the Declaration.

Part VII

7 Examples

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( ptr varchar (nn)) : integer
```
Then *nn* <= 256Mb.

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( ptr varchar ) : integer
```
Then because SIZE is omitted, the buffer for VARCHAR will have default SIZE = 256MB.
*** AVOID THIS UNLESS THAT IS THE ACTUAL SIZE OF THE DATA TO BE PASSED!**

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( integer ) : varchar(nn)
```
Then because SIZE is Specified, nn bytes <= 32K will be returned.

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( integer ) : varchar
```
Then because SIZE is omitted, the buffer for VARCHAR will have default SIZE = 32K.

When SIZE is Specified, the data passed as parameter or as return value, if Greater than the SIZE, will be truncated to SIZE.

7.1 Delphi

Example of a DLL created in Delphi exporting three functions:

```
// Begin Dll Code
library DemoLib;

uses
  SysUtils, Classes;

{$R *.res}

function MultInt (NumIN : Integer) : Integer; stdcall;
begin
  Result := (NumIN * 2);
end;

function MultDbl (NumDbl : Double) : Double; stdcall;
begin
  Result := (NumDbl * 2);
end;

procedure LCaseByREF(DataIN : PChar); stdcall;
begin
 ansiStrLower(DataIN);
end;

function LCaseByVAL (DataIN : PChar) : PChar; Stdcall;
begin
```

```
  Result := ansiStrLower(DataIN);
end;

Exports MultInt, LCaseByREF, LCaseByVAL;

begin

end.
// End Dll Code
```

7.2 R:BASE

Example Usage From Within R:BASE:

```
-- BEGIN Demo.rmd
-- Declare the functions to be used from the DLL
STDCALL function 'MultInt' ( Integer ) : Integer
STDCALL VOID 'LCaseByREF' (ptr TEXT (30))
STDCALL function 'LCaseByVAL (ptr TEXT (60)) : TEXT (60)

--Set somme variables for use
Set VAR vTEXT TEXT = 'RBASE TECHNOLOGIES'
SET VAR vINT INTEGER = 128
SET VAR v1 INTEGER = 0

-- OPTIONALLY CALL DLLOAD
SET VAR v1 = (DLLOAD('DemoLib.dll'))
IF v1 = 0 THEN
  PAUSE 2 USING 'DemoLib.dll NOT LOADED.. EXITING'
  RETURN
ENDIF

SET VAR V1 = (dlcall('demolib.dll', 'changecase', vtext))

{ The Value for v1 will be null because ChangeCase is a procedure and
doesn't
  RETURN A RESULT, but the value of vTEXT which is passed as a POINTER
has been
  changed to 'rbase technologies'}

SET VAR v1 = (DLCALL('demolib.dll', 'MultInt', vINT))

--The value for v1 will be 256 the value returned from the function.

-- running the following against RRBYW14
SELECT (DLCALL('demolib.dll','lcasebyval', Company))=60 FROM +
Customer WHERE LIMIT = 2

{Yields the following output:
 (DLCALL('demolib.dll','lcasebyval', Company)
  ------------------------------------------------------------
 computer warehouse - ii
 microtech university - i
```

```
}

SELECT ((ICAP2((DLCALL('demolib.dll','lcasebyval', Company))))) = 60 +
FROM Customer WHERE LIMIT = 2

{Yields the following otput:
 ((ICAP (DLCALL('demolib.dll','lcasebyval',
  ------------------------------------------------------------
 Computer Warehouse - Ii
 Microtech University - I
}

-- Optionally CALL DLFREE
SET VAR v1 = (DLFREE('DemoLib.dll'))

-- END Demo.rmd
```

7.3 C++

Example of a DLL created in C++ Exporting three functions:

```
// BEGIN C++ DLL
// loaddll.cpp : Defines the entry point for the DLL application.
//
#include <windows.h>
#include <stdio.h>

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                        )
{
    return TRUE;
}
#ifdef __cplusplus    // If used by C++ code,
extern "C" {          // we need to export the C interface
#endif

__declspec(dllexport) int cfunc1(int i){
        return i;
}
__declspec(dllexport) double cfunc2(double *inp){
        double rtn = *inp;
        rtn++;
        return rtn;
}
__declspec(dllexport) char * cfunc3(char *inp){
        strcat(inp," + ");
        return inp;
}

#ifdef __cplusplus
}
#endif
```

```
// END C++ DLL
```

Part VIII

8 Associated R:BASE Functions

8.1 CHKFUNC

(CHKFUNC('function_name'))

Checks to see if a DLL function exists or not.

If the DLL function exists, a 1 is returned. If the DLL function does not exist, a 0 is returned.

Example:

```
SET VAR v1 = (CHKFUNC('FunctionName'))
```

8.2 DELFUNC

(DELFUNC('function_name'))

Deletes a declared DLL function.

If the DLL function is successfully deleted, a 1 is returned. If the DLL function is not deleted, a 0 is returned.

Example:

```
SET VAR v1 = (DELFUNC('FunctionName'))
```

8.3 DLFREE

(DLFREE('*libraryname.ext*'))

Checks to see if a given library file can be freed.

The function returns an integer value of 1 if the library is freed and 0 if a given library is not freed.

Example:

```
SET VARIABLE vFreePlugin = (DLFREE('RAudioPlayer.RBL'))
```

8.4 DLLOAD

(DLLOAD('*libraryname.ext*'))

Checks to see if a given library file is loaded.

The function returns an integer value of 1 if the library is loaded and 0 if a given library is not loaded.

Example:

```
SET VARIABLE vLoadPlugin = (DLLOAD('RAudioPlayer.RBL'))
```

Index

Back Cover