RMAN Backup Reports - Source Code

Norman Dunbar

Version 1.0, 2018-04-07

Table of Contents

RMAN Backup Reports - Source Code

Introduction

See the file `UserGuide.pdf` if you want to use the `rmanBackups` utility. This file explains how to change and recompile it.

`RMANBackups.exe` is a utility that will login to a list of databases as the `DBA_USER` account, and create a report detailing all the RMAN jobs that were executed in the past 'n' days, where 'n' is a supplied parameter on the command line. It is written in plain, vanilla C code, with the Oracle access being facilitated by the *ocilib* library.

The program was compiled with the Windows version of the GNU C Compiler, version 5.1

Obtaining the Source Code

You may copy the source code from the DBA software Repository (`P:\Parcelnet System Development\dba\software_repository\RMANBackupReports\Latest`)to a safe place on your own PC, if necessary.

Only the `rmanBackupReports_SourceCode.zip` file is required. It contains the following:

- `Build.cmd` - a command file to build the utility on Windows. You may need to change some of the settings to match your own environment. It will build 32 bit executables if you tag '32' on the end of the command to call it, `build 32` for example. The default is 64 bit builds to match the installed Oracle client. It expects the main source file to be named `rmanBackups.c`.
- `Build.sh` - a command file to build the utility on Linux. You may need to change some of the settings to match your own environment.
- `rmanBackups.c` - the main source file.
- `rmanBackups.h` - the main include file.
- `css.h` - the include file that defines the CSS settings used by the output reports.
- `sql.h` - the include file that defines the SQL statement used to interrogate the database.

This file contains a couple of hard coded HTML `<br>` attributes, plus a couple of place holders, `%s`, for the number of days parameter that is supplied on the command line at runtime.

- `version.h` - the version number of the utility. Please update this whenever you make changes.
- `html.h` - an include file that defines a number of HTML strings and keeps the mess (HTML is *messy*!) out of the main file.
- `SourceCode.rst` - the ReStructuredText source file for this manual.
- `SourceCode.docx` - this manual.
- `UserGuide.rst` - the ReStructuredText source file for the user guide.
- `UserGuide.docx` - the user guide.
- `Example.png` - a screen dump of an example report, used in the User Guide.
- One or more HTML example files. Perhaps!

Compiler Environment

Compiler

The C compiler used to compile the utility is the 64 bit version of the MINGW GCC compiler - TDM-GCC-64. It can compile 32 and 64 bit applications, but the bittiness of the output must match the bittiness of your Oracle client installation. You cannot mix and match 32 and 64 bit binary files.

Oracle Access Library

The Oracle access library is the well renowned *ocilib* system which is used by numerous banks and other large companies, etc. And me!

The system provides a 32 and 64 bit static library for use when compiling any application that uses Oracle and *ocilib*.

You need to tell the compiler where to find the appropriate version of the `libocidlla.lib` file. This file will be statically linked to the application, however, at run time, it will need to find the appropriate version of the Oracle supplied `oci.dll` on the `PATH` somewhere (Windows) or on `LD_LIBRARY_PATH` (Linux). The latter file, `oci.dll`, is not needed at *compile* time.

Source Code

All source code, example reports and build scripts live in the same folder.

Compiling

A utility script, `build.cmd`, has been supplied that will compile the application on Windows, with all the correct settings required by the *ocilib* system, defined. `Build.sh` has been supplied for compilation on Linux, AWS for example.

These scripts *may* need changing to suit where you have installed the Oracle client software, the C compiler and the `ocilib` system. It is well commented so you should have no problem making the required changes.

The build scripts *must* live in the same folder as the `rmanBackups.c` file. It expects to find it there.

Compiling 64bit Applications

This is the default and is as simple as:

```
build
```

Obviously, your Oracle client must also be 64 bit or you will see some errors similar to the following:

```
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x1c8): undefined reference to
`OCI_Initialize'
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x277): undefined reference to
`OCI_ConnectionCreate'
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x2d6): undefined reference to
`OCI_StatementCreate'
...
```

Compiling 32bit Applications

This is equally as simple:

```
build 32
```

Obviously, your Oracle client must also be 32 bit or you will see some errors similar to the following:

```
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x1c8): undefined reference to
`OCI_Initialize'
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x277): undefined reference to
`OCI_ConnectionCreate'
C:\Users\...\Temp\ccO0X6PY.o:rmanBackups.c:(.text+0x2d6): undefined reference to
`OCI_StatementCreate'
...
```

Compiling by Hand

Should you wish to compile the code by hand, good luck! You will need a command line similar to the following, on Windows:

```
%gcc% %ccopts% -o rmanBackups.exe rmanBackups.c -L %libs% -I %includes%  -l ociliba
%compileOptions%
```

Or this, on Linux:

```
LD_LIBRARY_PATH=$LIBPATH $gcc -o $exeFile $srcFile -I $includes -L $libLocation -std=c99 -l
$libName $options
```

Read the comments at the top of build scripts for details of what each of the defined environment variables are, but they are, briefly, as follows:

- `%gcc%`, `$gcc` is the full path to the compiler executable.
- `%ccopts%` is the set of options required by the Windows C compiler. `-m32`, for example, to build 32 bit applications.
- `-o rmanBackups.exe`, `-o $exeFile` specifies the desired name of the application.
- `rmanBackups.c`, `$srcFile` is the name of the main source file.
- `-L %libs%`, `-L $libLocation` tells the linker where to find the 64 or 32 bit libraries for the `ocilib` system at link time.
- `-I %includes%`, `-I $includes` tells the compiler where to find the `ocilib` system's header files. (`-I` is an uppercase letter 'eye'.)
- `-l ociliba`, `-l $libName` tells the linker to use the file in `%libs%` named `libociliba.a` which is a static library and will be linked into the final executable. It is therefore not required at runtime to be on the path. (`-l` is a lowercase letter 'ell'.) If you change `%compileOptions%` (or `$options`) then you may have to change this library name too.
- `%compileOptions%`, `$options` is a list of options required by the compiler to know what to do when compiling `ocilib` applications. Do not change those from the ones defined in the `build.cmd` script, unless you know

exactly what you are doing!

The format of the command line *must* match that above because if the source file name comes after the libraries, then you will see a lot of errors from the linker.

The linker, is a tad dumb, and doesn't know what it needs until it reads in the newly compiled object file for the main source file. So, it needs to read the source's object file first to get a list of functions to link to, then resolve those functions by reading the various library files mentioned on the command line. *It really is a lot simpler to use the supplied build script!* Ask me how I know all this! :-)