Writing and Loading Dynamic C Functions for PostgreSQL Databases Running Under Windows

Overview

To add dynamic functions to a PostgreSQL database, they must be written in C and contain calls to certain postgreSQL-provided C code that makes them compatible with the protocols PostgreSQL uses to call functions; that code is in numerous PostgreSQL-provided "include" (C header) files and the library postgres.lib. The C source is compiled and then linked to form a "dynamic linked library" (DLL) that's placed into the directory where postgreSQL searches for such libraries. Then, via psql or pgAdmin or similar tool, the CREATE FUNCTION SQL command is used to add the functions into the postgreSQL database; authorized users can now employ them. Completing all these steps is not dificult—once you know what they are—but they must be done carefully to avoid errors.

References

- Dynamic C Language Functions - Section 34.9 of the postgreSQL manual
- CREATE FUNCTION page of the postgreSQL manual
- DROP FUNCTION page of the postgreSQL manual
- String Constants, Section 4.1.2.1 of the postgreSQL manual

Writing, Compiling and Linking

To illustrate writing, compiling and linking of C functions for inclusion in a postgreSQL database, we provide testFunctions.c, which contains two functions. The first, `add_one`, simply adds 1 to its parameter and returns the result, thus showing how numeric operations are handled. The second, `concat_text`, concatenates two strings and returns the result, to show how text is processed. Do review this program, in conjunction with the points below, to see how this program is fashioned:

- Some symbols in the files included in testFunctions.c need to be redefined; they were written for a Unix environment, and some changes are needed for them to work properly with C in a Windows environment. The changes are quite technical, so we provide the redefinitions for you at the start of the program—you're not expected to be an expert in the differences between Unix and Window DLLs. (Programmers rely on the expertise of other technical experts all the time, even when they don't notice it: Every time you use a source or object library or software written by others, you are using the results of their knowledge.) If you're interested in the details of why these changes are necessary, see the comments in the code and the documents they reference.

- The symbols that start with PG are aliases for C statements or define C macros needed for C functions to correctly interface with postgreSQL. For details on the purpose and behavior of the PG symbols not defined in the code itself, see Section 34.9 of the postgreSQL manual (reference above).

- Text variables in postgreSQL-compatible C functions are (usually) of type `text`, a type provided by postgres.lib that corresponds to the VARCHAR data type SQL employs. VARCHAR/text variables do not end in the "\0" that terminate standard C strings, so the standard C functions that manipulate strings, such as the string length function `strlen()`, do not work properly. In particular, using such functions can actually change the contents of the `text` data! (A common effect of using C string functions: When you call your string-handling procedure the first time, it produces a correct result; then, if you immediately call it again with the same parameters, it produces a different, incorrect result.)

- `VARCHAR` variables consist of header and data areas. `VARSIZE` is the size of the entire `VARCHAR` variable; `VARHDRSZ` is the size of its header; subtracting the two gives the size of the actual string data. `VARDATA()` returns the actual contents of the string stored in the `VARCHAR`.

Compile the file (in C), and link it to become a DLL:

- Go to `Start -> All Programs -> Microsoft Visual C++ 2008 Express Edition -> Visual Studio Tools -> Visual Studio 2008 Command Prompt`. This step calls up a command-line window whence you can invoke the C compiler and linker. Do not use the normal Windows command prompt: the Visual Studio command prompt "knows where to find" a number of C headers and libraries needed to compile and link C programs.

- Compile the code. if you have PostgreSQL installed in the default directory, `\Program Files\postgreSQL\8.3`, the command line is

  ```
  cl /I "\Program Files\postgreSQL\8.3\include\server" /I "\Program Files\postgreSQL\8.3\include\server\port\win32" /c <path to to wher
  ```

  `cl` is the C compiler (it can also link, but we are not using that capability; for our task, it's easier to call the linker directly). `/I` indicates directories to search for include files, `/c` indicates the file to compile. The quotes are needed around any path name that includes spaces so that the spaces are not taken as switch separators.

- Link the resulting object file to make a DLL: if you have postgreSQL installed in the default directory, the command line is

  ```
  link /DLL testFunctions.obj "\Program Files\postgreSQL\8.3\lib\postgres.lib"
  ```

  `link` is the linker. `/DLL` says to make a DLL file. postgres.lib needs to be linked with testFunctions.obj so that code needed for the DLL to communicate with postgreSQL is present.

Adding Functions to the Database

-

Place testFunctions.dll into a folder *that does not have any spaces in its path name*: If the path has a space, when you (try to) add the function to the database, you will get the error message "%1 is not a valid Win32 application"! Also remember that when you use a blackslash character in postgress strings, it must be escaped: wherever you want "\" to be part of the string, enter "\\".

-

Link the DLL into the database. To do so, log into the database via psql or pgAdmin as the postgres user of the database. (Since postgreSQL considers C functions "unsafe," only superusers can add C functions to a database.)

Add the functions to the database via the CREATE FUNCTION command:

```
CREATE FUNCTION add_one(INTEGER) RETURNS INTEGER AS E'<path to DLL>\\testFunctions', 'add_one' LANGUAGE C STRICT ;
CREATE FUNCTION concat(text, text) RETURNS text AS E'<path to DLL>\\testFunctions', 'concat_text' LANGUAGE C STRICT ;
```

(The E at the front of the string ensures that the \\ escaping mechanism is interpreted properly; see the String Constants reference, above, for details.)

You can check that the functions were added, and are working properly, by entering psql (or pgAdmin) as a user (e.g., testuser) and executing the functions:

```
fabflixs# SELECT add_one(1) ;
fabflixs# SELECT concat('Hello ', 'world') ;
```

If you need to remove a function from the database, you use the DROP FUNCTION command; for instance:

```
fabflixs# DROP FUNCTION add_one(integer) ;
```

If you want to replace a function with a new one with the same name and parameters, use CREATE OR REPLACE FUNCTION, as in

```
CREATE OR REPLACE FUNCTION add_one(INTEGER) RETURNS INTEGER AS E'<path to DLL>\\testFunctions', 'add_one' LANGUAGE C STRICT ;
```

Written by Chen Li, Winter 2005
Minor revisions made by Norman Jacobson for the Spring 2005 offering of ICS185, May 2005
Additional revisions, including additional comments about computing string lengths, made by Norman Jacobson, May 2005
Revised to include updates to reflect postgresql 8.1.4 made by Chen Li and Shengyue in their version of Phase 3 and other minor changes for clarity, by Norman Jacobson, April 2007
Minor revisions by Norman Jacobson, September 2007
A major revision of "A Primer on Writing Dynamic C Functions in PostgreSQL" to correct problems with provided postgres header files and to reflect how C functions are written, compiled and linked under postgreSQL 8.3.3 running under Windows, by Norman Jacobon, September 2008
Fix of minor typo in link command line, by Norman Jacobson, November 2008
Major revisions to compile line and CREATE FUNCTION lines to reflect posrgreSQL 8.3.7; additions to discuss replacing and dropping functions; adding of references; correction of some minor typos, by Norman Jacobson, March, April & May 2009