Addinga

Sometimes for performance reasons, you may want to add a function into the Postgres built-in function list rather than dynamically loading it. Also, because of limitations in the Postgres dynamic loader, this may be the only way to add a new function to Postgres. To do this, you will need a fair amount of familiarity with the C language, as well as the permission to delete the old Postgres binaries. This procedure is rather long and involved, and will require a re-installation of Postgres as described in Step 3, so it should be avoided unless necessary.

Generally, only the DBA should be handling these procedures.

- o Get this function added to the Postgres internal list of functions.

- o Get this function added to the Postgres table of user functions. How this is different from (1) will be discussed below.

- o Get this function linked into the Postgres binary.

You will need the contents of the following directories in order to do this. If you are running on a "sparse" system, this will not work. The directories are as follows:

```
~/src
~/newconf
```

First, you will need to write your function as discussed in the accompanying tutorial "User Defined Functions, Operators, and Types". The interfaces are identical whether the function is dynamically loaded or built in to Postgres.

Before proceeding, you will need to determine the type and returntype of your function. Suppose we are adding the functions line_in() and line_out(), and the declarations of these functions looked like:

```
LINE line_in(string)

char *string;

char *line_out(line)

LINE line;
```

1. STEP

To accomplish step 1, we do the following:

1.1. Add

Do this by editing the file

```
~/src/lib/H/builtins.h
```

and add the line

```
extern LINE line_in();
extern char *line_out();
```

1.2. Tell

Since <type> is a user defined type, we need to add the type declaration to the file

```
~/src/lib/H/tmp/c.h
```

as follows

```
typedef struct {
        int x0, y0, x1, y1;
} LINE;
```

(Do not do it in builtins.h)

If this function is to return an array of already existing Postgres types, you do not need to do this. But be sure to make sure the function formats the array correctly, as discussed in the accompanying tutorial, "Defining Array Types".

1.3. Add

Now we need to edit the file

```
~/src/lib/H/fmgr.h
```

and add the symbols

```
#define F_LINEIN  <number>
#define F_LINEOUT <number + 1>
```

NOTE: <number> MUST be UNIQUE, and must follow SEQUENTIALLY from the last number in this file. This number is used as the function key (OID) in searches of the function table. In fmgr.h the bottom of this file could look like

```
#define F_GETATTRIBUTE 654
#define F_LINEIN       655
#define F_LINEOUT      656

#endif  /* !defined(FMgrIncluded) */
```

when we are done. We will need to remember the number assigned to these functions for step 2.

1.4. Add

Now edit the file

```
~/src/utils/fmgr/fmgr.c
```

and we look at the declaration of FmgrCall. Now we go to the END of the builtins[] array and add the line

```
        {F_LINEIN, 1, (FmgrFunction) linein},
        {F_LINEOUT, 1, (FmgrFunction) lineout}
```

The number 1 in the second argument above is the number of arguments to this function. Note that in the third argument, there are no parentheses following the function name. After we are done with this step, the end of the builtins array in fmgr.c should look like

```
        {F_GETATTRIBUTE, 1, (FmgrFunction) GetAttribute },
        {F_LINEIN, 1, (FmgrFunction) linein},
        {F_LINEOUT, 1, (FmgrFunction) lineout}

};
```

Now we are done with Step 1.

2. STEP

To accomplish step 2, we will need to do the following:

NOTE: If you are not adding a new type, skip this step and proceed to step 2.1.

3. Add

Edit the file

```
~/src/lib/H/catalog/pg_type.h
```

and look at the declaration for pg_type. After we have looked at it, the following statements should be said about pg_type:

- o typname (the first field) must be unique and be no more than 15 characters.

- o typlen (the third field) must be equal to sizeof(type). In the case of the LINE type defined above, typlen is 16.

- o typprtlen (the fourth field) is how long the string is to be when this type is printed. It has to be at least as big as typlen, but can be larger. The length of this field depends on how the out function for the type (see "Defining User Defined Types, Operators, and Functions") is written.

- o If this is an array of existing types, typelem (the eighth field) must be equal to the OID of the array element.

- o typinput, typoutput, typreceive, typsend (the 10th through 13th elements) can generally be equal to the in/out functions of the type. See below example.

The other fields can be copied from the type "bool".

Knowing the above, we add the LINE type as follows:

```
DATA(insert OID = 705 ( LINE 6 16 40 t b t 0 0 linein lineout linein lineout -)
```

NOTE: If the type declaration is too long to fit in 80 characters, **DO**NOT insert a carriage return as the type will not be parsed correctly. Let the line wrap instead.

3.1. Adding

Now we edit the file

```
~/src/lib/H/catalog/pg_proc.h
```

and we look at the declaration of pg_proc. A few things should be said here about pg_proc.

- o proname (the first field) must be equal to the function name.

- o proiscacheable (the sixth field) is the same as the iscacheable flag described in the Reference Manual under "Define C Function". This flag will be set to false in these functions.

- o pronargs (the seventh field) is the number of function arguments.

- o prorettype (the eighth field) is the return type of the function. In our case, for linein it will be 705 (because that is the OID for the LINE type from Step 2.1) and for lineout (as for any out function) it will be 23.

Remembering the function key (OID) from Step 1B, we add the lines

```
DATA(insert OID = 655 (  linein      6 11 f t f 1 705 foo bar ));
DATA(insert OID = 656 (  lineout      6 11 f t f 1  23 foo bar ));
```

We are now done with Step 2.

4. Recompiling

To accomplish Step 3, we will need to recompile the Postgres binaries and reconfigure the Postgres databases. To do this, you will need to use the "copy" command described in the Reference Manual to create a backup of your databases.

- o After making sure any Postgres users are not using Postgres, kill the Postmaster and vacuum daemon.

- Have your users make copies of all user databases using the "copy to" command into separate files, as described in the Reference Manual.

- Using

```
% rm -rf data
% rm -rf <obj-dir>
% rm -rf bin
% rm -rf files
% rm newconf/everything.stat
```

  delete the old Postgres installation, as it is now out of date.

- Run "Make install" from the newconf directory

- Have the users repopulate their databases using the "copy from" command described in the Reference Manual. If they are using any ADT's or user-defined functions, they will need to run the queries to get Postgres to know about these as well.

Once a function has been added to the builtin list, it can be used to define a type or operator in the same fashion as a function defined using the POSTQUEL "DEFINE C FUNCTION" command.