

Search Documentation: Text Size: [Normal](#) / [Large](#)[Home](#) → [Documentation](#) → [Manuals](#) → [PostgreSQL 9.1](#)[PostgreSQL 9.1.1 Documentation](#)[Prev](#)[Fast](#)[Backward](#)[Fast](#)
[Forward](#)[Next](#)

Chapter 43. Server Programming Interface

Table of Contents

43.1. [Interface Functions](#)

[SPI_connect](#) -- connect a procedure to the SPI manager

[SPI_finish](#) -- disconnect a procedure from the SPI manager

[SPI_push](#) -- push SPI stack to allow recursive SPI usage

[SPI_pop](#) -- pop SPI stack to return from recursive SPI usage

[SPI_execute](#) -- execute a command

[SPI_exec](#) -- execute a read/write command

[SPI_execute_with_args](#) -- execute a command with out-of-line parameters

[SPI_prepare](#) -- prepare a plan for a command, without executing it yet

[SPI_prepare_cursor](#) -- prepare a plan for a command, without executing it yet

[SPI_prepare_params](#) -- prepare a plan for a command, without executing it yet

[SPI_getargcount](#) -- return the number of arguments needed by a plan prepared by `SPI_prepare`

[SPI_getargtypeid](#) -- return the data type OID for an argument of a plan prepared by

`SPI_prepare`

[SPI_is_cursor_plan](#) -- return true if a plan prepared by `SPI_prepare` can be used with

`SPI_cursor_open`

[SPI_execute_plan](#) -- execute a plan prepared by `SPI_prepare`

[SPI_execute_plan_with_paramlist](#) -- execute a plan prepared by `SPI_prepare`

[SPI_execp](#) -- execute a plan in read/write mode

[SPI_cursor_open](#) -- set up a cursor using a plan created with `SPI_prepare`

[SPI_cursor_open_with_args](#) -- set up a cursor using a query and parameters

[SPI_cursor_open_with_paramlist](#) -- set up a cursor using parameters

[SPI_cursor_find](#) -- find an existing cursor by name

[SPI_cursor_fetch](#) -- fetch some rows from a cursor

[SPI_cursor_move](#) -- move a cursor

[SPI_scroll_cursor_fetch](#) -- fetch some rows from a cursor

[SPI_scroll_cursor_move](#) -- move a cursor

[SPI_cursor_close](#) -- close a cursor

[SPI_saveplan](#) -- save a plan

43.2. [Interface Support Functions](#)

[SPI_fname](#) -- determine the column name for the specified column number

[SPI_fnumber](#) -- determine the column number for the specified column name

[SPI_getvalue](#) -- return the string value of the specified column

[SPI_getbinval](#) -- return the binary value of the specified column

[SPI_gettype](#) -- return the data type name of the specified column
[SPI_gettypeid](#) -- return the data type OID of the specified column
[SPI_getrelname](#) -- return the name of the specified relation
[SPI_getnsname](#) -- return the namespace of the specified relation

43.3. [Memory Management](#)

[SPI_palloc](#) -- allocate memory in the upper executor context
[SPI_realloc](#) -- reallocate memory in the upper executor context
[SPI_pfree](#) -- free memory in the upper executor context
[SPI_copytuple](#) -- make a copy of a row in the upper executor context
[SPI_returntuple](#) -- prepare to return a tuple as a Datum
[SPI_modifytuple](#) -- create a row by replacing selected fields of a given row
[SPI_freetuple](#) -- free a row allocated in the upper executor context
[SPI_freetuptable](#) -- free a row set created by `SPI_execute` or a similar function
[SPI_freeplan](#) -- free a previously saved plan

43.4. [Visibility of Data Changes](#)

43.5. [Examples](#)

The *Server Programming Interface* (SPI) gives writers of user-defined C functions the ability to run SQL commands inside their functions. SPI is a set of interface functions to simplify access to the parser, planner, and executor. SPI also does some memory management.

Note: The available procedural languages provide various means to execute SQL commands from procedures. Most of these facilities are based on SPI, so this documentation might be of use for users of those languages as well.

To avoid misunderstanding we'll use the term "function" when we speak of SPI interface functions and "procedure" for a user-defined C-function that is using SPI.

Note that if a command invoked via SPI fails, then control will not be returned to your procedure. Rather, the transaction or subtransaction in which your procedure executes will be rolled back. (This might seem surprising given that the SPI functions mostly have documented error-return conventions. Those conventions only apply for errors detected within the SPI functions themselves, however.) It is possible to recover control after an error by establishing your own subtransaction surrounding SPI calls that might fail. This is not currently documented because the mechanisms required are still in flux.

SPI functions return a nonnegative result on success (either via a returned integer value or in the global variable `SPI_result`, as described below). On error, a negative result or `NULL` will be returned.

Source code files that use SPI must include the header file `executor/spi.h`.

[Prev](#)

[Home](#)

[Next](#)

Environment Variables

[Up](#)

Interface Functions

[Privacy Policy](#) | Project hosted by [our server sponsors](#). | Designed by [tinysofa](#)

Copyright © 1996 – 2011 PostgreSQL Global Development Group