NAME

rpcgen − an RPC protocol compiler

SYNOPSIS

rpcgen *infile*
rpcgen [−D*name****[=****value****]] [−T] [−K*** *secs] infile*
rpcgen −c|−h|−l|−m|−t [−o *outfile] infile*
rpcgen [−I] −s *nettype* ***[−o*** *outfile] infile*
rpcgen −n *netid* ***[−o*** *outfile] infile*

DESCRIPTION

rpcgen is a tool that generates C code to implement an RPC protocol. The input to ***rpcgen*** is a language similar to C known as RPC Language (Remote Procedure Call Language).

rpcgen is normally used as in the first synopsis where it takes an input file and generates up to four output files. If the *infile* is named ***proto.x***, then ***rpcgen*** will generate a header file in ***proto.h***, XDR routines in ***proto_xdr.c***, server-side stubs in ***proto_svc.c***, and client-side stubs in ***proto_clnt.c***. With the ***−T*** option, it will also generate the RPC dispatch table in ***proto_tbl.i***. With the ***−Sc*** option, it will also generate sample code which would illustrate how to use the remote procedures on the client side. This code would be created in ***proto_client.c***. With the ***−Ss*** option, it will also generate a sample server code which would illustrate how to write the remote procedures. This code would be created in ***proto_server.c***.

The server created can be started both by the port monitors (for example, ***inetd*** or ***listen***) or by itself. When it is started by a port monitor, it creates servers only for the transport for which the file descriptor ***0*** was passed. The name of the transport must be specified by setting up the environmental variable ***PM_TRANSPORT***. When the server generated by ***rpcgen*** is executed, it creates server handles for all the transports specified in ***NETPATH*** environment variable, or if it is unset, it creates server handles for all the visible transports from ***/etc/netconfig*** file. Note: the transports are chosen at run time and not at compile time. When the server is self-started, it backgrounds itself by default. A special define symbol ***RPC_SVC_FG*** can be used to run the server process in foreground.

The second synopsis provides special features which allow for the creation of more sophisticated RPC servers. These features include support for user provided ***#defines*** and RPC dispatch tables. The entries in the RPC dispatch table contain:

- pointers to the service routine corresponding to that procedure,
- a pointer to the input and output arguments
- the size of these routines

A server can use the dispatch table to check authorization and then to execute the service routine; a client library may use it to deal with the details of storage management and XDR data conversion.

The other three synopses shown above are used when one does not want to generate all the output files, but only a particular one. Some examples of their usage is described in the EXAMPLE section below. When ***rpcgen*** is executed with the ***−s*** option, it creates servers for that particular class of transports. When executed with the ***−n*** option, it creates a server for the transport specified by *netid*. If *infile* is not specified, ***rpcgen*** accepts the standard input.

The C preprocessor, ***cc −E*** [see ***cc***(1)], is run on the input file before it is actually interpreted by ***rpcgen***. For each type of output file, ***rpcgen*** defines a special preprocessor symbol for use by the ***rpcgen*** programmer:

RPC_HDR	defined when compiling into header files
RPC_XDR	defined when compiling into XDR routines
RPC_SVC	defined when compiling into server-side stubs
RPC_CLNT	defined when compiling into client-side stubs
RPC_TBL	defined when compiling into RPC dispatch tables

Any line beginning with '***%***' is passed directly into the output file, uninterpreted by ***rpcgen***.

For every data type referred to in *infile*, ***rpcgen*** assumes that there exists a routine with the string ***xdr_*** prepended to the name of the data type. If this routine does not exist in the RPC/XDR library, it must be

provided. Providing an undefined data type allows customization of XDR routines.

The following options are available:

−a Generate all the files including sample code for client and server side.

−b This generates code for the SunOS4.1 style of rpc. It is for backward compatibilty. This is the default.

−5 This generates code for the SysVr4 style of rpc. It is used by the Transport Independent RPC that is in Svr4 systems. By default rpcgen generates code for SunOS4.1 stype of rpc.

−c Compile into XDR routines.

−C Generate code in ANSI C. This option also generates code that could be compiled with the C++ compiler. This is the default.

−k Generate code in K&R C. The default is ANSI C.

−D*name[=value]*

Define a symbol *name*. Equivalent to the ***#define*** directive in the source. If no *value* is given, *value* is defined as ***1***. This option may be specified more than once.

−h Compile into ***C*** data-definitions (a header file). ***−T*** option can be used in conjunction to produce a header file which supports RPC dispatch tables.

−I Generate a service that can be started from inetd. The default is to generate a static service that handles transports selected with ***−s***. Using ***−I*** allows starting a service by either method.

-K *secs* By default, services created using ***rpcgen*** wait ***120*** seconds after servicing a request before exiting. That interval can be changed using the ***-K*** flag. To create a server that exits immediately upon servicing a request, ***-K 0*** can be used. To create a server that never exits, the appropriate argument is ***-K -1***.

When monitoring for a server, some portmonitors, like ***listen***(1M), *always* spawn a new process in response to a service request. If it is known that a server will be used with such a monitor, the server should exit immediately on completion. For such servers, ***rpcgen*** should be used with ***-K -1***.

−l Compile into client-side stubs.

−m Compile into server-side stubs, but do not generate a “main” routine. This option is useful for doing callback-routines and for users who need to write their own “main” routine to do initialization.

−n *netid*

Compile into server-side stubs for the transport specified by *netid*. There should be an entry for *netid* in the netconfig database. This option may be specified more than once, so as to compile a server that serves multiple transports.

−N Use the newstyle of rpcgen. This allows procedures to have multiple arguments. It also uses the style of parameter passing that closely resembles C. So, when passing an argument to a remote procedure you do not have to pass a pointer to the argument but the argument itself. This behaviour is different from the oldstyle of rpcgen generated code. The newstyle is not the default case because of backward compatibility.

−o *outfile*

Specify the name of the output file. If none is specified, standard output is used (***−c***, ***−h***, ***−l***, ***−m***, ***−n***, ***−s***, ***−s −sand −t*** modes only).

−s *nettype*

Compile into server-side stubs for all the transports belonging to the class *nettype*. The supported classes are ***netpath***, ***visible***, ***circuit_n***, ***circuit_v***, ***datagram_n***, ***datagram_v***, ***tcp***, and ***udp*** [see ***rpc***(3N) for the meanings associated with these classes]. This option may be specified more than once. Note: the transports are chosen at run time and not at compile time.

−Sc Generate sample code to show the use of remote procedure and how to bind to the server before calling the client side stubs generated by rpcgen.

−Ss Generate skeleton code for the remote procedures on the server side. You would need to fill in the actual code for the remote procedures.

−t Compile into RPC dispatch table.

−T Generate the code to support RPC dispatch tables.

The options ***−c***, ***−h***, ***−l***, ***−m***, ***−s*** and ***−t*** are used exclusively to generate a particular type of file, while the options ***−D*** and ***−T*** are global and can be used with the other options.

NOTES

The RPC Language does not support nesting of structures. As a work-around, structures can be declared at the top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

The server code generated with ***−n*** option refers to the transport indicated by *netid* and hence is very site specific.

EXAMPLE

The following example:

$ rpcgen −T prot.x

generates the five files: ***prot.h***, ***prot_clnt.c***, ***prot_svc.c***, ***prot_xdr.c*** and ***prot_tbl.i***.

The following example sends the C data-definitions (header file) to the standard output.

$ rpcgen −h prot.x

To send the test version of the ***-DTEST***, server side stubs for all the transport belonging to the class ***datagram_n*** to standard output, use:

$ rpcgen −s datagram_n −DTEST prot.x

To create the server side stubs for the transport indicated by *netid* ***tcp***, use:

$ rpcgen −n tcp −o prot_svc.c prot.x

SEE ALSO

cc(1).