

```
new/usr/src/cmd/rpcgen/rpc_clntout.c
```

```
*****
10166 Sun Aug 3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_clntout.c
rpcgen should only produce ANSI code
*****
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
22 /*
23 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
24 *
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26 * Use is subject to license terms.
27 */
28 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29 /* All Rights Reserved */
30 /*
31 * University Copyright- Copyright (c) 1982, 1986, 1988
32 * The Regents of the University of California
33 * All Rights Reserved
34 *
35 * University Acknowledgment- Portions of this document are derived from
36 * software developed by the University of California, Berkeley, and its
37 * contributors.
38 */
40 /*
41 * rpc_clntout.c, Client-stub outputter for the RPC protocol compiler
42 */
43 #include <stdio.h>
44 #include <string.h>
45 #include <rpc/types.h>
46 #include "rpc_parse.h"
47 #include "rpc_util.h"
49 extern void pdeclaration(char *, declaration *, int, char *);
50 extern void printarglist(proc_list *, char *, char *, char *);
52 static void write_program(definition *);
53 static void printbody(proc_list *);
55 static char RESULT[] = "clnt_res";
57 #define DEFAULT_TIMEOUT 25      /* in seconds */
59 void
60 write_stubs(void)
61 {
```

```
1
```

```
new/usr/src/cmd/rpcgen/rpc_clntout.c
```

```
62     list *l;
63     definition *def;
64
65     f_print(fout,
66             "\n/* Default timeout can be changed using clnt_control() */\n");
67     f_print(fout, "static struct timeval TIMEOUT = { %d, 0 }; \n",
68             DEFAULT_TIMEOUT);
69     for (l = defined; l != NULL; l = l->next) {
70         def = (definition *) l->val;
71         if (def->def_kind == DEF_PROGRAM) {
72             write_program(def);
73         }
74     }
75 }
```

unchanged\_portion\_omitted

```
105 /*
106 * Writes out declarations of procedure's argument list.
107 * In either ANSI C style, in one of old rpcgen style (pass by reference),
108 * or new rpcgen style (multiple arguments, pass by value);
109 */
111 /* sample addargname = "clnt"; sample addargtype = "CLIENT * " */
113 void
114 printarglist(proc_list *proc, char *result, char *addargname, char *addargtype)
115 {
116     bool_t oneway = streq(proc->res_type, "oneway");
117     decl_list *l;
119     if (!newstyle) {
120         /* old style: always pass argument by reference */
121         if (Cflag) { /* C++ style heading */
122             f_print(fout, "(");
123             ptype(proc->args.decls->decl.prefix,
124                   proc->args.decls->decl.type, 1);
125             if (mtfflag) { /* Generate result field */
126                 f_print(fout, "*argp, ");
127                 if (!oneway) {
128                     ptype(proc->res_prefix, proc->res_type, 1);
129                     ptype(proc->res_prefix,
130                           proc->res_type, 1);
131                     f_print(fout, "*%s, ", result);
132                 }
133                 f_print(fout, "%s%s)\n", addargtype, addargname);
134                 f_print(fout, "%s%s)\n",
135                         addargtype, addargname);
136             } else {
137                 f_print(fout, "**argp, %s%s)\n", addargtype, addargname);
138                 f_print(fout, "**argp, %s%s)\n",
139                         addargtype, addargname);
140             }
141             if (!mtfflag)
142                 f_print(fout, "(argp, %s)\n", addargname);
143             else {
144                 f_print(fout, "(argp, %s)\n",
145                         addargtype, addargname);
146                 f_print(fout, "%s, ",
147                         result);
148             }
149         }
150         f_print(fout, "\t");
151         ptype(proc->args.decls->decl.prefix,
```

```
2
```

```

150             proc->args.decls->decl.type, 1);
151             f_print(fout, "*argp;\n");
152             if (mtflag && !oneway) {
153                 f_print(fout, "\t");
154                 ptype(proc->res_prefix, proc->res_type, 1);
155                 f_print(fout, "*%s;\n", result);
156             }
157         }
158     } else if (streq(proc->args.decls->decl.type, "void")) {
159         /* newstyle, 0 argument */
160         if (mtflag) {
161             f_print(fout, "(");
162
163             if (Cflag) {
164                 if (!oneway) {
165                     ptype(proc->res_prefix, proc->res_type, 1);
166                     ptype(proc->res_prefix,
167                           proc->res_type, 1);
168                     f_print(fout, "*%s, ", result);
169
170                     f_print(fout, "%s%s)\n", addargtype, addargname);
171                     f_print(fout, "%s%s)\n",
172                           addargtype, addargname);
173                 } else
174                     f_print(fout, "(%s)\n", addargname);
175
176             } else
177                 f_print(fout, "(%s%s)\n", addargtype, addargname);
178         } else
179             f_print(fout, "(%s)\n", addargname);
180     } else { /* new style, 1 or multiple arguments */
181         if (!Cflag) {
182             f_print(fout, "(");
183             for (l = proc->args.decls; l != NULL; l = l->next)
184                 f_print(fout, "%s, ", l->decl.name);
185             if (mtflag && !oneway)
186                 f_print(fout, "%s, ", result);
187
188             f_print(fout, "%s)\n", addargname);
189             for (l = proc->args.decls; l != NULL; l = l->next) {
190                 pdeclaration(proc->args.argname, &l->decl, 0, "", "");
191                 pdeclaration(proc->args.argname,
192                               &l->decl, 1, ";\\n");
193             }
194             if (mtflag && !oneway) {
195                 f_print(fout, "\t");
196                 ptype(proc->res_prefix, proc->res_type, 1);
197                 f_print(fout, "*%s;\n", result);
198             }
199         } else { /* C++ style header */
200             f_print(fout, "(");
201             for (l = proc->args.decls; l != NULL; l = l->next) {
202                 pdeclaration(proc->args.argname, &l->decl, 0,
203                               "", "");
204             }
205             if (mtflag && !oneway) {
206                 ptype(proc->res_prefix, proc->res_type, 1);
207                 f_print(fout, "*%s, ", result);
208             }
209         }
210     }
211     f_print(fout, "%s%s)\n", addargtype, addargname);
212 }
```

```

214         if (!Cflag)
215             f_print(fout, "\t%s%s;\n", addargtype, addargname);
216     }
217 }
```

unchanged\_portion\_omitted

```
new/usr/src/cmd/rpcgen/rpc_cout.c
```

```
*****
21350 Sun Aug 3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_cout.c
rpcgen should only produce ANSI code
*****
1 /*
2  * CDDL HEADER START
3 *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7 *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
23  *
24  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
25  * Use is subject to license terms.
26  */
27 /*
28  * Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29  * All Rights Reserved */
30 /*
31  * University Copyright- Copyright (c) 1982, 1986, 1988
32  * The Regents of the University of California
33  * All Rights Reserved
34  *
35  * University Acknowledgment- Portions of this document are derived from
36  * software developed by the University of California, Berkeley, and its
37  * contributors.
38 */
39 /*
40  * rpc_cout.c, XDR routine outputter for the RPC protocol compiler
41  */
42 #include <stdio.h>
43 #include <stdlib.h>
44 #include <string.h>
45 #include <cctype.h>
46 #include <ctype.h>
47 #include "rpc_parse.h"
48 #include "rpc_util.h"
49
50 extern void crash(void);
51
52 static void print_header(definition *);
53 static void print_trailer(void);
54 static void emit_enum(definition *);
55 static void emit_program(definition *);
56 static void emit_union(definition *);
57 static void emit_struct(definition *);
58 static void emit_typedef(definition *);
59 static void print_stat(int, declaration *);
60 static void emit_inline(int, declaration *, int);
61 static void emit_inline64(int, declaration *, int);
```

```
1
```

```
new/usr/src/cmd/rpcgen/rpc_cout.c
```

```
62 static void emit_single_in_line(int, declaration *, int, relation);
63 static void emit_single_in_line64(int, declaration *, int, relation);
64 static char *upcase(char *);

66 /*
67  * Emit the C-routine for the given definition
68  */
69 void
70 emit(definition *def)
71 {
72     if (def->def_kind == DEF_CONST)
73         return;
74     if (def->def_kind == DEF_PROGRAM) {
75         emit_program(def);
76         return;
77     }
78     if (def->def_kind == DEF_TYPEDEF) {
79         /*
80          * now we need to handle declarations like
81          * struct typedef foo foo;
82          * since we dont want this to be expanded into 2 calls
83          * to xdr_foo
84          */
85
86         if (strcmp(def->def_ty.old_type, def->def_name) == 0)
87             return;
88     }
89     print_header(def);
90     switch (def->def_kind) {
91     case DEF_UNION:
92         emit_union(def);
93         break;
94     case DEF_ENUM:
95         emit_enum(def);
96         break;
97     case DEF_STRUCT:
98         emit_struct(def);
99         break;
100    case DEF_TYPEDEF:
101        emit_typedef(def);
102        break;
103    }
104    print_trailer();
105 }
```

unchanged\_portion\_omitted

```
126 static void
127 print_generic_header(char *procname, int pointerp)
128 {
129     f_print(fout, "\n");
130     f_print(fout, "bool_t\n");
131     if (Cflag) {
132         f_print(fout, "xdr_%s(", procname);
133         f_print(fout, "XDR *xdrs, ");
134         f_print(fout, "%s ", procname);
135         if (pointerp)
136             f_print(fout, "*");
137         f_print(fout, "objp)\n{\n\n");
138     } else {
139         f_print(fout, "xdr_%s(xdrs, objp)\n", procname);
140         f_print(fout, "\tXDR *xdrs;\n");
141         f_print(fout, "\t%s ", procname);
142         if (pointerp)
143             f_print(fout, "*");
144         f_print(fout, "objp;\n{\n\n");
145     }
```

```
2
```

```
143     }
137 }
```

unchanged portion omitted

```
new/usr/src/cmd/rpcgen/rpc_hout.c
```

```
*****
11738 Sun Aug 3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_hout.c
rpcgen should only produce ANSI code
*****
```

1 /\*  
2 \* CDDL HEADER START  
3 \*  
4 \* The contents of this file are subject to the terms of the  
5 \* Common Development and Distribution License (the "License").  
6 \* You may not use this file except in compliance with the License.  
7 \*  
8 \* You can obtain a copy of the license at [usr/src/OPENSOLARIS.LICENSE](#)  
9 \* or <http://www.opensolaris.org/os/licensing>.  
10 \* See the License for the specific language governing permissions  
11 \* and limitations under the License.  
12 \*  
13 \* When distributing Covered Code, include this CDDL HEADER in each  
14 \* file and include the License file at [usr/src/OPENSOLARIS.LICENSE](#).  
15 \* If applicable, add the following below this CDDL HEADER, with the  
16 \* fields enclosed by brackets "[]" replaced with your own identifying  
17 \* information: Portions Copyright [yyyy] [name of copyright owner]  
18 \*  
19 \* CDDL HEADER END  
20 \*/

22 /\*  
23 \* Copyright 2014 Garrett D'Amore <[garrett@damore.org](mailto:garrett@damore.org)>  
24 \*  
25 \* Copyright 2009 Sun Microsystems, Inc. All rights reserved.  
26 \* Use is subject to license terms.  
27 \*/  
28 /\* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T \*/  
29 /\* All Rights Reserved \*/  
30 /\*  
31 \* University Copyright- Copyright (c) 1982, 1986, 1988  
32 \* The Regents of the University of California  
33 \* All Rights Reserved  
34 \*  
35 \* University Acknowledgment- Portions of this document are derived from  
36 \* software developed by the University of California, Berkeley, and its  
37 \* contributors.  
38 \*/

40 /\*  
41 \* rpc\_hout.c, Header file outputter for the RPC protocol compiler  
42 \*/  
43 #include <stdio.h>  
44 #include <stdlib.h>  
45 #include <ctype.h>  
46 #include "rpc\_parse.h"  
47 #include "rpc\_util.h"  
  
49 extern void pprocdef(proc\_list \*, version\_list \*, char \*, int);  
47 extern void pprocdef(proc\_list \*, version\_list \*, char \*, int, int);  
50 extern void pdeclaration(char \*, declaration \*, int, char \*);  
  
52 static void storexdrfuncdecl(char \*, int);  
53 static void pconstdef(definition \*);  
54 static void pstructdef(definition \*);  
55 static void puniondef(definition \*);  
56 static void pdefine(char \*, char \*);  
57 static void pprogramdef(definition \*);  
58 static void parlist(proc\_list \*, char \*);  
59 static void penumdef(definition \*);  
60 static void ptypedef(definition \*);

```
1
```

```
new/usr/src/cmd/rpcgen/rpc_hout.c
```

```
61 static uint_t undefined2(char *, char *);  
  
63 enum rpc_gvc {  
64     PROGRAM,  
65     VERSION,  
66     PROCEDURE  
67 };  
    unchanged_portion_omitted  
  
143 void  
144 print_xdr_func_def(char *name, int pointerp)  
142 print_xdr_func_def(char *name, int pointerp, int i)  
145 {  
144     if (i == 2)  
145         f_print(fout, "extern bool_t xdr_%s();\n", name);  
146     else  
146         f_print(fout, "extern bool_t xdr_%s(XDR *, %s%s);\n", name,  
147             name, pointerp ? "*" : "");  
148 }  
    unchanged_portion_omitted  
  
263 static void  
264 pfreeprocdef(char *name, char *vers)  
265 pfreeprocdef(char *name, char *vers, int mode)  
265 {  
266     f_print(fout, "extern int ");  
267     pfname(name, vers);  
269     if (mode == 1)  
268         f_print(fout, "_freeresult(SVCXPRT *, xdrproc_t, caddr_t);\n");  
271     else  
272         f_print(fout, "_freeresult();\n");  
269 }  
  
271 static void  
272 pprogramdef(definition *def)  
273 {  
274     version_list *vers;  
275     proc_list *proc;  
280     int i;  
281     char *ext;  
  
277     pargdef(def);  
  
279     puldefine(def->def_name, def->def.pr.prog_num, PROGRAM);  
280     for (vers = def->def.pr.versions; vers != NULL; vers = vers->next) {  
281         if (tblflag) {  
282             f_print(fout,  
283                 "extern struct rpcgen_table %s_%s_table[];\n",  
284                 lowercase(def->def_name), vers->vers_num);  
285             f_print(fout,  
286                 "extern int %s_%s_nproc;\n",  
287                 lowercase(def->def_name), vers->vers_num);  
288         }  
289         puldefine(vers->vers_name, vers->vers_num, VERSION);  
297         /*  
298          * Print out 2 definitions, one for ANSI-C, another for  
299          * old K & R C  
300          */  
302         if (!Cflag) {  
303             ext = "extern ";  
291             for (proc = vers->procs; proc != NULL;  
292                 proc = proc->next) {  
306                 if (!define_printed(proc, def->def.pr.versions))  
307                     puldefine(proc->proc_name,
```

```
2
```

```

308         f_print(fout, "%s", ext);
309         pprocdef(proc, vers, NULL, 0, 2);
310
311         if (mtflag) {
312             f_print(fout, "%s", ext);
313             pprocdef(proc, vers, NULL, 1, 2);
314         }
315     } else {
316         pfreeprocdef(def->def_name, vers->vers_num, 2);
317         for (i = 1; i < 3; i++) {
318             if (i == 1) {
319                 f_print(fout, "\n#ifndef defined(__STDC__)"
320                         " || defined(__cplusplus)\n");
321                 ext = "extern ";
322             } else {
323                 f_print(fout, "\n#else /* K&R C */\n");
324                 ext = "extern ";
325             }
326
327             for (proc = vers->procs; proc != NULL;
328                  proc = proc->next) {
329                 if (!define_printed(proc,
330                     def->def.pr.versions)) {
331                     puldefine(proc->proc_name,
332                             proc->proc_num, PROCEDURE);
333                 }
334                 f_print(fout, "extern ");
335                 pprocdef(proc, vers, "CLIENT **", 0);
336                 f_print(fout, "extern ");
337                 pprocdef(proc, vers, "struct svc_req **", 1);
338                 f_print(fout, "%s", ext);
339                 pprocdef(proc, vers, "CLIENT **", 0, i);
340                 f_print(fout, "%s", ext);
341                 pprocdef(proc, vers,
342                         "struct svc_req **", 1, i);
343             }
344             pfreeprocdef(def->def_name, vers->vers_num);
345             pfreeprocdef(def->def_name, vers->vers_num, i);
346         }
347     }
348     f_print(fout, "#endif /* K&R C */\n");
349 }
350
351 void
352 pprocdef(proc_list *proc, version_list *vp, char *addargtype, int server_p)
353 pprocdef(proc_list *proc, version_list *vp, char *addargtype, int server_p,
354           int mode)
355 {
356     if (mtflag) {
357         /* Print MT style stubs */
358         if (server_p)
359             f_print(fout, "bool_t ");
360         else
361             f_print(fout, "enum clnt_stat ");
362     } else {
363         ptype(proc->res_prefix, proc->res_type, 1);
364         f_print(fout, "* ");
365     }
366     if (server_p)
367         pvname_svc(proc->proc_name, vp->vers_num);
368     else
369         pvname(proc->proc_name, vp->vers_num);

```

```

368     /*
369      * mode 1 = ANSI-C, mode 2 = K&R C
370      */
371     if (mode == 1)
372         parlist(proc, addargtype);
373     else
374         f_print(fout, "();\n");
375 }
376
377 unchanged_portion_omitted

```

new/usr/src/cmd/rpcgen/rpc\_main.c

\*\*\*\*\*

31733 Sun Aug 3 11:09:16 2014

new/usr/src/cmd/rpcgen/rpc\_main.c

rpcgen should only produce ANSI code

\*\*\*\*\*

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
```

```
22 /*
23  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
24  *
25  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27 */
28 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29 /* All Rights Reserved */
30 /*
31 * University Copyright- Copyright (c) 1982, 1986, 1988
32 * The Regents of the University of California
33 * All Rights Reserved
34 *
35 * University Acknowledgment- Portions of this document are derived from
36 * software developed by the University of California, Berkeley, and its
37 * contributors.
38 */
39 /*
40 * rpc_main.c, Top level of the RPC protocol compiler.
41 */
42 */
```

```
44 #include <stdio.h>
45 #include <stdlib.h>
46 #include <string.h>
47 #include <strings.h>
48 #include <unistd.h>
49 #include <ctype.h>
50 #include <sys/types.h>
51 #include <sys/param.h>
52 #include <sys/file.h>
53 #include <sys/stat.h>
54 #include "rpc_parse.h"
55 #include "rpc_util.h"
56 #include "rpc_scan.h"

59 extern void write_sample_svc(definition *);
60 extern int write_sample_clnt(definition *);
61 extern void write_sample_clnt_main(void);
```

1

new/usr/src/cmd/rpcgen/rpc\_main.c

```
62 extern void reinitialize(void);
63 extern void crash(void);
64 extern void add_type(int, char *);
65 extern void add_sample_msg(void);

66 static void svc_output(char *, char *, int, char *);
67 static void clnt_output(char *, char *, int, char *);
68 static void c_output(char *, char *, int, char *);
69 static void mkfile_output(struct commandline *);
70 static void c_initialize(void);
71 static void h_output(char *, char *, int, char *);
72 static void s_output(int, char *[], char *, char *, int, char *, int, int);
73 static void l_output(char *, char *, int, char *);
74 static void t_output(char *, char *, int, char *);
75 static void do_registers(int, char *[]);
76 static uint_t parseargs(int, char *[], struct commandline *);
77 static void usage(void);
78 static void version_info(void);
79 static void options_usage(void);

80 #define EXTEND 1 /* alias for TRUE */
81 #define DONT_EXTEND 0 /* alias for FALSE */

82 #define SUNOS_CPP "/usr/lib/cpp"
83 static int cppDefined = 0; /* explicit path for C preprocessor */

84 static char *cmdname;

85 static char *svcclosetime = "120";
86 static char *CPP = SUNOS_CPP;
87 static char CPPFLAGS[] = "-C";
88 static char pathbuf[MAXPATHLEN + 1];
89 static char *allv[] = {
90     "rpcgen", "-s", "udp", "-s", "tcp",
91 };
92 static int allnc = sizeof(allnv)/sizeof(allnv[0]);

93 /*
94  * machinations for handling expanding argument list
95 */
96 static void addarg(char *); /* add another argument to the list */
97 static void putarg(int, char *); /* put argument at specified location */
98 static void clear_args(void); /* clear argument list */
99 static void checkfiles(char *, char *); /* check if out file already exists */

100 /*
101  * Support for inetd is now the default */
102 int inetdflag; /* Support for port monitors */
103 int logflag; /* Use syslog instead of fprintf for errors */
104 int tblflag; /* Support for dispatch table file */
105 int mtflag = 0; /* Support for MT */
106 int mtauto = 0; /* Enable automatic mode */
107 int rflag = 1; /* Eliminate tail recursion from structures */
108 #define INLINE 5
109 /* length at which to start doing an inline */

110 #define ARGLISTLEN 20
111 #define FIXEDARGS 2

112 static char *arglist[ARGLISTLEN];
113 static int argcount = FIXEDARGS;

114 int nonfatalerrors; /* errors */
115 int inetdflag; /* Support for inetd */
116 int pmflag; /* Support for port monitors */
117 int logflag; /* Use syslog instead of fprintf for errors */
118 int tblflag; /* Support for dispatch table file */
119 int mtflag = 0; /* Support for MT */
120 int mtauto = 0; /* Enable automatic mode */
121 int rflag = 1; /* Eliminate tail recursion from structures */
122 /* length at which to start doing an inline */
```

2

```

131 int inlinelen = INLINE;
132 /*
133  * Length at which to start doing an inline. INLINE = default
134  * if 0, no xdr_inline code
135 */
136
137 int indefinitewait; /* If started by port monitors, hang till it wants */
138 int exitnow; /* If started by port monitors, exit after the call */
139 int timerflag; /* TRUE if !indefinite && !exitnow */
140 int newstyle; /* newstyle of passing arguments (by value) */
141 int Cflag = 0; /* ANSI C syntax */
142 int CCflag = 0; /* C++ files */
143 static int allfiles; /* generate all files */
144 int tirpcflag = 1; /* generating code for tirpc, by default */
145 xdrfunc *xdrfunc_head = NULL; /* xdr function list */
146 xdrfunc *xdrfunc_tail = NULL; /* xdr function list */
147 pid_t childpid;
148
149 int
150 main(int argc, char *argv[])
151 {
152     struct commandline cmd;
153
154     (void) memset(&cmd, 0, sizeof (struct commandline));
155     clear_args();
156     if (!parseargs(argc, argv, &cmd))
157         usage();
158
159     /* Only the client and server side stubs are likely to be customized,
160      * so in that case only, check if the outfile exists, and if so,
161      * print an error message and exit.
162     */
163     if (cmd.Ssflag || cmd.Scflag || cmd.makefileflag)
164         checkfiles(cmd.infile, cmd.outfile);
165     else
166         checkfiles(cmd.infile, NULL);
167
168     if (cmd.cflag) {
169         c_output(cmd.infile, "-DRPC_XDR", DONT_EXTEND, cmd.outfile);
170     } else if (cmd.hflag) {
171         h_output(cmd.infile, "-DRPC_HDR", DONT_EXTEND, cmd.outfile);
172     } else if (cmd.lflag) {
173         l_output(cmd.infile, "-DRPC_CLNT", DONT_EXTEND, cmd.outfile);
174     } else if (cmd.sflag || cmd.mflag || (cmd.nflag)) {
175         s_output(argc, argv, cmd.infile, "-DRPC_SVC", DONT_EXTEND,
176                  cmd.outfile, cmd.mflag, cmd.nflag);
177     } else if (cmd.tflag) {
178         t_output(cmd.infile, "-DRPC_TBL", DONT_EXTEND, cmd.outfile);
179     } else if (cmd.Ssflag) {
180         svc_output(cmd.infile, "-DRPC_SERVER", DONT_EXTEND,
181                    cmd.outfile);
182     } else if (cmd.Scflag) {
183         clnt_output(cmd.infile, "-DRPC_CLIENT", DONT_EXTEND,
184                     cmd.outfile);
185     } else if (cmd.makefileflag) {
186         mkfile_output(&cmd);
187     } else {
188         /* the rescans are required, since cpp may effect input */
189         c_output(cmd.infile, "-DRPC_XDR", EXTEND, "_xdr.c");
190         reinitialize();
191         h_output(cmd.infile, "-DRPC_HDR", EXTEND, ".h");
192         reinitialize();
193         l_output(cmd.infile, "-DRPC_CLNT", EXTEND, "_clnt.c");
194         reinitialize();
195         if (inetdflag || !tirpcflag)

```

```

196         s_output(allc, allv, cmd.infile, "-DRPC_SVC", EXTEND,
197                  "_svc.c", cmd.mflag, cmd.nflag);
198     else
199         s_output(allnc, allnv, cmd.infile, "-DRPC_SVC",
200                  EXTEND, "_svc.c", cmd.mflag, cmd.nflag);
201     if (tblflag) {
202         reinitialize();
203         t_output(cmd.infile, "-DRPC_TBL", EXTEND, "_tbl.i");
204     }
205     if (allfiles) {
206         reinitialize();
207         svc_output(cmd.infile, "-DRPC_SERVER", EXTEND,
208                  "_server.c");
209         reinitialize();
210         clnt_output(cmd.infile, "-DRPC_CLIENT", EXTEND,
211                  "_client.c");
212     }
213     if (allfiles || (cmd.makefileflag == 1)) {
214         reinitialize();
215         mkfile_output(&cmd);
216     }
217     return (nonfatalerrors);
218 }
219
220
221
222 } unchanged_portion_omitted
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505 */
506 /* Compile into an XDR header file
507 */
508
509 static void
510 h_output(char *infile, char *define, int extend, char *outfile)
511 {
512     definition *def;
513     char *outfilename;
514     long tell;
515     char *guard;
516     list *l;
517     xdrfunc *xdrfuncp;
518     int i;
519
520     open_input(infile, define);
521     outfilename = extend ? extendfile(infile, outfile) : outfile;
522     open_output(infile, outfilename);
523     add_warning();
524     if (outfilename || infile)
525         guard = generate_guard(outfilename ? outfilename : infile);
526     else
527         guard = "STDIN_";
528
529     f_print(fout, "#ifndef %s\n#define %s\n", guard, guard);
530     f_print(fout, "#include <rpc/rpc.h>\n");
531
532     if (mtflag) {
533         f_print(fout, "#ifndef _KERNEL\n");
534         f_print(fout, "#include <synch.h>\n");
535         f_print(fout, "#include <thread.h>\n");
536         f_print(fout, "#endif /* !_KERNEL */\n");
537     }
538
539     /* put the C++ support */

```

```

541     if (!CCflag) {
541     if (Cflag && !CCflag) {
542         f_print(fout, "\n#ifndef __cplusplus\n");
543         f_print(fout, "extern \"C\" {\n");
544         f_print(fout, "#endif\n}\n");
545     }
546
547     /* put in a typedef for quadprecision. */
547     /* put in a typedef for quadprecision. Only with Cflag */
548
549     /*
550      * declaration of struct rpcgen_table must go before
551      * the definition of arrays like *_1_table[]
552      */
553     if (tblflag) {
554         f_print(fout, rpcgen_table_dcl1);
555         if (tirpcflag)
556             f_print(fout, rpcgen_table_proc);
557         else
558             f_print(fout, rpcgen_table_proc_b);
559     }
560
561     f_print(fout, rpcgen_table_dcl2);
562
563     tell = ftell(fout);
564
565     /* print data definitions */
566     while (def = get_definition())
567         print_datadef(def);
568
569     /*
570      * print function declarations.
571      * Do this after data definitions because they might be used as
572      * arguments for functions
573      */
574     for (l = defined; l != NULL; l = l->next)
575         print_funcdef(l->val);
576
577     /* Now print all xdr func declarations */
578     if (xdrfunc_head != NULL) {
579         f_print(fout, "\n/* the xdr functions */\n");
580
581         if (CCflag) {
582             f_print(fout, "\n#ifndef __cplusplus\n");
583             f_print(fout, "extern \"C\" {\n");
584             f_print(fout, "#endif\n");
585         }
586
587         f_print(fout, "\n");
588         if (!Cflag) {
589             xdrfuncp = xdrfunc_head;
590             while (xdrfuncp != NULL) {
591                 print_xdr_func_def(xdrfuncp->name,
592                     xdrfuncp->pointerp, 2);
593                 xdrfuncp = xdrfuncp->next;
594             }
595         } else {
596             for (i = 1; i < 3; i++) {
597                 if (i == 1)
598                     f_print(fout,
599                         "\n#ifndef __cplusplus\n");
600                 else
601                     f_print(fout, "\n#else /* K&R C */\n");
602
603             xdrfuncp = xdrfunc_head;
604             while (xdrfuncp != NULL) {
605                 print_xdr_func_def(xdrfuncp->name, xdrfuncp->pointerp);
606                 print_xdr_func_def(xdrfuncp->name,

```

```

607                     xdrfuncp->pointerp, i);
608
609             xdrfuncp = xdrfuncp->next;
610         }
611         f_print(fout, "\n");
612     }
613     f_print(fout, "\n#endif /* K&R C */\n");
614
615     if (extend && tell == ftell(fout)) {
616         (void) unlink(outfilename);
617     }
618
619     if (Cflag) {
620         f_print(fout, "\n#ifndef __cplusplus\n");
621         f_print(fout, "}\n");
622         f_print(fout, "#endif\n");
623
624     f_print(fout, "\n#endif /* !__s */\n", guard);
625
626     /*
627      * Compile into an RPC service
628      */
629     static void
630     s_output(int argc, char *argv[], char *infile, char *define, int extend,
631              char *outfile, int nomain, int netflag)
632     {
633         char *include;
634         definition *def;
635         int foundprogram = 0;
636         char *outfilename;
637
638         open_input(infile, define);
639         outfilename = extend ? extendfile(infile, outfile) : outfile;
640         open_output(infile, outfilename);
641         add_warning();
642         if (infile && (include = extendfile(infile, ".h")) != NULL) {
643             f_print(fout, "#include \"%s\"\n", include);
644             free(include);
645         } else
646             f_print(fout, "#include <rpc/rpc.h>\n");
647
648         f_print(fout, "#include <stdio.h>\n");
649         f_print(fout, "#include <stdlib.h> /* getenv, exit */\n");
650         f_print(fout, "#include <signal.h>\n");
651
652         if (Cflag) {
653             f_print(fout,
654                 "#include <rpc/pmap_clnt.h> /* for pmap_unset */\n");
655             f_print(fout, "#include <string.h> /* strcmp */\n");
656
657         }
658         if (strcmp(svc closetime, "-1") == 0)
659             indefinitewait = 1;
660         else if (strcmp(svc closetime, "0") == 0)
661             exitnow = 1;
662         else if (inetdflag || pmfflag)
663             timerflag = 1;
664
665         if (!tirpcflag && inetdflag)
666             f_print(fout, "#include <sys/termios.h> /* TIOCNNOTTY */\n");
667         if (inetdflag || pmfflag)
668             if (Cflag && (inetdflag || pmfflag))
669                 if (tirpcflag)

```

```

647     f_print(fout, "#include <unistd.h> /* setsid */\n");
648     if (tirpcflag)
649         f_print(fout, "#include <sys/types.h>\n");
650
651     f_print(fout, "#include <memory.h>\n");
652     f_print(fout, "#include <stropts.h>\n");
653     if (inetdflag || !tirpcflag) {
654         f_print(fout, "#include <sys/socket.h>\n");
655         f_print(fout, "#include <netinet/in.h>\n");
656         f_print(fout, "#include <rpc/svc_soc.h>\n");
657     }
658
659     if ((netflag || pmflag) && tirpcflag && !nomain)
660         f_print(fout, "#include <netconfig.h>\n");
661     if (tirpcflag)
662         f_print(fout, "#include <sys/resource.h> /* rlimit */\n");
663     if (logflag || inetdflag || pmflag)
664         f_print(fout, "#include <syslog.h>\n");
665
666     /* for ANSI-C */
667     f_print(fout, "\n#ifndef SIG_PF\n"
668             "#define SIG_PF void(*)(int)\n#endif\n");
669     if (Cflag)
670         f_print(fout,
671                 "\n#ifndef SIG_PF\n#define SIG_PF void(*)(\n"
672                 "int)\n#endif\n");
673
674     f_print(fout, "\n#ifndef DEBUG\n#define RPC_SVC_FG\n#endif\n");
675     if (timerflag)
676         f_print(fout, "\n#define _RPCSVC_CLOSEDOWN %s\n",
677                 svclosetime);
678     while (def = get_definition())
679         foundprogram |= (def->def_kind == DEF_PROGRAM);
680     if (extend && !foundprogram) {
681         (void) unlink(outfilename);
682         return;
683     }
684     write_most(infile, netflag, nomain);
685     if (!nomain) {
686         if (!do_registers(argc, argv)) {
687             if (outfile)
688                 (void) unlink(outfile);
689             usage();
690         }
691         write_rest();
692     }
693
694     /* generate client side stubs
695 */
696     static void
697     l_output(char *infile, char *define, int extend, char *outfile)
698     {
699         char *include;
700         definition *def;
701         int foundprogram = 0;
702         char *outfilename;
703
704         open_input(infile, define);
705         outfilename = extend ? extendfile(infile, outfile) : outfile;
706         open_output(infile, outfilename);
707         add_warning();
708         if (Cflag)
709             f_print(fout, "#include <memory.h> /* for memset */\n");
710         if (infile && (include = extendfile(infile, ".h")))
711             f_print(fout, "#include <%s>\n", include);
712     }

```

```

707         f_print(fout, "#include \"%s\"\n", include);
708         free(include);
709     } else
710         f_print(fout, "#include <rpc/rpc.h>\n");
711
712     f_print(fout, "#ifndef __KERNEL__\n");
713     f_print(fout, "#include <stdio.h>\n");
714     f_print(fout, "#include <stdlib.h> /* getenv, exit */\n");
715     f_print(fout, "#endif /* __KERNEL__ */\n");
716
717     while (def = get_definition())
718         foundprogram |= (def->def_kind == DEF_PROGRAM);
719     if (extend && !foundprogram) {
720         (void) unlink(outfilename);
721         return;
722     }
723     write_stubs();
724 }

unchanged_portion_omitted

1007 /*
1008  * Parse command line arguments
1009  */
1010 static uint_t
1011 parseargs(int argc, char *argv[], struct commandline *cmd)
1012 {
1013     int i;
1014     int j;
1015     char c, ch;
1016     char flag[(1 << 8 * sizeof (char))];
1017     int nflags;

1018     cmdname = argv[0];
1019     cmd->infile = cmd->outfile = NULL;
1020     if (argc < 2)
1021         return (0);
1022     allfiles = 0;
1023     flag['c'] = 0;
1024     flag['h'] = 0;
1025     flag['l'] = 0;
1026     flag['m'] = 0;
1027     flag['o'] = 0;
1028     flag['s'] = 0;
1029     flag['n'] = 0;
1030     flag['t'] = 0;
1031     flag['S'] = 0;
1032     flag['C'] = 0;
1033     flag['M'] = 0;

1034     for (i = 1; i < argc; i++) {
1035         if (argv[i][0] != '-') {
1036             if (cmd->infile)
1037                 f_print(stderr,
1038                         "Cannot specify more than one input file.\n");
1039             return (0);
1040         }
1041         cmd->infile = argv[i];
1042     }
1043     cmd->infile = argv[i];
1044
1045     } else {
1046         for (j = 1; argv[i][j] != 0; j++) {
1047             c = argv[i][j];
1048             switch (c) {
1049                 case 'a':
1050                     allfiles = 1;
1051                     break;
1052                 case 'c':
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971

```

new/usr/src/cmd/rpcgen/rpc\_main.c

9

```

1053
1054
1055
1056
1057         case 'h':
1058             if (flag[c])
1059                 return (0);
1060             flag[c] = 1;
1061             break;
1062         case 'S':
1063             /*
1064             * sample flag: Ss or Sc.
1065             * Ss means set flag['S'];
1066             * Sc means set flag['C'];
1067             * Sm means set flag['M'];
1068             */
1069         ch = argv[i][++j]; /* get next char */
1070         if (ch == 's')
1071             ch = 'S';
1072         else if (ch == 'c')
1073             ch = 'C';
1074         else if (ch == 'm')
1075             ch = 'M';
1076         else
1077             return (0);

1078         if (flag[ch])
1079             return (0);
1080         flag[ch] = 1;
1081         break;
1082     case 'C': /* ANSI C syntax (default) */
1083     case 'C': /* ANSI C syntax */
1084         Cflag = 1;
1085         ch = argv[i][j+1]; /* get next char */

1086         if (ch != 'C')
1087             break;
1088         /* Undocumented C++ mode */
1089         CCflag = 1;
1090         break;
1091     case 'b':
1092         /*
1093         * Turn TIRPC flag off for
1094         * generating backward compatible
1095         * code
1096         */
1097         tirpcflag = 0;
1098         break;

1099     case 'I':
1100         inetdflag = 1;
1101         break;
1102     case 'N':
1103         newstyle = 1;
1104         break;
1105     case 'L':
1106         logflag = 1;
1107         break;
1108     case 'K':
1109         if (++i == argc)
1110             return (0);
1111         svcclosetime = argv[i];
1112         goto nextarg;
1113     case 'T':
1114         tblflag = 1;
1115         break;
1116     case 'A':

```

new/usr/src/cmd/rpcgen/rpc\_main.c

```

1117                         mtauto = 1;
1118                         /* FALLTHRU */
1119         case 'M':
1120             mtflag = 1;
1121             break;
1122         case 'i':
1123             if (++i == argc)
1124                 return (0);
1125             inlinelen = atoi(argv[i]);
1126             goto nextarg;
1127         case 'n':
1128         case 'o':
1129         case 's':
1130             if (argv[i][j - 1] != '-' || argv[i][j + 1] != 0)
1131                 return (0);
1132             flag[c] = 1;
1133             if (++i == argc)
1134                 return (0);
1135             if (c == 'o') {
1136                 if (cmd->outfile)
1137                     return (0);
1138                 cmd->outfile = argv[i];
1139             }
1140             goto nextarg;
1141         case 'D':
1142             if (argv[i][j - 1] != '-')
1143                 return (0);
1144             (void) addarg(argv[i]);
1145             goto nextarg;
1146         case 'v':
1147             version_info();
1148             return (0);
1149         case 'Y':
1150             if (++i == argc)
1151                 return (0);
1152             (void) strcpy(pathbuf, argv[i]);
1153             (void) strcat(pathbuf, "/cpp");
1154             CPP = pathbuf;
1155             cppDefined = 1;
1156             goto nextarg;
1157         case 'r':
1158             rflag = !rflag;
1159             break;
1160         default:
1161             return (0);
1162         }
1163     }
1164 }
1165 nextarg:
1166     ;
1167 }
1168 }

1169 cmd->cflag = flag['c'];
1170 cmd->hflag = flag['h'];
1171 cmd->lflag = flag['l'];
1172 cmd->mflag = flag['m'];
1173 cmd->nflag = flag['n'];
1174 cmd->sflag = flag['s'];
1175 cmd->tflag = flag['t'];
1176 cmd->Ssflag = flag['S'];
1177 cmd->Scflag = flag['C'];
1178 cmd->makefileflag = flag['M'];

1179 if (tirpcfflag) {
1180     if (inetdflag) {

```

```
1183         f_print(stderr,
1184                 "Cannot use -I flag without -b flag.\n");
1185     }
1186     pmflag = 1;
1187 } else { /* 4.1 mode */
1188     pmflag = 0; /* set pmflag only in tirpcmode */
1189     inetdflag = 1; /* inetdflag is TRUE by default */
1190     if (cmd->nflag) { /* netid needs TIRPC */
1191         f_print(stderr,
1192                 "Cannot use netid flag without TIRPC.\n");
1193         return (0);
1194     }
1195 }
1196
1197 if (newstyle && (tblflag || cmd->tflag)) {
1198     f_print(stderr, "Cannot use table flags with newstyle.\n");
1199     return (0);
1200 }
1201
1202 /* check no conflicts with file generation flags */
1203 nflags = cmd->cflag + cmd->hflag + cmd->lflag + cmd->mflag +
1204     cmd->sflag + cmd->nflag + cmd->tflag + cmd->Ssflag +
1205     cmd->Scflag + cmd->makefileflag;
1206
1207 if (nflags == 0) {
1208     if (cmd->outfile != NULL || cmd->infile == NULL)
1209         return (0);
1210 } else if (cmd->infile == NULL &&
1211     (cmd->Ssflag || cmd->Scflag || cmd->makefileflag)) {
1212     f_print(stderr, "\"infile\" is required for template"
1213             " generation flags.\n");
1214     return (0);
1215 }
1216 if (nflags > 1) {
1217     f_print(stderr,
1218             "Cannot have more than one file generation flag.\n");
1219     return (0);
1220 }
1221
1222 return (1);
1223 }
```

unchanged portion omitted

new/usr/src/cmd/rpcgen/rpc\_sample.c

```
*****
8516 Sun Aug 3 11:09:17 2014
new/usr/src/cmd/rpcgen/rpc_sample.c
rpcgen should only produce ANSI code
*****
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
22 /*
23 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
24 *
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26 * Use is subject to license terms.
27 */
28 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29 /* All Rights Reserved */
30 /*
31 * University Copyright- Copyright (c) 1982, 1986, 1988
32 * The Regents of the University of California
33 * All Rights Reserved
34 *
35 * University Acknowledgment- Portions of this document are derived from
36 * software developed by the University of California, Berkeley, and its
37 * contributors.
38 */
40 /*
41 * rpc_sample.c, Sample client-server code outputter
42 * for the RPC protocol compiler
43 */
45 #include <stdio.h>
46 #include <string.h>
47 #include "rpc_parse.h"
48 #include "rpc_util.h"

51 static char RQSTP[] = "rqstp";
53 extern void printarglist(proc_list *, char *, char *, char *);
55 static void write_sample_client(char *, version_list *);
56 static void write_sample_server(defination *);
57 static void return_type(proc_list *);

59 void
60 write_sample_svc(defination *def)
61 {
```

1

new/usr/src/cmd/rpcgen/rpc\_sample.c

```
62     if (def->def_kind != DEF_PROGRAM)
63         return;
64     write_sample_server(def);
65 }
_____unchanged_portion_omitted_____
83 static void
84 write_sample_client(char *program_name, version_list *vp)
85 {
86     proc_list *proc;
87     int i;
88     decl_list *l;

89     f_print(fout, "\n\nvoid\n");
90     pfname(program_name, vp->vers_num);
91     if (Cflag)
92         f_print(fout, "(char *host)\n{\n");
93     else
94         f_print(fout, "(host)\ntchar *host;\n{\n");
95     f_print(fout, "\tCLIENT *clnt;\n");

96     i = 0;
97     for (proc = vp->procs; proc != NULL; proc = proc->next) {
98         f_print(fout, "\t");
99         if (mtflag) {
100             f_print(fout, "enum clnt_stat retrval_%d;\n", ++i);
101             if (!streq(proc->res_type, "oneway")) {
102                 f_print(fout, "\t");
103                 if (!streq(proc->res_type, "void"))
104                     ptype(proc->res_prefix,
105                           proc->res_type, 1);
106                 else
107                     f_print(fout, "void *");
108                 f_print(fout, "result_%d;\n", i);
109             } else {
110                 ptype(proc->res_prefix, proc->res_type, 1);
111                 f_print(fout, " *result_%d;\n", ++i);
112             }
113             /* print out declarations for arguments */
114             if (proc->arg_num < 2 && !newstyle) {
115                 f_print(fout, "\t");
116                 if (!streq(proc->args.decls->decl.type, "void"))
117                     ptype(proc->args.decls->decl.prefix,
118                           proc->args.decls->decl.type, 1);
119                 else
120                     /* cannot have "void" type */
121                     f_print(fout, "char * ");
122                 f_print(fout, "\t");
123                 pfname(proc->proc_name, vp->vers_num);
124                 f_print(fout, "_arg;\n");
125             } else if (!streq(proc->args.decls->decl.type, "void")) {
126                 for (l = proc->args.decls; l != NULL; l = l->next) {
127                     f_print(fout, "\t");
128                     ptype(l->decl.prefix, l->decl.type, 1);
129                     if (strcmp(l->decl.type, "string") == 1)
130                         f_print(fout, " ");
131                     pfname(proc->proc_name, vp->vers_num);
132                     f_print(fout, "_%s;\n", l->decl.name);
133                 }
134             }
135         }
136         /* generate creation of client handle */
137         f_print(fout, "\n#ifndef tDEBUG\n");
138         f_print(fout, "\tclnt = clnt_create(host, %s, %s, \"%s\");\n",
139             program_name, vp->vers_num, host);
```

2

new/usr/src/cmd/rpcgen/rpc\_sample.c

3

```

140     program_name, vp->vers_name, tirpcflg? "netpath" : "udp");
141     f_print(fout, "\tif (clnt == (CLIENT *) NULL) {\n");
142     f_print(fout, "\t\tclnt_pcreateerror(host);\n");
143     f_print(fout, "\t\texit(1);\n\t}\n");
144     f_print(fout, "#endif\n/* DEBUG */\n");
145
146     /* generate calls to procedures */
147     i = 0;
148     for (proc = vp->procs; proc != NULL; proc = proc->next) {
149         if (mtfflag)
150             f_print(fout, "\tretval_%d = ", ++i);
151         else
152             f_print(fout, "\tresult_%d = ", ++i);
153         pvnname(proc->proc_name, vp->vers_num);
154         if (proc->arg_num < 2 && !newstyle) {
155             f_print(fout, "(");
156             if (streq(proc->args.decls->decl.type, "void"))
157                 /* cast to void */
158                 f_print(fout, "(void *)");
159             f_print(fout, "&");
160             pvnname(proc->proc_name, vp->vers_num);
161             if (mtfflag) {
162                 if (streq(proc->res_type, "oneway"))
163                     f_print(fout, "_args, clnt);\n");
164                 else
165                     f_print(fout,
166                             "_arg, &result_%d, clnt);\n", i);
167             } else
168                 f_print(fout, "_arg, clnt);\n");
169
170         } else if (streq(proc->args.decls->decl.type, "void")) {
171             if (mtfflag) {
172                 if (streq(proc->res_type, "oneway"))
173                     f_print(fout, "(clnt);\n");
174                 else
175                     f_print(fout,
176                             "(&result_%d, clnt);\n", i);
177             } else
178                 f_print(fout, "(clnt);\n");
179         } else {
180             f_print(fout, "(");
181             for (l = proc->args.decls; l != NULL; l = l->next) {
182                 pvnname(proc->proc_name, vp->vers_num);
183                 f_print(fout, "_%s, ", l->decl.name);
184             }
185             if (mtfflag) {
186                 if (!streq(proc->res_type, "oneway"))
187                     f_print(fout, "&result_%d, ", i);
188             }
189
190             f_print(fout, "clnt);\n");
191         }
192         if (mtfflag) {
193             f_print(fout, "\tif (retval_%d != RPC_SUCCESS) {\n", i);
194         } else {
195             f_print(fout, "\tif (result_%d == (", i);
196             ptype(proc->res_prefix, proc->res_type, 1);
197             f_print(fout, ") NULL) {\n");
198         }
199         f_print(fout, "\t\tclnt_perror(clnt, \"call failed\");\n");
200         f_print(fout, "\t}\n");
201     }
202
203     f_print(fout, "#ifndef\n\tDEBUG\n#endif\n");
204     f_print(fout, "\tclnt_destroy(clnt);\n");
205     f_print(fout, "#endif\n/* DEBUG */\n");

```

new/usr/src/cmd/rpcgen/rpc\_sample.c

```

206         f_print(fout, "}\n");
207     }

208 static void
209 write_sample_server(definition *def)
210 {
211     version_list *vp;
212     proc_list *proc;

213     for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
214         for (proc = vp->procs; proc != NULL; proc = proc->next) {
215             f_print(fout, "\n");
216             if (!mtflag) {
217                 return_type(proc);
218                 f_print(fout, "*\n");
219             } else {
220                 f_print(fout, "bool_t\n");
221             }
222             if (Cflag || mtflag)
223                 pvname_svc(proc->proc_name, vp->vers_num);
224             else
225                 pvname(proc->proc_name, vp->vers_num);
226             printarglist(proc, "result", RQSTP, "struct svc_req *");
227
228             f_print(fout, "{\n");
229             if (!mtflag) {
230                 f_print(fout, "\tstatic ");
231                 if (!strcmp(proc->res_type, "void")) &&
232                     (!strcmp(proc->res_type, "oneway")))
233                         return_type(proc);
234                 else
235                     f_print(fout, "char *");
236                 /* cannot have void type */
237                 f_print(fout, " result;\n");
238             }
239
240             f_print(fout, "\n\t/*\n\t * insert server code "
241             "here\n\t */\n");
242
243             if (!mtflag)
244                 if (!strcmp(proc->res_type, "void"))
245                     f_print(fout,
246                             "\treturn (&result);\n");
247                 else /* cast back to void */
248                     f_print(fout, "\treturn((void *) "
249                             "&result);\n");
250             else
251                 f_print(fout, "\treturn (retval);\n");
252
253             /* put in sample freeing routine */
254             if (mtflag) {
255                 f_print(fout, "\nint\n");
256                 pvname(def->def_name, vp->vers_num);
257                 if (Cflag)
258                     f_print(fout, "_freeresult(SVCXPRT *transp,"
259                             " xdrproc_t xdr_result,"
260                             " caddr_t result)\n");
261                 else {
262                     f_print(fout, "_freeresult(transp, xdr_result,"
263                             " result)\n");
264                     f_print(fout, "\tSVCXPRT *transp;\n");
265                     f_print(fout, "\txdrproc_t xdr_result;\n");
266                     f_print(fout, "\tcaddr_t result;\n");
267                 }
268             f_print(fout, "{\n"

```

```
261             "\t(void) xdr_free(xdr_result, result);\n"
262             "\n\t/*\n\t * Insert additional freeing\n"
263             " code here, if needed\n\t */\n"
264             "\n\n\treturn (TRUE);\n}\n");
265     }
266 }
267 }



---

unchanged_portion_omitted_

285 void
286 write_sample_clnt_main(void)
287 {
288     list *l;
289     definition *def;
290     version_list *vp;

292     f_print(fout, "\n\n");
293     if (Cflag)
294         f_print(fout, "int\nmain(int argc, char *argv[])\n{\n");
295     else
296         f_print(fout, "int\nmain(argc, argv)\n\tint argc;\n"
297                 "\tchar *argv[];\n{\n");

298     f_print(fout, "\tchar *host;");
299     f_print(fout, "\n\n\tif (argc < 2) {\n");
300     f_print(fout, "\n\t\tprintf(\"usage: %s server_host\\n\", "
301             " argv[0]);\n");
302     f_print(fout, "\t\texit(1);\n\t}");
303     f_print(fout, "\n\thost = argv[1];\n");

304     for (l = defined; l != NULL; l = l->next) {
305         def = l->val;
306         if (def->def_kind != DEF_PROGRAM)
307             continue;
308         for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
309             f_print(fout, "\t");
310             pvname(def->def_name, vp->vers_num);
311             f_print(fout, "(host);\n");
312         }
313     }
314     f_print(fout, "}\n");
315 }



---

unchanged_portion_omitted_
```

new/usr/src/cmd/rpcgen/rpc\_svcout.c

```
*****
30210 Sun Aug 3 11:09:17 2014
new/usr/src/cmd/rpcgen/rpc_svcout.c
rpcgen should only produce ANSI code
*****
```

```
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
24 *
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26 * Use is subject to license terms.
27 */
28 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29 /* All Rights Reserved */
30 /*
31 * University Copyright- Copyright (c) 1982, 1986, 1988
32 * The Regents of the University of California
33 * All Rights Reserved
34 *
35 * University Acknowledgment- Portions of this document are derived from
36 * software developed by the University of California, Berkeley, and its
37 * contributors.
38 */

40 /*
41 * rpc_svcout.c, Server-skeleton outputter for the RPC protocol compiler
42 */
43 #include <stdio.h>
44 #include <string.h>
45 #include <stdarg.h>
46 #include "rpc_parse.h"
47 #include "rpc_util.h"

49 extern int nullproc(proc_list *);

51 static char RQSTP[] = "rqstp";
52 static char TRANSP[] = "transp";
53 static char ARG[] = "argument";
54 static char RESULT[] = "result";
55 static char ROUTINE[] = "local";
56 static char RETVAL[] = "retval";

58 #define ERRBUFLLEN 256

60 static void internal_proctype(proc_list *);
61 static void write_real_program(definition *);
```

1

new/usr/src/cmd/rpcgen/rpc\_svcout.c

```
62 static void write_programs(char *);
63 static void write_program(definition *, char *);
64 static void printerr(char *, char *);
65 static void write_svc_aux(int);
66 static void printf(char *, char *, char *, char *, char *);
67 static void write_inetmost(char *);
68 static void print_return(char *);
69 static void print_pmapunset(char *);
70 static void print_err_message(const char *, const char *, ...);
71 static void write_msg_out(void);
72 static void write_timeout_func(void);
73 static void write_pm_most(char *, int);
74 static void write_rpc_svc_fg(char *, char *);
75 static void open_log_file(char *, char *);

77 static void
78 p_xdrfunc(char *rname, char *typename)
79 {
80     if (Cflag) {
81         f_print(fout, "\t\t_xdr_%s = (xdrproc_t)\n", rname);
82         f_print(fout, "\t\t\t_xdr_%s;\n", stringfix(typename));
83     } else {
84         f_print(fout, "\t\t_xdr_%s = xdr_%s;\n",
85                 rname, stringfix(typename));
86     }
87 }
```

unchanged\_portion\_omitted

```
275 /*
276 * write the rest of the service
277 */
278 void
279 write_rest(void)
280 {
281     f_print(fout, "\n");
282     if (inetdflag) {
283         f_print(fout, "\tif (%s == (SVCXPRT *)NULL) {\n", TRANSP);
284         print_err_message("\t\t", "could not create a handle");
285         f_print(fout, "\t\texit(1);\n");
286         f_print(fout, "\t}\n");
287     } else {
288         f_print(fout, "\tif (_rpcpmstart) {\n");
289         if (mtflag) {
290             f_print(fout,
291                     "\t\tif (thr_create(NULL, 0, closedown, NULL, 0, NULL) != 0) {\n");
292             print_err_message("\t\t\t",
293                             "cannot create closedown thread");
294             f_print(fout, "\t\t\texit(1);\n");
295             f_print(fout, "\t\t}\n");
296             f_print(fout, "\t\t}\n");
297         } else {
298             f_print(fout,
299                     "\t\t(void) signal(SIGALRM, %s closedown);\n",
300                     "(SIG_PF)");
301             Cflag? "(SIG_PF)":"(void(*)())");
302             f_print(fout,
303                     "\t\talarm(_RPCSVC_CLOSEDOWN/2);\n");
304             f_print(fout, "\t\t}\n");
305         }
306     }
307     f_print(fout, "\tsvc_run();\n");
308     print_err_message("\t", "svc_run returned");
309     f_print(fout, "\texit(1);\n");
310     f_print(fout, "\t/* NOTREACHED */\n");
311 }
```

2

```

312 }
313 unchanged_portion_omitted_
314
315 /*
316  * write out definition of internal function (e.g. _printmsg_1(...))
317  * which calls server's defintion of actual function (e.g. printmsg_1(...)).
318  * Unpacks single user argument of printmsg_1 to call-by-value format
319  * expected by printmsg_1.
320 */
321 static void
322 write_real_program(definition *def)
323 {
324     version_list *vp;
325     proc_list *proc;
326     decl_list *l;
327
328     if (!newstyle)
329         return; /* not needed for old style */
330     for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
331         for (proc = vp->procs; proc != NULL; proc = proc->next) {
332             int oneway = streq(proc->res_type, "oneway");
333
334             f_print(fout, "\n");
335             if (proc->arg_num < 2 &&
336                 streq(proc->args.decls->decl.type, "void")) {
337                 f_print(fout, /* ARGSUSED */"\n");
338             }
339             if (!mtflag)
340                 internal_proctype(proc);
341             else
342                 f_print(fout, "int");
343             f_print(fout, "\n");
344             pvname(proc->proc_name, vp->vers_num);
345
346             if (Cflag) {
347                 f_print(fout, "(\n");
348                 f_print(fout, "    ");
349                 /* arg name */
350                 if (proc->arg_num > 1)
351                     /* LINTED variable format */
352                     f_print(fout, proc->args.argname);
353                 else
354                     ptype(proc->args.decls->decl.prefix,
355                           proc->args.decls->decl.type, 0);
356                 f_print(fout, " *argp,\n");
357                 if (mtflag) {
358                     f_print(fout, "    ");
359                     ptype(proc->res_prefix, proc->res_type, 1);
360                     ptype(proc->res_prefix,
361                           proc->res_type, 1);
362                     f_print(fout, "*%s,\n", RESULT);
363                 }
364                 f_print(fout, "    struct svc_req *%s)\n", RQSTP);
365                 f_print(fout, "    struct svc_req *%s)\n",
366                         RQSTP);
367
368             } else {
369                 if (mtflag)
370                     f_print(fout, "(argp, %s, %s)\n",
371                             RESULT, RQSTP);
372                 else
373                     f_print(fout, "(argp, %s)\n", RQSTP);
374                 /* arg name */
375                 if (proc->arg_num > 1)
376                     f_print(fout, "\t%s *argp,\n",
377                           proc->args.argname);
378             }
379
380             if (Cflag) {
381                 f_print(fout, "(\n");
382                 f_print(fout, "    ");
383                 /* arg name */
384                 if (proc->arg_num > 1)
385                     f_print(fout, "\t%s *argp,\n",
386                           proc->args.argname);
387             }
388
389             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
390             f_print(fout, "    struct svc_req *%s)\n",
391                         RQSTP);
392
393             if (mtflag)
394                 f_print(fout, "(argp, %s, %s)\n",
395                             RESULT, RQSTP);
396             else
397                 f_print(fout, "(argp, %s)\n", RQSTP);
398             /* arg name */
399             if (proc->arg_num > 1)
400                 f_print(fout, "\t%s *argp,\n",
401                           proc->args.argname);
402
403             if (Cflag) {
404                 f_print(fout, "(\n");
405                 f_print(fout, "    ");
406                 /* arg name */
407                 if (proc->arg_num > 1)
408                     f_print(fout, "\t%s *argp,\n",
409                           proc->args.argname);
410             }
411
412             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
413             f_print(fout, "    struct svc_req *%s)\n",
414                         RQSTP);
415
416             if (Cflag) {
417                 f_print(fout, "(\n");
418                 f_print(fout, "    ");
419                 /* arg name */
420                 if (proc->arg_num > 1)
421                     f_print(fout, "\t%s *argp,\n",
422                           proc->args.argname);
423             }
424
425             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
426             f_print(fout, "    register SVCXPRT *%s)\n", TRANSP);
427
428             if (Cflag) {
429                 f_print(fout, "(\n");
430                 f_print(fout, "    ");
431                 /* arg name */
432                 if (proc->arg_num > 1)
433                     f_print(fout, "\t%s *argp,\n",
434                           proc->args.argname);
435             }
436
437             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
438             f_print(fout, "    register SVCXPRT *%s)\n",
439                         TRANSP);
440
441             if (Cflag) {
442                 f_print(fout, "(\n");
443                 f_print(fout, "    ");
444                 /* arg name */
445                 if (proc->arg_num > 1)
446                     f_print(fout, "\t%s *argp,\n",
447                           proc->args.argname);
448             }
449
450             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
451             f_print(fout, "    register SVCXPRT *%s)\n",
452                         TRANSP);
453
454             if (Cflag) {
455                 f_print(fout, "(\n");
456                 f_print(fout, "    ");
457                 /* arg name */
458                 if (proc->arg_num > 1)
459                     f_print(fout, "\t%s *argp,\n",
460                           proc->args.argname);
461             }
462
463             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
464             f_print(fout, "    register SVCXPRT *%s)\n",
465                         TRANSP);
466
467             if (Cflag) {
468                 f_print(fout, "(\n");
469                 f_print(fout, "    ");
470                 /* arg name */
471                 if (proc->arg_num > 1)
472                     f_print(fout, "\t%s *argp,\n",
473                           proc->args.argname);
474             }
475
476             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
477             f_print(fout, "    register SVCXPRT *%s)\n",
478                         TRANSP);
479
480             if (Cflag) {
481                 f_print(fout, "(\n");
482                 f_print(fout, "    ");
483                 /* arg name */
484                 if (proc->arg_num > 1)
485                     f_print(fout, "\t%s *argp,\n",
486                           proc->args.argname);
487             }
488
489             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
490             f_print(fout, "    register SVCXPRT *%s)\n",
491                         TRANSP);
492
493             if (Cflag) {
494                 f_print(fout, "(\n");
495                 f_print(fout, "    ");
496                 /* arg name */
497                 if (proc->arg_num > 1)
498                     f_print(fout, "\t%s *argp,\n",
499                           proc->args.argname);
500             }
501
502             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
503             f_print(fout, "    register SVCXPRT *%s)\n",
504                         TRANSP);
505
506             if (Cflag) {
507                 f_print(fout, "(\n");
508                 f_print(fout, "    ");
509                 /* arg name */
510                 if (proc->arg_num > 1)
511                     f_print(fout, "\t%s *argp,\n",
512                           proc->args.argname);
513             }
514
515             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
516             f_print(fout, "    register SVCXPRT *%s)\n",
517                         TRANSP);
518
519             if (Cflag) {
520                 f_print(fout, "(\n");
521                 f_print(fout, "    ");
522                 /* arg name */
523                 if (proc->arg_num > 1)
524                     f_print(fout, "\t%s *argp,\n",
525                           proc->args.argname);
526             }
527
528             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
529             f_print(fout, "    register SVCXPRT *%s)\n",
530                         TRANSP);
531
532             if (Cflag) {
533                 f_print(fout, "(\n");
534                 f_print(fout, "    ");
535                 /* arg name */
536                 if (proc->arg_num > 1)
537                     f_print(fout, "\t%s *argp,\n",
538                           proc->args.argname);
539             }
540
541             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
542             f_print(fout, "    register SVCXPRT *%s)\n",
543                         TRANSP);
544
545             if (Cflag) {
546                 f_print(fout, "(\n");
547                 f_print(fout, "    ");
548                 /* arg name */
549                 if (proc->arg_num > 1)
550                     f_print(fout, "\t%s *argp,\n",
551                           proc->args.argname);
552             }
553
554             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
555             f_print(fout, "    register SVCXPRT *%s)\n",
556                         TRANSP);
557
558             if (Cflag) {
559                 f_print(fout, "(\n");
560                 f_print(fout, "    ");
561                 /* arg name */
562                 if (proc->arg_num > 1)
563                     f_print(fout, "\t%s *argp,\n",
564                           proc->args.argname);
565             }
566
567             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
568             f_print(fout, "    register SVCXPRT *%s)\n",
569                         TRANSP);
570
571             if (Cflag) {
572                 f_print(fout, "(\n");
573                 f_print(fout, "    ");
574                 /* arg name */
575                 if (proc->arg_num > 1)
576                     f_print(fout, "\t%s *argp,\n",
577                           proc->args.argname);
578             }
579
580             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
581             f_print(fout, "    register SVCXPRT *%s)\n",
582                         TRANSP);
583
584             if (Cflag) {
585                 f_print(fout, "(\n");
586                 f_print(fout, "    ");
587                 /* arg name */
588                 if (proc->arg_num > 1)
589                     f_print(fout, "\t%s *argp,\n",
590                           proc->args.argname);
591             }
592
593             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
594             f_print(fout, "    register SVCXPRT *%s)\n",
595                         TRANSP);
596
597             if (Cflag) {
598                 f_print(fout, "(\n");
599                 f_print(fout, "    ");
600                 /* arg name */
601                 if (proc->arg_num > 1)
602                     f_print(fout, "\t%s *argp,\n",
603                           proc->args.argname);
604             }
605
606             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
607             f_print(fout, "    register SVCXPRT *%s)\n",
608                         TRANSP);
609
610             if (Cflag) {
611                 f_print(fout, "(\n");
612                 f_print(fout, "    ");
613                 /* arg name */
614                 if (proc->arg_num > 1)
615                     f_print(fout, "\t%s *argp,\n",
616                           proc->args.argname);
617             }
618
619             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
620             f_print(fout, "    register SVCXPRT *%s)\n",
621                         TRANSP);
622
623             if (Cflag) {
624                 f_print(fout, "(\n");
625                 f_print(fout, "    ");
626                 /* arg name */
627                 if (proc->arg_num > 1)
628                     f_print(fout, "\t%s *argp,\n",
629                           proc->args.argname);
630             }
631
632             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
633             f_print(fout, "    register SVCXPRT *%s)\n",
634                         TRANSP);
635
636             if (Cflag) {
637                 f_print(fout, "(\n");
638                 f_print(fout, "    ");
639                 /* arg name */
640                 if (proc->arg_num > 1)
641                     f_print(fout, "\t%s *argp,\n",
642                           proc->args.argname);
643             }
644
645             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
646             f_print(fout, "    register SVCXPRT *%s)\n",
647                         TRANSP);
648
649             if (Cflag) {
650                 f_print(fout, "(\n");
651                 f_print(fout, "    ");
652                 /* arg name */
653                 if (proc->arg_num > 1)
654                     f_print(fout, "\t%s *argp,\n",
655                           proc->args.argname);
656             }
657
658             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
659             f_print(fout, "    register SVCXPRT *%s)\n",
660                         TRANSP);
661
662             if (Cflag) {
663                 f_print(fout, "(\n");
664                 f_print(fout, "    ");
665                 /* arg name */
666                 if (proc->arg_num > 1)
667                     f_print(fout, "\t%s *argp,\n",
668                           proc->args.argname);
669             }
670
671             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
672             f_print(fout, "    register SVCXPRT *%s)\n",
673                         TRANSP);
674
675             if (Cflag) {
676                 f_print(fout, "(\n");
677                 f_print(fout, "    ");
678                 /* arg name */
679                 if (proc->arg_num > 1)
680                     f_print(fout, "\t%s *argp,\n",
681                           proc->args.argname);
682             }
683
684             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
685             f_print(fout, "    register SVCXPRT *%s)\n",
686                         TRANSP);
687
688             if (Cflag) {
689                 f_print(fout, "(\n");
690                 f_print(fout, "    ");
691                 /* arg name */
692                 if (proc->arg_num > 1)
693                     f_print(fout, "\t%s *argp,\n",
694                           proc->args.argname);
695             }
696
697             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
698             f_print(fout, "    register SVCXPRT *%s)\n",
699                         TRANSP);
700
701             if (Cflag) {
702                 f_print(fout, "(\n");
703                 f_print(fout, "    ");
704                 /* arg name */
705                 if (proc->arg_num > 1)
706                     f_print(fout, "\t%s *argp,\n",
707                           proc->args.argname);
708             }
709
710             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
711             f_print(fout, "    register SVCXPRT *%s)\n",
712                         TRANSP);
713
714             if (Cflag) {
715                 f_print(fout, "(\n");
716                 f_print(fout, "    ");
717                 /* arg name */
718                 if (proc->arg_num > 1)
719                     f_print(fout, "\t%s *argp,\n",
720                           proc->args.argname);
721             }
722
723             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
724             f_print(fout, "    register SVCXPRT *%s)\n",
725                         TRANSP);
726
727             if (Cflag) {
728                 f_print(fout, "(\n");
729                 f_print(fout, "    ");
730                 /* arg name */
731                 if (proc->arg_num > 1)
732                     f_print(fout, "\t%s *argp,\n",
733                           proc->args.argname);
734             }
735
736             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
737             f_print(fout, "    register SVCXPRT *%s)\n",
738                         TRANSP);
739
740             if (Cflag) {
741                 f_print(fout, "(\n");
742                 f_print(fout, "    ");
743                 /* arg name */
744                 if (proc->arg_num > 1)
745                     f_print(fout, "\t%s *argp,\n",
746                           proc->args.argname);
747             }
748
749             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
750             f_print(fout, "    register SVCXPRT *%s)\n",
751                         TRANSP);
752
753             if (Cflag) {
754                 f_print(fout, "(\n");
755                 f_print(fout, "    ");
756                 /* arg name */
757                 if (proc->arg_num > 1)
758                     f_print(fout, "\t%s *argp,\n",
759                           proc->args.argname);
760             }
761
762             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
763             f_print(fout, "    register SVCXPRT *%s)\n",
764                         TRANSP);
765
766             if (Cflag) {
767                 f_print(fout, "(\n");
768                 f_print(fout, "    ");
769                 /* arg name */
770                 if (proc->arg_num > 1)
771                     f_print(fout, "\t%s *argp,\n",
772                           proc->args.argname);
773             }
774
775             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
776             f_print(fout, "    register SVCXPRT *%s)\n",
777                         TRANSP);
778
779             if (Cflag) {
780                 f_print(fout, "(\n");
781                 f_print(fout, "    ");
782                 /* arg name */
783                 if (proc->arg_num > 1)
784                     f_print(fout, "\t%s *argp,\n",
785                           proc->args.argname);
786             }
787
788             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
789             f_print(fout, "    register SVCXPRT *%s)\n",
790                         TRANSP);
791
792             if (Cflag) {
793                 f_print(fout, "(\n");
794                 f_print(fout, "    ");
795                 /* arg name */
796                 if (proc->arg_num > 1)
797                     f_print(fout, "\t%s *argp,\n",
798                           proc->args.argname);
799             }
800
801             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
802             f_print(fout, "    register SVCXPRT *%s)\n",
803                         TRANSP);
804
805             if (Cflag) {
806                 f_print(fout, "(\n");
807                 f_print(fout, "    ");
808                 /* arg name */
809                 if (proc->arg_num > 1)
810                     f_print(fout, "\t%s *argp,\n",
811                           proc->args.argname);
812             }
813
814             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
815             f_print(fout, "    register SVCXPRT *%s)\n",
816                         TRANSP);
817
818             if (Cflag) {
819                 f_print(fout, "(\n");
820                 f_print(fout, "    ");
821                 /* arg name */
822                 if (proc->arg_num > 1)
823                     f_print(fout, "\t%s *argp,\n",
824                           proc->args.argname);
825             }
826
827             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
828             f_print(fout, "    register SVCXPRT *%s)\n",
829                         TRANSP);
830
831             if (Cflag) {
832                 f_print(fout, "(\n");
833                 f_print(fout, "    ");
834                 /* arg name */
835                 if (proc->arg_num > 1)
836                     f_print(fout, "\t%s *argp,\n",
837                           proc->args.argname);
838             }
839
840             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
841             f_print(fout, "    register SVCXPRT *%s)\n",
842                         TRANSP);
843
844             if (Cflag) {
845                 f_print(fout, "(\n");
846                 f_print(fout, "    ");
847                 /* arg name */
848                 if (proc->arg_num > 1)
849                     f_print(fout, "\t%s *argp,\n",
850                           proc->args.argname);
851             }
852
853             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
854             f_print(fout, "    register SVCXPRT *%s)\n",
855                         TRANSP);
856
857             if (Cflag) {
858                 f_print(fout, "(\n");
859                 f_print(fout, "    ");
860                 /* arg name */
861                 if (proc->arg_num > 1)
862                     f_print(fout, "\t%s *argp,\n",
863                           proc->args.argname);
864             }
865
866             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
867             f_print(fout, "    register SVCXPRT *%s)\n",
868                         TRANSP);
869
870             if (Cflag) {
871                 f_print(fout, "(\n");
872                 f_print(fout, "    ");
873                 /* arg name */
874                 if (proc->arg_num > 1)
875                     f_print(fout, "\t%s *argp,\n",
876                           proc->args.argname);
877             }
878
879             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
880             f_print(fout, "    register SVCXPRT *%s)\n",
881                         TRANSP);
882
883             if (Cflag) {
884                 f_print(fout, "(\n");
885                 f_print(fout, "    ");
886                 /* arg name */
887                 if (proc->arg_num > 1)
888                     f_print(fout, "\t%s *argp,\n",
889                           proc->args.argname);
890             }
891
892             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
893             f_print(fout, "    register SVCXPRT *%s)\n",
894                         TRANSP);
895
896             if (Cflag) {
897                 f_print(fout, "(\n");
898                 f_print(fout, "    ");
899                 /* arg name */
900                 if (proc->arg_num > 1)
901                     f_print(fout, "\t%s *argp,\n",
902                           proc->args.argname);
903             }
904
905             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
906             f_print(fout, "    register SVCXPRT *%s)\n",
907                         TRANSP);
908
909             if (Cflag) {
910                 f_print(fout, "(\n");
911                 f_print(fout, "    ");
912                 /* arg name */
913                 if (proc->arg_num > 1)
914                     f_print(fout, "\t%s *argp,\n",
915                           proc->args.argname);
916             }
917
918             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
919             f_print(fout, "    register SVCXPRT *%s)\n",
920                         TRANSP);
921
922             if (Cflag) {
923                 f_print(fout, "(\n");
924                 f_print(fout, "    ");
925                 /* arg name */
926                 if (proc->arg_num > 1)
927                     f_print(fout, "\t%s *argp,\n",
928                           proc->args.argname);
929             }
930
931             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
932             f_print(fout, "    register SVCXPRT *%s)\n",
933                         TRANSP);
934
935             if (Cflag) {
936                 f_print(fout, "(\n");
937                 f_print(fout, "    ");
938                 /* arg name */
939                 if (proc->arg_num > 1)
940                     f_print(fout, "\t%s *argp,\n",
941                           proc->args.argname);
942             }
943
944             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
945             f_print(fout, "    register SVCXPRT *%s)\n",
946                         TRANSP);
947
948             if (Cflag) {
949                 f_print(fout, "(\n");
950                 f_print(fout, "    ");
951                 /* arg name */
952                 if (proc->arg_num > 1)
953                     f_print(fout, "\t%s *argp,\n",
954                           proc->args.argname);
955             }
956
957             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
958             f_print(fout, "    register SVCXPRT *%s)\n",
959                         TRANSP);
960
961             if (Cflag) {
962                 f_print(fout, "(\n");
963                 f_print(fout, "    ");
964                 /* arg name */
965                 if (proc->arg_num > 1)
966                     f_print(fout, "\t%s *argp,\n",
967                           proc->args.argname);
968             }
969
970             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
971             f_print(fout, "    register SVCXPRT *%s)\n",
972                         TRANSP);
973
974             if (Cflag) {
975                 f_print(fout, "(\n");
976                 f_print(fout, "    ");
977                 /* arg name */
978                 if (proc->arg_num > 1)
979                     f_print(fout, "\t%s *argp,\n",
980                           proc->args.argname);
981             }
982
983             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
984             f_print(fout, "    register SVCXPRT *%s)\n",
985                         TRANSP);
986
987             if (Cflag) {
988                 f_print(fout, "(\n");
989                 f_print(fout, "    ");
990                 /* arg name */
991                 if (proc->arg_num > 1)
992                     f_print(fout, "\t%s *argp,\n",
993                           proc->args.argname);
994             }
995
996             f_print(fout, "    struct svc_req *%s)\n", RQSTP);
997             f_print(fout, "    register SVCXPRT *%s)\n",
998                         TRANSP);
999
1000            if (Cflag) {
1001                f_print(fout, "(\n");
1002                f_print(fout, "    ");
1003                /* arg name */
1004                if (proc->arg_num > 1)
1005                    f_print(fout, "\t%s *argp,\n",
1006                          proc->args.argname);
1007            }
1008
1009            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1010            f_print(fout, "    register SVCXPRT *%s)\n",
1011                          TRANSP);
1012
1013            if (Cflag) {
1014                f_print(fout, "(\n");
1015                f_print(fout, "    ");
1016                /* arg name */
1017                if (proc->arg_num > 1)
1018                    f_print(fout, "\t%s *argp,\n",
1019                          proc->args.argname);
1020            }
1021
1022            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1023            f_print(fout, "    register SVCXPRT *%s)\n",
1024                          TRANSP);
1025
1026            if (Cflag) {
1027                f_print(fout, "(\n");
1028                f_print(fout, "    ");
1029                /* arg name */
1030                if (proc->arg_num > 1)
1031                    f_print(fout, "\t%s *argp,\n",
1032                          proc->args.argname);
1033            }
1034
1035            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1036            f_print(fout, "    register SVCXPRT *%s)\n",
1037                          TRANSP);
1038
1039            if (Cflag) {
1040                f_print(fout, "(\n");
1041                f_print(fout, "    ");
1042                /* arg name */
1043                if (proc->arg_num > 1)
1044                    f_print(fout, "\t%s *argp,\n",
1045                          proc->args.argname);
1046            }
1047
1048            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1049            f_print(fout, "    register SVCXPRT *%s)\n",
1050                          TRANSP);
1051
1052            if (Cflag) {
1053                f_print(fout, "(\n");
1054                f_print(fout, "    ");
1055                /* arg name */
1056                if (proc->arg_num > 1)
1057                    f_print(fout, "\t%s *argp,\n",
1058                          proc->args.argname);
1059            }
1060
1061            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1062            f_print(fout, "    register SVCXPRT *%s)\n",
1063                          TRANSP);
1064
1065            if (Cflag) {
1066                f_print(fout, "(\n");
1067                f_print(fout, "    ");
1068                /* arg name */
1069                if (proc->arg_num > 1)
1070                    f_print(fout, "\t%s *argp,\n",
1071                          proc->args.argname);
1072            }
1073
1074            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1075            f_print(fout, "    register SVCXPRT *%s)\n",
1076                          TRANSP);
1077
1078            if (Cflag) {
1079                f_print(fout, "(\n");
1080                f_print(fout, "    ");
1081                /* arg name */
1082                if (proc->arg_num > 1)
1083                    f_print(fout, "\t%s *argp,\n",
1084                          proc->args.argname);
1085            }
1086
1087            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1088            f_print(fout, "    register SVCXPRT *%s)\n",
1089                          TRANSP);
1090
1091            if (Cflag) {
1092                f_print(fout, "(\n");
1093                f_print(fout, "    ");
1094                /* arg name */
1095                if (proc->arg_num > 1)
1096                    f_print(fout, "\t%s *argp,\n",
1097                          proc->args.argname);
1098            }
1099
1100            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1101            f_print(fout, "    register SVCXPRT *%s)\n",
1102                          TRANSP);
1103
1104            if (Cflag) {
1105                f_print(fout, "(\n");
1106                f_print(fout, "    ");
1107                /* arg name */
1108                if (proc->arg_num > 1)
1109                    f_print(fout, "\t%s *argp,\n",
1110                          proc->args.argname);
1111            }
1112
1113            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1114            f_print(fout, "    register SVCXPRT *%s)\n",
1115                          TRANSP);
1116
1117            if (Cflag) {
1118                f_print(fout, "(\n");
1119                f_print(fout, "    ");
1120                /* arg name */
1121                if (proc->arg_num > 1)
1122                    f_print(fout, "\t%s *argp,\n",
1123                          proc->args.argname);
1124            }
1125
1126            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1127            f_print(fout, "    register SVCXPRT *%s)\n",
1128                          TRANSP);
1129
1130            if (Cflag) {
1131                f_print(fout, "(\n");
1132                f_print(fout, "    ");
1133                /* arg name */
1134                if (proc->arg_num > 1)
1135                    f_print(fout, "\t%s *argp,\n",
1136                          proc->args.argname);
1137            }
1138
1139            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1140            f_print(fout, "    register SVCXPRT *%s)\n",
1141                          TRANSP);
1142
1143            if (Cflag) {
1144                f_print(fout, "(\n");
1145                f_print(fout, "    ");
1146                /* arg name */
1147                if (proc->arg_num > 1)
1148                    f_print(fout, "\t%s *argp,\n",
1149                          proc->args.argname);
1150            }
1151
1152            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1153            f_print(fout, "    register SVCXPRT *%s)\n",
1154                          TRANSP);
1155
1156            if (Cflag) {
1157                f_print(fout, "(\n");
1158                f_print(fout, "    ");
1159                /* arg name */
1160                if (proc->arg_num > 1)
1161                    f_print(fout, "\t%s *argp,\n",
1162                          proc->args.argname);
1163            }
1164
1165            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1166            f_print(fout, "    register SVCXPRT *%s)\n",
1167                          TRANSP);
1168
1169            if (Cflag) {
1170                f_print(fout, "(\n");
1171                f_print(fout, "    ");
1172                /* arg name */
1173                if (proc->arg_num > 1)
1174                    f_print(fout, "\t%s *argp,\n",
1175                          proc->args.argname);
1176            }
1177
1178            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1179            f_print(fout, "    register SVCXPRT *%s)\n",
1180                          TRANSP);
1181
1182            if (Cflag) {
1183                f_print(fout, "(\n");
1184                f_print(fout, "    ");
1185                /* arg name */
1186                if (proc->arg_num > 1)
1187                    f_print(fout, "\t%s *argp,\n",
1188                          proc->args.argname);
1189            }
1190
1191            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1192            f_print(fout, "    register SVCXPRT *%s)\n",
1193                          TRANSP);
1194
1195            if (Cflag) {
1196                f_print(fout, "(\n");
1197                f_print(fout, "    ");
1198                /* arg name */
1199                if (proc->arg_num > 1)
1200                    f_print(fout, "\t%s *argp,\n",
1201                          proc->args.argname);
1202            }
1203
1204            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1205            f_print(fout, "    register SVCXPRT *%s)\n",
1206                          TRANSP);
1207
1208            if (Cflag) {
1209                f_print(fout, "(\n");
1210                f_print(fout, "    ");
1211                /* arg name */
1212                if (proc->arg_num > 1)
1213                    f_print(fout, "\t%s *argp,\n",
1214                          proc->args.argname);
1215            }
1216
1217            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1218            f_print(fout, "    register SVCXPRT *%s)\n",
1219                          TRANSP);
1220
1221            if (Cflag) {
1222                f_print(fout, "(\n");
1223                f_print(fout, "    ");
1224                /* arg name */
1225                if (proc->arg_num > 1)
1226                    f_print(fout, "\t%s *argp,\n",
1227                          proc->args.argname);
1228            }
1229
1230            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1231            f_print(fout, "    register SVCXPRT *%s)\n",
1232                          TRANSP);
1233
1234            if (Cflag) {
1235                f_print(fout, "(\n");
1236                f_print(fout, "    ");
1237                /* arg name */
1238                if (proc->arg_num > 1)
1239                    f_print(fout, "\t%s *argp,\n",
1240                          proc->args.argname);
1241            }
1242
1243            f_print(fout, "    struct svc_req *%s)\n", RQSTP);
1244            f_print(fout, "    register SVCXPRT *%s)\n",
1245                          TRANSP);
1246
1247            if (Cflag) {
1248                f_print(fout, "(\n");
1249                f_print(fout, "    ");
1
```

new/usr/src/cmd/rpcgen/rpc\_svcout.c

5

```

431     for (proc = vp->procs; proc != NULL; proc = proc->next) {
432         if (proc->arg_num < 2) { /* single argument */
433             if (streq(proc->args.decls->decl.type,
434                     "void")) {
435                 continue;
436             }
437             filled = 1;
438             f_print(fout, "\t\t");
439             ptype(proc->args.decls->decl.prefix,
440                   proc->args.decls->decl.type, 0);
441             p pname(proc->proc_name, vp->vers_num);
442             f_print(fout, "_arg;\n");
443
444         } else {
445             filled = 1;
446             f_print(fout, "\t\t%s", proc->args.argname);
447             f_print(fout, " ");
448             p pname(proc->proc_name, vp->vers_num);
449             f_print(fout, "_arg;\n");
450         }
451     }
452     if (!filled) {
453         f_print(fout, "\t\tint fill;\n");
454     }
455     f_print(fout, "\t\t%s;\n", ARG);
456
457     if (mtfflag) {
458         filled = 0;
459         f_print(fout, "\tunion {\n");
460         for (proc = vp->procs; proc != NULL;
461               proc = proc->next) {
462             if (streq(proc->res_type, "void") ||
463                 streq(proc->res_type, "oneway"))
464                 continue;
465             filled = 1;
466             f_print(fout, "\t\t");
467             ptype(proc->res_prefix, proc->res_type, 0);
468             p pname(proc->proc_name, vp->vers_num);
469             f_print(fout, "_res;\n");
470         }
471         if (!filled)
472             f_print(fout, "\t\tint fill;\n");
473         f_print(fout, "\t\t%s;\n", RESULT);
474         f_print(fout, "\tbool_t %s;\n", RETVAL);
475
476     } else
477         f_print(fout, "\tchar *%s;\n", RESULT);
478
479     f_print(fout, "\txdrproc_t _xdr_%s, _xdr_%s;\n", ARG, RESULT);
480     if (Cflag) {
481         f_print(fout, "\txdrproc_t _xdr_%s, _xdr_%s;\n",
482                 ARG, RESULT);
483     }
484     if (mtfflag)
485         f_print(fout, "\tbool_t "
486                 "(*%s)(char *, void *, struct svc_req *);\n",
487                 ROUTINE);
488     else
489         f_print(fout, "\tchar *(*%s)"
490                 "(char *, struct svc_req *);\n",
491                 ROUTINE);
492
493     } else {
494         f print(fout,

```

new/usr/src/cmd/rpcgen/rpc\_svcout.c

```

592         }
593     }
594     f_print(fout, "\n\t\t");
595     if (newstyle) { /* new style: calls internal routine */
596         f_print(fout, "_");
597     }
598     if (!newstyle)
599     if ((Cflag || mtflag) && !newstyle)
600         pfname_svc(proc->proc_name, vp->vers_num);
601     else
602         pfname(proc->proc_name, vp->vers_num);
603     f_print(fout, ";\n");
604     f_print(fout, "\t\tbreak;\n\n");
605 }
606 f_print(fout, "\tdefault:\n");
607 printerr("noproc", TRANSP);
608 print_return("\t\t");
609 f_print(fout, "\t\t");
610
611 f_print(fout,
612     "\t(void) memset((char *)&s, 0, sizeof (%s));\n",
613     ARG, ARG);
614 printf("getargs", TRANSP, "(caddr_t)&", ARG);
615 printerr("decode", TRANSP);
616 print_return("\t\t");
617 f_print(fout, "\t\t");
618
619 if (!mtflag)
620     if (Cflag)
621         f_print(fout,
622             "\t%s = (*s)((char *)&s, %s);\n",
623             RESULT, ROUTINE, ARG, RQSTP);
624     else
625         f_print(fout, "\t%s = (%s)(%s, %s);\n",
626             RESULT, ROUTINE, ARG, RQSTP);
627
628     else
629         if (Cflag)
630             f_print(fout,
631                 "\t%s = (bool_t)(%s)"
632                 "((char *)&s, (void *)&s, %s);\n",
633                 RETVAL, ROUTINE, ARG, RESULT, RQSTP);
634
635     "\t%s = (bool_t)(%s)(%s, %s, %s);\n",
636                 RETVAL, ROUTINE, ARG, RESULT, RQSTP);
637
638
639     if (mtflag)
640         f_print(fout,
641             "\tif (_xdr_%s && %s > 0 &&\n"
642             "!svc_sendreply(%s, _xdr_%s, (char *)&s)) {\n";
643             RESULT, RETVAL, TRANSP, RESULT, RESULT);
644
645     else
646         f_print(fout,
647             "\tif (_xdr_%s && %s != NULL &&\n"
648             "!svc_sendreply(%s, _xdr_%s, %s)) {\n";
649             RESULT, RETVAL, TRANSP, RESULT, RESULT);
650
651     printerr("systemerr", TRANSP);
652     f_print(fout, "\t\t");
653
654     printf("freeargs", TRANSP, "(caddr_t)&", ARG);

```

```

587     print_err_message("\t\t", "unable to free arguments");
588     f_print(fout, "\t\texit(1);\n");
589     f_print(fout, "\t\t");
590     /* print out free routine */
591     if (mtflag) {
592         f_print(fout, "\tif (_xdr_%s != NULL) {\n", RESULT);
593         f_print(fout, "\t\tif (!");
594
595         pfname(def->def_name, vp->vers_num);
596         f_print(fout, "\t\t_freeresult(%s, _xdr_%s,\n",
597                 TRANSP, RESULT);
598         f_print(fout, "\t\t\t(caddr_t)&s))\n",
599                 RESULT);
600         print_err_message("\t\t\t", "unable to free results");
601         f_print(fout, "\n");
602         f_print(fout, "\t\t");
603     };
604     print_return("\t\t");
605     f_print(fout, "\t\t");
606 }
607 } unchanged portion omitted
608
609 /*
610 * Write the RPC_MSGOUT function
611 *
612 * Note that while we define RPC_MSGOUT to be printf-like, all existing
613 * calls are of the form "%s", "<msg>" and this implementation assumes that
614 * trivial case. If in the future it's desirable to generate richer calls
615 * this implementation can change to match. This way we don't (yet) have
616 * to introduce varargs into the generated code.
617 */
618 static void
619 write_msg_out(void)
620 {
621     f_print(fout, "\n");
622     f_print(fout, "#if !defined(RPC_MSGOUT)\n");
623     if (!Cflag) {
624         f_print(fout, "extern void RPC_MSGOUT();\n");
625     } else {
626         f_print(fout, "extern void RPC_MSGOUT(const char *, ...);\n");
627     }
628     f_print(fout, "#else\t/* defined(RPC_MSGOUT) */\n");
629     f_print(fout, "static ");
630     if (!Cflag) {
631         f_print(fout, "void\nRPC_MSGOUT(fmt, msg)\n");
632         f_print(fout, "\tchar *fmt;\n");
633         f_print(fout, "\tchar *msg;\n");
634     } else {
635         f_print(fout, "void\nRPC_MSGOUT(const char *fmt, char *msg)\n");
636     }
637     f_print(fout, "\n");
638     f_print(fout, "#ifdef RPC_SVC_FG\n");
639     if (inetdflag || pmflag) {
640         f_print(fout, "\tif (.rpccmstart)\n");
641         f_print(fout, "\ttsyslog(LOG_ERR, fmt, msg);\n");
642         f_print(fout, "\telse \n");
643         f_print(fout, "\t\tprintf(stderr, fmt, msg);\n");
644         f_print(fout, "\t\tputc('\n', stderr);\n");
645         f_print(fout, "\t\t");
646         f_print(fout, "#else\n");
647         f_print(fout, "\ttsyslog(LOG_ERR, fmt, msg);\n");
648         f_print(fout, "#endif\n");
649         f_print(fout, "\t\t");
650     }
651     f_print(fout, "#endif\t/* defined(RPC_MSGOUT) */\n");
652 }

```

```

768 /*
769  * Write the timeout function
770  */
771 static void
772 write_timeout_func(void)
773 {
774     if (!timerflag)
775         return;
776
777     f_print(fout, "\n");
778     if (mtflag) {
779         f_print(fout, /*ARGSUSED*/"\n");
780         f_print(fout, "static void *\n");
781         if (!Cflag) {
782             f_print(fout, "closedown(arg)\n");
783             f_print(fout, "\tvvoid *arg;\n");
784         } else
785             f_print(fout, "closedown(void *arg)\n");
786         f_print(fout, "{\n");
787         f_print(fout, "\t/*CONSTCOND*/\n");
788         f_print(fout, "\twhile (1) {\n");
789         f_print(fout, "\t\t(void) sleep(_RPCSVC_CLOSEDOWN/2);\n\n");
790         f_print(fout,
791             "\t\tif (mutex_trylock(&_svctype_lock) != 0)\n");
792             f_print(fout, "\t\t\tcontinue;\n\n");
793         f_print(fout,
794             "\t\tif (_rpcsvcstate == _IDLE & _rpcsvccount == 0) {\n");
795             if (tirpcflag) {
796                 f_print(fout, "\t\t\tint size;\n");
797             } else {
798                 f_print(fout, "\t\t\textern fd_set svc_fdset;\n");
799                 f_print(fout, "\t\t\tint size;\n");
800             }
801             f_print(fout, "\t\t\tint i, openfd = 0;\n\n");
802             if (tirpcflag) {
803                 f_print(fout, "\t\t\tsize = svc_max_pollfd;\n");
804             } else {
805                 f_print(fout, "\t\t\tif (size == 0) {\n");
806                 f_print(fout, "\t\t\t\tsize = getdtablesize();\n");
807                 f_print(fout, "\t\t\t}\n");
808             }
809             f_print(fout,
810                 "\t\t\tfor (i = 0; i < size && openfd < 2; i++)\n");
811             if (tirpcflag) {
812                 f_print(fout, "\t\t\t\tif (svc_pollfd[i].fd >= 0)\n");
813             } else {
814                 f_print(fout, "\t\t\t\tif (FD_ISSET(i, &svc_fdset))\n");
815             }
816             f_print(fout, "\t\t\t\topenfd++;\n");
817             f_print(fout, "\t\t\t\tif (openfd <= 1)\n");
818             f_print(fout, "\t\t\t\texit(0);\n");
819             f_print(fout, "\t\t\t} else\n");
820             f_print(fout, "\t\t\t\t_rpcsvcstate = _IDLE;\n\n");
821             f_print(fout, "\t\t\t(void) mutex_unlock(&_svctype_lock);\n");
822             f_print(fout, "\t\t}\n");
823             f_print(fout, "static void\n");
824             if (!Cflag) {
825                 f_print(fout, "closedown(sig)\n");
826                 f_print(fout, "\ttint sig;\n");
827             } else
828                 f_print(fout, "closedown(int sig)\n");

```

```

825     f_print(fout, "{\n");
826     if (_rpcsvcstate == _IDLE & _rpcsvccount == 0) {\n");
827     if (tirpcflag) {
828         f_print(fout, "\t\tint size;\n");
829     } else {
830         f_print(fout, "\t\textern fd_set svc_fdset;\n");
831         f_print(fout, "\t\tint size;\n");
832     }
833     f_print(fout, "\t\tint i, openfd = 0;\n\n");
834     if (tirpcflag) {
835         f_print(fout, "\t\ttsize = svc_max_pollfd;\n");
836     } else {
837         f_print(fout, "\t\tif (size == 0) {\n");
838         f_print(fout, "\t\t\tsize = getdtablesize();\n");
839         f_print(fout, "\t\t}\n");
840     }
841     f_print(fout,
842         "\t\tfor (i = 0; i < size && openfd < 2; i++)\n");
843     if (tirpcflag) {
844         f_print(fout, "\t\t\tif (svc_pollfd[i].fd >= 0)\n");
845     } else {
846         f_print(fout, "\t\t\tif (FD_ISSET(i, &svc_fdset))\n");
847     }
848     f_print(fout, "\t\t\topenfd++;\n");
849     f_print(fout, "\t\t\tif (openfd <= 1)\n");
850     f_print(fout, "\t\t\texit(0);\n");
851     f_print(fout, "\t\t} else\n");
852     f_print(fout, "\t\t\t_rpcsvcstate = _IDLE;\n\n");
853
854     f_print(fout, "\t(void) signal(SIGALRM, (SIG_PF) closedown);\n");
855     f_print(fout, "\t(void) signal(SIGALRM, %s closedown);\n",
856           Cflag ? "(SIG_PF)" : "(void(*)())");
857     f_print(fout, "\t(void) alarm(_RPCSVC_CLOSEDOWN/2);\n");
858     f_print(fout, "}\n");
859
860     /* Write the most of port monitor support
861      */
862 static void
863 write_pm_most(char *infile, int netflag)
864 {
865     list *l;
866     definition *def;
867     version_list *vp;
868
869     f_print(fout, "\t(void) sigset(SIGPIPE, SIG_IGN);\n\n");
870     f_print(fout, "\t/*\n");
871     f_print(fout, "\t * If stdin looks like a TLI endpoint, we assume\n");
872     f_print(fout, "\t * that we were started by a port monitor. If\n");
873     f_print(fout, "\t * t_getstate fails with TBADF, this is not a\n");
874     f_print(fout, "\t * TLI endpoint.\n");
875     f_print(fout, "\t */\n");
876     f_print(fout, "\tif (t_getstate(0) != -1 || t_errno != TBADF) {\n");
877     f_print(fout, "\t\tchar *netid;\n");
878     if (!netflag) { /* Not included by -n option */
879         f_print(fout, "\t\tstruct netconfig *nconf = NULL;\n");
880         f_print(fout, "\t\tTSVXPRT *%s;\n", TRANSP);
881     }
882     if (timerflag)
883         f_print(fout, "\t\tint pmclose;\n");
884
885     /* Not necessary, defined in /usr/include/stdlib
886      */
887     f_print(fout, "\textern char *getenv();\n");
888
889     f_print(fout, "\n");

```

```

889         f_print(fout, "\t\t_rpcpmstart = 1;\n");
890         open_log_file(infile, "\t\t");
891         f_print(fout,
892 "\n\t\tif ((netid = getenv(\"NLSPROVIDER\")) == NULL) {\n";
893
894         if (timerflag) {
895             f_print(fout, "\t\t/* started from inetd */\n");
896             f_print(fout, "\t\t\t_tpmclose = 1;\n");
897         }
898         f_print(fout, "\t\t} else {\n");
899         f_print(fout, "\t\t\tif ((nconf = getnetconfigent(netid)) == NULL)\n");
900         print_err_message("\t\t\t", "cannot get transport info");
901         if (timerflag)
902             f_print(fout,
903                 "\n\t\t\t_tpmclose = (t_getstate(0) != T_DATAFER);\n");
904         f_print(fout, "\t\t}\n");
905         f_print(fout,
906 "\t\tif ((%s = svc_tli_create(0, nconf, NULL, 0, 0)) == NULL) {\n",
907             TRANSP);
908         print_err_message("\t\t\t", "cannot create server handle");
909         f_print(fout, "\t\t\texit(1);\n");
910         f_print(fout, "\t\t}\n");
911         f_print(fout, "\t\tif (nconf)\n");
912         f_print(fout, "\t\t\tfreenetconfigent(nconf);\n");
913         for (l = defined; l != NULL; l = l->next) {
914             def = (definition *) l->val;
915             if (def->def_kind != DEF_PROGRAM) {
916                 continue;
917             }
918             for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
919                 f_print(fout,
920                     "\t\t\tif (!svc_reg(%s, %s, %s,\n",
921                         TRANSP, def->def_name, vp->vers_name);
922                     f_print(fout, "\t\t\t\t");
923                     pfname(def->def_name, vp->vers_num);
924                     f_print(fout, ", 0)) {\n");
925                     print_err_message("\t\t\t",
926                         "unable to register (%s, %s).",
927                         def->def_name, vp->vers_name);
928                     f_print(fout, "\t\t\texit(1);\n");
929                     f_print(fout, "\t\t}\n");
930             }
931         }
932         if (timerflag) {
933             f_print(fout, "\t\tif (pmclose) {\n");
934                 if (mtflag)
935                     f_print(fout,
936 "\t\t\tif (thr_create(NULL, 0, closedown, NULL,\n\t\t\t\t0, NULL) != 0) {\n");
937                     print_err_message("\t\t\t",
938                         "cannot create closedown thread");
939                     f_print(fout, "\t\t\texit(1);\n");
940                 } else {
941                     f_print(fout,
942
943 "\t\t\t\t(void) signal(SIGALRM, (SIG_PF) closedown),\n");
944         "\t\t\t\t(void) signal(SIGALRM, %s closedown);\n",
945             cflag? "(SIG_PF)" : "(void(*)())");
946             f_print(fout,
947 "\t\t\talarm(_RPCSVC_CLOSEDOWN/2);\n");
948         }
949         f_print(fout, "\t\t}\n");
950         f_print(fout, "\t\ttsvc_run();\n");
951         f_print(fout, "\t\t/* NOTREACHED */\n");
952         f_print(fout, "\t}\n");

```

```

953 }
unchanged portion omitted

```

new/usr/src/cmd/rpcgen/rpc\_tblout.c

```

***** 4924 Sun Aug 3 11:09:17 2014 *****
new/usr/src/cmd/rpcgen/rpc_tblout.c
rpcgen should only produce ANSI code
*****
1  /*
2   * CDDL HEADER START
3   *
4   * The contents of this file are subject to the terms of the
5   * Common Development and Distribution License (the "License").
6   * You may not use this file except in compliance with the License.
7   *
8   * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9   * or http://www.opensolaris.org/os/licensing.
10  * See the License for the specific language governing permissions
11  * and limitations under the License.
12  *
13  * When distributing Covered Code, include this CDDL HEADER in each
14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15  * If applicable, add the following below this CDDL HEADER, with the
16  * fields enclosed by brackets "[]" replaced with your own identifying
17  * information: Portions Copyright [yyyy] [name of copyright owner]
18  *
19  * CDDL HEADER END
20  */
21  /*
22   * Copyright 2014 Garrett D'Amore <garrett@damore.org>
23   *
24   * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
25   * Use is subject to license terms.
26   */
27  /*
28  /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
29  /* All Rights Reserved */
30  /*
31  * University Copyright- Copyright (c) 1982, 1986, 1988
32  * The Regents of the University of California
33  * All Rights Reserved
34  *
35  * University Acknowledgment- Portions of this document are derived from
36  * software developed by the University of California, Berkeley, and its
37  * contributors.
38  */
39  /*
40  * rpc_tblout.c, Dispatch table outputter for the RPC protocol compiler
41  */
42  /*
43  #include <stdio.h>
44  #include <string.h>
45  #include "rpc_parse.h"
46  #include "rpc_util.h"
47  */
48  extern int nullproc(proc_list *);
49  static void write_table(definition *);
50  static void printit(char *, char *);
51  static void printit(char *, char *);
52  static void printit(char *, char *);
53  #define TABSIZE 8
54  #define TABCOUNT 5
55  #define TABSTOP (TABSIZE*TABCOUNT)
56  static char tabstr[TABCOUNT+1] = "\t\t\t\t\t\t";
57  static char tbl_hdr[] = "struct rpcgen_table %s_table[] = {\n";
58  static char tbl_end[] = "};\n";

```

1

```
128         }
129         if (tirpcflag)
130             f_print(fout,
131                     "\n\t(void *(*)())RPCGEN_ACTION(\"");
132         else
133             f_print(fout,
134                     "\n\t(char *(*)())RPCGEN_ACTION(\"");

136         /* routine to invoke */
137         if (!newstyle)
138             if (Cflag && !newstyle)
139                 pvname_svc(proc->proc_name, vp->vers_num);
140             else {
141                 if (newstyle) /* calls internal func */
142                     f_print(fout, "_");
143                 pvname(proc->proc_name, vp->vers_num);
144             }
145         f_print(fout, "),\n");

146         /* argument info */
147         if (proc->arg_num > 1)
148             printit(NULL, proc->args.argname);
149         else
150             /* do we have to do something special for newstyle */
151             printit(proc->args.decls->decl.prefix,
152                     proc->args.decls->decl.type);
153             /* result info */
154             printit(proc->res_prefix, proc->res_type);
155         }

156         /* print the table trailer */
157         f_print(fout, tbl_end);
158         f_print(fout, tbl_nproc, progvers, progvers, progvers);
159     }
160 }
```

unchanged portion omitted

new/usr/src/cmd/rpcgen/rpc\_util.h

```
*****  
4722 Sun Aug 3 11:09:17 2014  
new/usr/src/cmd/rpcgen/rpc_util.h  
rpcgen should only produce ANSI code  
*****  
1 /*  
2 * CDDL HEADER START  
3 *  
4 * The contents of this file are subject to the terms of the  
5 * Common Development and Distribution License (the "License").  
6 * You may not use this file except in compliance with the License.  
7 *  
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE  
9 * or http://www.opensolaris.org/os/licensing.  
10 * See the License for the specific language governing permissions  
11 * and limitations under the License.  
12 *  
13 * When distributing Covered Code, include this CDDL HEADER in each  
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.  
15 * If applicable, add the following below this CDDL HEADER, with the  
16 * fields enclosed by brackets "[]" replaced with your own identifying  
17 * information: Portions Copyright [yyyy] [name of copyright owner]  
18 *  
19 * CDDL HEADER END  
20 */  
  
22 /*  
23 * Copyright 2014 Garrett D'Amore <garrett@damore.org>  
24 *  
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.  
26 * Use is subject to license terms.  
27 */  
28 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */  
29 /* All Rights Reserved */  
30 /*  
31 * University Copyright- Copyright (c) 1982, 1986, 1988  
32 * The Regents of the University of California  
33 * All Rights Reserved  
34 *  
35 * University Acknowledgment- Portions of this document are derived from  
36 * software developed by the University of California, Berkeley, and its  
37 * contributors.  
38 */  
  
40 #ifndef _RPC_UTIL_H  
41 #define _RPC_UTIL_H  
  
43 #include <sys/types.h>  
44 #include <stdlib.h>  
45 #include "rpc_scan.h"  
  
47 #ifdef __cplusplus  
48 extern "C" {  
49 #endif  
  
51 /*  
52 * rpc_util.h, Useful definitions for the RPC protocol compiler  
53 */  
  
56 /* Current version number of rpcgen. */  
57 #define RPCGEN_MAJOR 1  
58 #define RPCGEN_MINOR 1  
  
60 #define f_print (void) fprintf
```

1

new/usr/src/cmd/rpcgen/rpc\_util.h

```
62 struct list {  
63     definition *val;  
64     struct list *next;  
65 };  
unchanged_portion_omitted_  
  
90 #define PUT 1  
91 #define GET 2  
  
93 /*  
94 * Global variables  
95 */  
96 #define MAXLINESIZE 1024  
97 extern char curline[MAXLINESIZE];  
98 extern char *where;  
99 extern int linenum;  
  
101 extern char *filename;  
102 extern FILE *fout;  
103 extern FILE *fin;  
  
105 extern list *defined;  
  
107 extern bas_type *typ_list_h;  
108 extern bas_type *typ_list_t;  
109 extern xdfunc *xdrfunc_head, *xdrfunc_tail;  
  
111 /*  
112 * All the option flags  
113 */  
114 extern int inetdflag;  
115 extern int pmflag;  
116 extern int tbiflag;  
117 extern int logflag;  
118 extern int newstyle;  
119 extern int Cflag; /* ANSI-C/C++ flag */  
120 extern int CCflag; /* C++ flag */  
121 extern int tirpcflag; /* flag for generating tirpc code */  
122 extern int inlinemode; /* if this is 0, then do not generate inline code */  
123 extern int mtflag;  
124 extern int mtauto;  
125 extern int rflag;  
  
126 /*  
127 * Other flags related with inetd jumpstart.  
128 */  
129 extern int indefinitewait;  
130 extern int exitnow;  
131 extern int timerflag;  
  
133 extern int nonfatalerrors;  
135 extern pid_t childpid;  
  
137 /*  
138 * rpc_util routines  
139 */  
140 extern void storeval(list **, definition *);  
  
142 #define STOREVAL(list, item) \\\n143     storeval(list, item)  
  
145 extern definition *findval(list *, char *, int (*)());  
  
147 #define FINDVAL(list, item, finder) \\\n148     findval(list, item, finder)
```

2

```
150 extern char *fixtype(char *);
151 extern char *stringfix(char *);
152 extern char *lcase(char *);
153 extern void pfname_svc(char *, char *);
154 extern void pfname(char *, char *);
155 extern void ptype(char *, char *, int);
156 extern int isvectordef(char *, relation);
157 extern int streq(char *, char *);
158 extern void error(char *);
159 extern void expected1(tok_kind);
160 extern void expected2(tok_kind, tok_kind);
161 extern void expected3(tok_kind, tok_kind, tok_kind);
162 extern void tabify(FILE *, int);
163 extern void record_open(char *);
164 extern bas_type *find_type(char *);

166 /*
167  * rpc_cout routines
168 */
169 extern void emit(definition *);

171 /*
172  * rpc_hout routines
173 */
174 extern void print_datadef(definition *);
175 extern void print_funcdef(definition *);
176 extern void print_xdr_func_def(char *, int);
175 extern void print_xdr_func_def(char *, int, int);

178 /*
179  * rpc_svcout routines
180 */
181 extern void write_most(char *, int, int);
182 extern void write_rest(void);
183 extern void write_inetd_register(char *);
184 extern void write_netid_register(char *);
185 extern void write_nettpe_register(char *);

187 /*
188  * rpc_clntout routines
189 */
190 extern void write_stubs(void);

192 /*
193  * rpc_tblout routines
194 */
195 extern void write_tables(void);

197 #ifdef __cplusplus
198 }


---

unchanged_portion_omitted
```

```
new/usr/src/man/man1/rpcgen.1
```

```
*****
12959 Sun Aug 3 11:09:17 2014
new/usr/src/man/man1/rpcgen.1
rpcgen should only produce ANSI code
*****
1 .\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
1 '\"
1 te
2 .\" Copyright (C) 2009, Sun Microsystems, Inc. All Rights Reserved
3 .\" Copyright 1989 AT&T
4 .\" The contents of this file are subject to the terms of the Common Development
5 .\" See the License for the specific language governing permissions and limitat
6 .\" fields enclosed by brackets "[]" replaced with your own identifying informat
7 .Dd "Aug 2, 2014"
8 .Dt RPCGEN 1
9 .Os
10 .Sh NAME
11 .Nm rpcgen
12 .Nd an RPC protocol compiler
13 .Sh SYNOPSIS
14 .Nm
15 .Ar infile
16 .
17 .Nm
18 .Op Fl a
19 .Op Fl A
20 .Op Fl b
21 .Op Fl C
22 .Op Fl D Ar name Ns Op = Ns Ar value
23 .Op Fl i Ar size
24 .Op Fl I Op Fl K Ar seconds
25 .Op Fl L
26 .Op Fl M
27 .Op Fl N
28 .Op Fl T
29 .Op Fl v
30 .Op Fl Y Ar pathname
31 .Ar infile
32 .
33 .Nm
34 .Op Fl c | Fl h | Fl l | Fl m | Fl t | Fl "Sc" | Fl "Ss" | Fl "Sm"
35 .Op Fl o Ar outfile
36 .Op Ar infile
37 .
38 .Nm
39 .Op Fl s Ar nettype
40 .Op Fl o Ar outfile
41 .Op Ar infile
42 .
43 .Nm
44 .Op Fl n Ar netid
45 .Op Fl o Ar outfile
46 .Op infile
47 .
48 .Sh DESCRIPTION
49 The
50 .Nm
51 utility is a tool that generates C code to implement an
52 RPC protocol. The input to
53 .Nm
54 is a language similar to C known
55 as
56 .Em RPC Language
57 (Remote Procedure Call Language).
58 .Lp
59 The
60 .Nm
```

```
1
```

```
new/usr/src/man/man1/rpcgen.1
```

```
61 utility is normally used as in the first synopsis where it
62 takes an input file and generates four output files. If the
63 .Ar infile
64 is
65 named
66 .Pa proto.x ,
67 then
68 .Nm
69 generates a header in
70 .Pa proto.h ,
71 XDR routines in
72 .Pa proto_xdr.c ,
73 server-side stubs in
74 .Pa proto_svc.c ,
75 and client-side stubs in
76 .Pa proto_clnt.c .
77 With the
78 .Fl T
79 option, it also generates the RPC dispatch table in
80 .Pa proto_tbl.i .
81 .Lp
82 .Nm
83 can also generate sample client and server files that can be
84 customized to suit a particular application. The
85 .Fl "Sc" ,
86 .Fl "Ss" ,
87 and
88 .Fl "Sm"
89 options generate sample client, server and makefile, respectively.
90 The
91 .Fl a
92 option generates all files, including sample files. If the infile
93 is
94 .Pa proto.x ,
95 then the client side sample file is written to
96 .Pa proto_client.c ,
97 the server side sample file to
98 .Pa proto_server.c
99 and the sample makefile to
100 .Pa makefile.proto .
101 .Lp
7 .TH RPCGEN 1 "Dec 16, 2013"
8 .SH NAME
9 rpcgen \- an RPC protocol compiler
10 .SH SYNOPSIS
11 .LP
12 .nf
13 \fBrpcgen\fR \fIinfile\fR
14 .fi
16 .LP
17 .nf
18 \fBrpcgen\fR [\fB-a\fR] [\fB-A\fR] [\fB-b\fR] [\fB-C\fR] [\fB-D\fR \fIname\fR [=
19 [\fB-I\fR [\fB-K\fR \fIseconds\fR]] [\fB-L\fR] [\fB-M\fR] [\fB-N\fR] [\fB-
20 [\fB-Y\fR \fIpathname\fR] \fIinfile\fR
21 .fi
23 .LP
24 .nf
25 \fBrpcgen\fR [\fB-c\fR / \fB-h\fR / \fB-l\fR / \fB-m\fR / \fB-t\fR / \fB-Sc\fR /
26 [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
27 .fi
29 .LP
30 .nf
31 \fBrpcgen\fR [\fB-s\fR \fInettype\fR] [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
```

```
2
```

```

32 .fi
34 .LP
35 .nf
36 \fBrpcgen\fR [\fB-n\fR \fInetid\fR] [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
37 .fi

39 .SH DESCRIPTION
40 .sp
41 .LP
42 The \fBrpcgen\fR utility is a tool that generates C code to implement an
43 \fBRPC\fR protocol. The input to \fBrpcgen\fR is a language similar to C known
44 as \fBRPC\fR Language (Remote Procedure Call Language).
45 .sp
46 .LP
47 The \fBrpcgen\fR utility is normally used as in the first synopsis where it
48 takes an input file and generates four output files. If the \fIinfile\fR is
49 named \fBproto.x\fR, then \fBrpcgen\fR generates a header in \fBproto.h\fR,
50 \fBXDR\fR routines in \fBproto_xdr.c\fR, server-side stubs in
51 \fBproto_svc.c\fR, and client-side stubs in \fBproto_clnt.c\fR. With the
52 \fB-T\fR option, it also generates the \fBRPC\fR dispatch table in
53 \fBproto_tbl.i\fR.
54 .sp
55 .LP
56 \fBrpcgen\fR can also generate sample client and server files that can be
57 customized to suit a particular application. The \fB-Sc\fR, \fB-Ss\fR, and
58 \fB-Sm\fR options generate sample client, server and makefile, respectively.
59 The \fB-a\fR option generates all files, including sample files. If the infile
60 is \fBproto.x\fR, then the client side sample file is written to
61 \fBproto_client.c\fR, the server side sample file to \fBproto_server.c\fR and
62 the sample makefile to \fBmakefile.proto\fR.
63 .sp
64 .LP
102 The server created can be started both by the port monitors (for example,
103 .Xr ineted 1M
104 or
105 .Xr listen 1M )
106 or by itself. When it is started by a port
66 \fBinetd\fR or \fBlisten\fR) or by itself. When it is started by a port
107 monitor, it creates servers only for the transport for which the file
108 descriptor 0 was passed. The name of the transport must be specified by
109 setting up the environment variable
110 .Ev PM_TRANSPORT .
111 When the server
112 generated by
113 .Nm
114 is executed, it creates server handles for all the
115 transports specified in the
116 .Ev NETPATH
117 environment variable, or if it is
68 descriptor \fBO\fR was passed. The name of the transport must be specified by
69 setting up the environment variable \fBPM_TRANSPORT\fR. When the server
70 generated by \fBrpcgen\fR is executed, it creates server handles for all the
71 transports specified in the \fBNETPATH\fR environment variable, or if it is
72 unset, it creates server handles for all the visible transports from the
118 .Pa /etc/netconfig
120 file. Note: the transports are chosen at run time and not
73 \fB/etc/netconfig\fR file. Note: the transports are chosen at run time and not
121 at compile time. When the server is self-started, it backgrounds itself by
122 default. A special define symbol
123 .Dv RPC_SVC_FG
124 can be used to run the server process in foreground.
125 .Lp
75 default. A special define symbol \fBRPC_SVC_FG\fR can be used to run the server
76 process in foreground.
77 .sp

```

```

78 .LP
126 The second synopsis provides special features which allow for the creation of
127 more sophisticated RPC servers. These features include support for
128 user-provided
129 .Li #defines
130 and RPC dispatch tables. The entries in the
131 RPC dispatch table contain:
132 .Bl -bullett -offset indent
133 .It
80 more sophisticated \fBRPC\fR servers. These features include support for
81 user-provided \fB#defines\fR and \fBRPC\fR dispatch tables. The entries in the
82 \fBRPC\fR dispatch table contain:
83 .RS +4
84 .TP
85 .ie t \(\bu
86 .el o
134 pointers to the service routine corresponding to that procedure
135 .It
88 .RE
89 .RS +4
90 .TP
91 .ie t \(\bu
92 .el o
136 a pointer to the input and output arguments
137 .It
94 .RE
95 .RS +4
96 .TP
97 .ie t \(\bu
98 .el o
138 the size of these routines
139 .El
140 .Lp
100 .RE
101 .sp
102 .Lp
141 A server can use the dispatch table to check authorization and then to execute
142 the service routine. A client library can use the dispatch table to deal with
143 the details of storage management and XDR data conversion.
144 .Lp
105 the details of storage management and \fBXDR\fR data conversion.
106 .sp
107 .LP
145 The other three synopses shown above are used when one does not want to
146 generate all the output files, but only a particular one. See the
147 .Sx EXAMPLES
148 section below for examples of
149 .Nm
150 usage. When
151 .Nm
152 is executed with the
153 .Fl s
154 option, it creates servers for that particular class of
155 transports. When executed with the
156 .Fl n
157 option, it creates a server for the
158 transport specified by
159 .Ar netid .
160 If
161 .Ar infile
162 is not specified,
163 .Nm
164 accepts the standard input.
165 .Lp
109 generate all the output files, but only a particular one. See the EXAMPLES
110 section below for examples of \fBrpcgen\fR usage. When \fBrpcgen\fR is executed

```

```

111 with the \fB-\fR option, it creates servers for that particular class of
112 transports. When executed with the \fB-n\fR option, it creates a server for the
113 transport specified by \fIinetid\fR. If \fIinfile\fR is not specified,
114 \fBrpcgen\fR accepts the standard input.
115 .sp
116 .LP
166 All the options mentioned in the second synopsis can be used with the other
167 three synopses, but the changes are made only to the specified output file.
168 .LP
169 The C preprocessor
170 .It cc Fl E
171 is run on the input file before it is
172 actually interpreted by
173 .Nm .
174 For each type of output file,
175 .Nm
176 defines a special preprocessor symbol for use by the
177 .Nm
119 .sp
120 .LP
121 The C preprocessor \fBcc\fR \fB-E\fR is run on the input file before it is
122 actually interpreted by \fBrpcgen\fR. For each type of output file,
123 \fBrpcgen\fR defines a special preprocessor symbol for use by the \fBrpcgen\fR
178 programmer:
179 .Bl -tag -width Dv -offset indent
180 .It Dv RPC_HDR
125 .sp
126 .ne 2
127 .na
128 \fB\fBRPC_HDR\fR\fR
129 .ad
130 .RS 12n
181 defined when compiling into headers
182 .It Dv RPC_XDR
183 defined when compiling into XDR routines
184 .It Dv RPC_SVC
132 .RE

134 .sp
135 .ne 2
136 .na
137 \fB\fBRPC_XDR\fR\fR
138 .ad
139 .RS 12n
140 defined when compiling into \fBXDR\fR routines
141 .RE

143 .sp
144 .ne 2
145 .na
146 \fB\fBRPC_SVC\fR\fR
147 .ad
148 .RS 12n
185 defined when compiling into server-side stubs
186 .It Dv RPC_CLNT
150 .RE

152 .sp
153 .ne 2
154 .na
155 \fB\fBRPC_CLNT\fR\fR
156 .ad
157 .RS 12n
187 defined when compiling into client-side stubs
188 .It Dv RPC_TBL
189 defined when compiling into RPC dispatch tables

```

```

190 .El
191 .Lp
192 Any line beginning with
193 .Dq %
194 is passed directly into the output file,
195 uninterpreted by
196 .Nm ,
197 except that the leading
198 .Dq %
199 is stripped
200 off. To specify the path name of the C preprocessor, use the
201 .Fl Y
202 flag.
203 .Lp
204 For every data type referred to in
205 .Ar infile ,
206 .Nm
207 assumes that
208 there exists a routine with the string
209 .Sy xdr_
210 prepended to the name of the
211 data type. If this routine does not exist in the RPC/XDR library,
159 .RE

161 .sp
162 .ne 2
163 .na
164 \fB\fBRPC_TBL\fR\fR
165 .ad
166 .RS 12n
167 defined when compiling into \fBRPC\fR dispatch tables
168 .RE

170 .sp
171 .LP
172 Any line beginning with ''\fB%\fR'' is passed directly into the output file,
173 uninterpreted by \fBrpcgen\fR, except that the leading ''\fB%\fR'' is stripped
174 off. To specify the path name of the C preprocessor, use the \fB-Y\fR flag.
175 .sp
176 .LP
177 For every data type referred to in \fIinfile\fR, \fBrpcgen\fR assumes that
178 there exists a routine with the string \fBxdr_\fR prepended to the name of the
179 data type. If this routine does not exist in the \fBRPC\fR/\fBXDR\fR library,
212 it must be provided. Providing an undefined data type allows customization of
213 XDR routines.
214 .Ss "Server Error Reporting"
215 By default, errors detected by
216 .Pa proto_svc.c
217 is reported to standard error and/or the system log.
218 .Lp
181 \fBXDR\fR routines.
182 .SS "Server Error Reporting"
183 .sp
184 .LP
185 By default, errors detected by \fBproto_svc.c\fR is reported to standard error
186 and/or the system log.
187 .sp
188 .LP
219 This behavior can be overridden by compiling the file with a definition of
220 .Dv RPC_MSGOUT ,
221 for example,
222 .Fl D Dv RPC_MSGOUT Ns = Ns Ar mymsgfunc .
223 The function
190 \fBRPC_MSGOUT\fR, for example, \fB-DRPC_MSGOUT=mymsgfunc\fR. The function
224 specified is called to report errors. It must conform to the following
225 .Xr printf 3C

```

```

226 style signature:
227 .Lp
228 .Dl extern void RPC_MSGOUT(const char *fmt, ...);
229 .Sh OPTIONS
192 \fBprintf\fR-like signature:
193 .sp
194 .in +2
195 .nf
196 extern void RPC_MSGOUT(const char *fmt, ...);
197 .fi
198 .in -2
199 .sp
201 .Sh OPTIONS
202 .sp
203 .LP
230 The following options are supported:
231 .Bl -tag -width Fl
232 .
233 .It Fl a
205 .sp
206 .ne 2
207 .na
208 \fB\fB-a\fR\fR
209 .ad
210 .RS 18n
234 Generates all files, including sample files.
235 .
236 .It Fl A
237 Enables the Automatic
238 MT mode in the server main program. In this mode,
239 the RPC library automatically creates threads to service client requests.
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fB\fB-A\fR\fR
218 .ad
219 .RS 18n
220 Enables the Automatic \fBMT\fR mode in the server main program. In this mode,
221 the \fBRPC\fR library automatically creates threads to service client requests.
240 This option generates multithread-safe stubs by implicitly turning on the
241 .Fl M
242 option. Server multithreading modes and parameters can be set using
243 the
244 .Xr rpc_control 3NSL
245 call.
246 .Nm
247 generated code does not change
248 the default values for the Automatic MT mode.
249 .
250 .It Fl b
251 Backward compatibility mode. Generates transport-specific RPC code for
223 \fB-M\fR option. Server multithreading modes and parameters can be set using
224 the \fBrpc_control\fR(3NSL) call. \fBrpcgen\fR generated code does not change
225 the default values for the Automatic \fBMT\fR mode.
226 .RE

228 .sp
229 .ne 2
230 .na
231 \fB\fB-b\fR\fR
232 .ad
233 .RS 18n
234 Backward compatibility mode. Generates transport-specific \fBRPC\fR code for

```

```

252 older versions of the operating system.
253 .
254 .It Fl c
255 Compiles into XDR routines.
256 .
257 .It Fl C
236 .RE

238 .sp
239 .ne 2
240 .na
241 \fB\fB-C\fR\fR
242 .ad
243 .RS 18n
244 Compiles into \fBXDR\fR routines.
245 .RE

247 .sp
248 .ne 2
249 .na
250 \fB\fB-C\fR\fR
251 .ad
252 .RS 18n
258 Generates header and stub files which can be used with ANSI C compilers.
259 Headers generated with this flag can also be used with C++ programs. This
260 behavior is now default, and therefore this option
261 is redundant. It remains here for compatibility.
262 .It Fl D Ar name Ns Op = Ns Ar value
254 Headers generated with this flag can also be used with C++ programs.
255 .RE

257 .sp
258 .ne 2
259 .na
260 \fB\fB-D\fR\fIname\fR\fB[=\fR\fIvalue\fR\fB]\fR\fR
261 .ad
262 .RS 18n
263 Defines a symbol \fIname\fR. Equivalent to the \fB#define\fR directive in the
264 source. If no \fIvalue\fR is given, \fIvalue\fR is defined as \fB1\fR. This
265 option can be specified more than once.
266 .
267 .It Fl h
268 Compiles into C data-definitions (a header). The
269 .Fl T
270 option can be
271 used in conjunction to produce a header which supports RPC dispatch
266 .RE

268 .sp
269 .ne 2
270 .na
271 \fB\fB-h\fR\fR
272 .ad
273 .RS 18n
274 Compiles into \fBC\fR data-definitions (a header). The \fB-T\fR option can be
275 used in conjunction to produce a header which supports \fBRPC\fR dispatch
272 tables.
273 .
274 .It Fl i Ar size
277 .RE

279 .sp
280 .ne 2
281 .na
282 \fB\fB-i\fR\fIsize\fR\fR
283 .ad

```

```

284 .RS 18n
275 Size at which to start generating inline code. This option is useful for
276 optimization. The default
277 .Ar size
278 is 5.
279 .
280 .It Fl I
281 Compiles support for
282 .Xr inetd 1M
283 in the server side stubs. Such servers can
284 be self-started or can be started by
285 .Xr inetd 3C .
286 When the server is
286 optimization. The default \fIsize\fR is 5.
287 .RE

288 .sp
290 .ne 2
291 .na
292 \fB\fB-I\fR\fR
293 .ad
294 .RS 18n
295 Compiles support for \fBinetd\fR(1M) in the server side stubs. Such servers can
296 be self-started or can be started by \fBinetd\fR. When the server is
297 self-started, it backgrounds itself by default. A special define symbol
298 .Dv RPC_SVC_FG
299 can be used to run the server process in foreground, or the
300 user can simply compile without the
301 .Fl I
302 option.
303 .Lp
304 If there are no pending client requests, the
305 .Xr inetd 1M
306 servers exit after 120
307 seconds (default). The default can be changed with the
308 .Fl -K
309 option. All of
310 the error messages for
311 .Xr inetd 1M
312 servers are always logged with
313 .Xr syslog 3C .
314 .Lp
315 .Em Note:
316 This option is supported for backward compatibility only. It should
317 always be used in conjunction with the
318 .Fl b
319 option which generates backward
320 compatibility code. By default (that is, when
321 .Fl b
322 is not specified),
323 .Nm
324 generates servers that can be invoked through portmonitors.
325 .
326 .It Fl K Ar seconds
327 By default, services created using
328 .Nm
329 and invoked through port
330 \fBRPC_SVC_FG\fR can be used to run the server process in foreground, or the
331 user can simply compile without the \fB-I\fR option.
332 .sp
333 If there are no pending client requests, the \fBinetd\fR servers exit after 120
334 seconds (default). The default can be changed with the \fB-K\fR option. All of
335 the error messages for \fBinetd\fR servers are always logged with
336 .Xr syslog\fR(3C).
337 .sp
338 \fBNote:\fR This option is supported for backward compatibility only. It should

```

```

307 always be used in conjunction with the \fB-b\fR option which generates backward
308 compatibility code. By default (that is, when \fB-b\fR is not specified),
309 \fBrpcgen\fR generates servers that can be invoked through portmonitors.
310 .RE

311 .sp
312 .ne 2
313 .na
314 \fB\fB-K\fR \fIseconds\fR\fR
315 .ad
316 .RS 18n
317 By default, services created using \fBrpcgen\fR and invoked through port
318 monitors wait 120 seconds after servicing a request before exiting. That
319 interval can be changed using the
320 .Fl K
321 flag. To create a server that exits
322 immediately upon servicing a request, use
323 .Fl K Li 0 .
324 To create a server
325 that never exits, the appropriate argument is
326 .Fl K Li \f(mil\fR .
327 .Lp
328 When monitoring for a server, some portmonitors, like
329 .Xr listen 1M ,
330 .Em always
331 spawn a new process in response to a service request. If it is
332 interval can be changed using the \fB-K\fR flag. To create a server that exits
333 immediately upon servicing a request, use \fB-K\fR \fB0\fR. To create a server
334 that never exits, the appropriate argument is \fB-K\fR \fB\f(mil\fR\&.
335 .sp
336 When monitoring for a server, some portmonitors, like \fBlisten\fR(1M),
337 \fBalways\fR spawn a new process in response to a service request. If it is
338 known that a server are used with such a monitor, the server should exit
339 immediately on completion. For such servers,
340 .Nm
341 should be used with
342 .Fl K Li 0 .
343 .
344 .It Fl 1
345 immediately on completion. For such servers, \fBrpcgen\fR should be used with
346 \fB-K\fR \fB0\fR.
347 .RE

348 .sp
349 .ne 2
350 .na
351 \fB\fB-L\fR\fR
352 .ad
353 .RS 18n
354 Compiles into client-side stubs.
355 .
356 .It Fl L
357 When the servers are started in foreground, uses
358 .Xr syslog 3C
359 to log the server errors instead of printing them on the standard error.
360 .
361 .It Fl m
362 .RE

363 .sp
364 .ne 2
365 .na
366 \fB\fB-L\fR\fR
367 .ad
368 .RS 18n
369 When the servers are started in foreground, uses \fBsyslog\fR(3C) to log the

```

```

347 server errors instead of printing them on the standard error.
348 .RE

350 .sp
351 .ne 2
352 .na
353 \fB\fB-m\fR\fR
354 .ad
355 .RS 18n
349 Compiles into server-side stubs, but do not generate a "main" routine. This
350 option is useful for doing callback-routines and for users who need to write
351 their own "main" routine to do initialization.
352 .
353 .It Fl M
359 .RE

361 .sp
362 .ne 2
363 .na
364 \fB\fB-M\fR\fR
365 .ad
366 .RS 18n
354 Generates multithread-safe stubs for passing arguments and results between
355 code generated by
356 .Nm rpcgen
357 and user written code. This option is useful for
358 \fBrpcgen\fR-generated code and user written code. This option is useful for
359 users who want to use threads in their code.
359 .
360 .It Fl N
370 .RE

372 .sp
373 .ne 2
374 .na
375 \fB\fB-N\fR\fR
376 .ad
377 .RS 18n
361 This option allows procedures to have multiple arguments. It also uses the
362 style of parameter passing that closely resembles C. So, when passing an
363 argument to a remote procedure, you do not have to pass a pointer to the
364 argument, but can pass the argument itself. This behavior is different from the
365 old style of code generated by
366 .Nm .
367 To maintain backward compatibility, this option is not the default.
368 .
369 .It Fl n Ar netid
370 Compiles into server-side stubs for the transport specified by
371 Ar netid .
372 There should be an entry for
373 Ar netid
374 in the
375 .Xr netconfig 4
376 database. This
382 old style of \fBrpcgen\fR-generated code. To maintain backward compatibility,
383 this option is not the default.
384 .RE

386 .sp
387 .ne 2
388 .na
389 \fB\fB-n\fR\fIinetid\fR\fR
390 .ad
391 .RS 18n
392 Compiles into server-side stubs for the transport specified by \fIinetid\fR.
393 There should be an entry for \fIinetid\fR in the \fBnetconfig\fR database. This

```

```

377 option can be specified more than once, so as to compile a server that serves
378 multiple transports.
379 .
380 .It Fl o Ar outfile
396 .RE

398 .sp
399 .ne 2
400 .na
401 \fB\fB-o\fR \fIoutfile\fR\fR
402 .ad
403 .RS 18n
381 Specifies the name of the output file. If none is specified, standard output is
382 used
383 .Po
384 .Fl c , h , l , m , n , s , "Sc" , "Sm" , "Ss" ,
385 and
386 .Fl t
387 modes only
388 .Pc .
389 .
390 .It Fl s Ar nettype
405 used (\fB-c\fR, \fB-h\fR, \fB-l\fR, \fB-m\fR, \fB-n\fR, \fB-s\fR, \fB-Sc\fR,
406 \fB-Sm\fR, \fB-Ss\fR, and \fB-t\fR modes only).
407 .RE

409 .sp
410 .ne 2
411 .na
412 \fB\fB-s\fR \fInettype\fR\fR
413 .ad
414 .RS 18n
391 Compiles into server-side stubs for all the transports belonging to the class
392 Ar nettype .
393 The supported classes are
394 .Sy netpath ,
395 .Sy visible ,
396 .Sy circuit_n ,
397 .Sy circuit_v ,
398 .Sy datagram_n ,
399 .Sy datagram_v ,
400 .Sy tcp ,
401 and
402 .Sy udp
403 (see
404 .Xr rpc 3NSL
405 for the meanings associated with
406 these classes). This option can be specified more than once.
407 .Em Note:
408 The transports are chosen at run time and not at compile time.
409 .
410 .It Fl "Sc"
416 \fInettype\fR. The supported classes are \fBnetpath\fR, \fBvisible\fR,
417 \fBcircuit_n\fR, \fBcircuit_v\fR, \fBdatagram_n\fR, \fBdatagram_v\fR,
418 \fBtcp\fR, and \fBudp\fR (see \fBrpc\fR(3NSL) for the meanings associated with
419 these classes). This option can be specified more than once. \fBNote:\fR The
420 transports are chosen at run time and not at compile time.
421 .RE

423 .sp
424 .ne 2
425 .na
426 \fB\fB-Sc\fR\fR
427 .ad
428 .RS 18n
411 Generates sample client code that uses remote procedure calls.

```

```

412 .
413 .It Fl "Sm"
430 .RE

432 .sp
433 .ne 2
434 .na
435 \fB\fB-Sm\fR\fR
436 .ad
437 .RS 18n
414 Generates a sample Makefile which can be used for compiling the application.
415 .
416 .It Fl "Ss"
439 .RE

441 .sp
442 .ne 2
443 .na
444 \fB\fB-Ss\fR\fR
445 .ad
446 .RS 18n
417 Generates sample server code that uses remote procedure calls.
418 .
419 .It Fl t
420 Compiles into RPC dispatch table.
421 .

422 .It Fl T
423 Generates the code to support RPC dispatch tables.
424 .Lp
425 The options
426 .Fl c , h , l , m , s R , "Sc" , "Sm" , "Ss" ,
427 and
428 .Fl t
429 are used exclusively to generate a
430 particular type of file, while the options
431 .Fl D
432 and
433 .Fl T
434 are global and can be used with the other options.
435 .
436 .It Fl v
448 .RE

450 .sp
451 .ne 2
452 .na
453 \fB\fB-t\fR\fR
454 .ad
455 .RS 18n
456 Compiles into \fBRPC\fR dispatch table.
457 .RE

459 .sp
460 .ne 2
461 .na
462 \fB\fB-T\fR\fR
463 .ad
464 .RS 18n
465 Generates the code to support \fBRPC\fR dispatch tables.
466 .sp
467 The options \fB-c\fR, \fB-h\fR, \fB-l\fR, \fB-m\fR, \fB-s\fR, \fB-Sc\fR,
468 \fB-Sm\fR, \fB-Ss\fR, and \fB-t\fR are used exclusively to generate a
469 particular type of file, while the options \fB-D\fR and \fB-T\fR are global and
470 can be used with the other options.
471 .RE

```

```

473 .sp
474 .ne 2
475 .na
476 \fB\fB-v\fR\fR
477 .ad
478 .RS 18n
437 Displays the version number.
438 .
439 .It Fl Y Ar pathname
440 Gives the name of the directory where
441 .Nm
442 starts looking for the C preprocessor.
443 .El
444 .Sh OPERANDS
480 .RE

482 .sp
483 .ne 2
484 .na
485 \fB\fB-Y\fR \fIpathname\fR\fR
486 .ad
487 .RS 18n
488 Gives the name of the directory where \fBrpcgen\fR starts looking for the C
489 preprocessor.
490 .RE

492 .SH OPERANDS
493 .sp
494 .LP
445 The following operand is supported:
446 .Bl -tag -width Ar
447 .
448 .It Ar infile
496 .sp
497 .ne 2
498 .na
499 \fB\fIinfile\fR\fR
500 .ad
501 .RS 10n
449 input file
450 .El
451 .Sh EXIT STATUS
452 .Ex -std
453 .Sh EXAMPLES
454 .Ss Example 1 Generating the output files and dispatch table
503 .RE

505 .SH EXAMPLES
506 .LP
507 \fBExample 1\fR Generating the output files and dispatch table
508 .sp
509 .LP
455 The following entry
456 .Lp
457 .Dl example% rpcgen -T prot.x
458 .Lp
459 generates all the five files:
460 .Pa prot.h , prot_clnt.c R, prot_svc.c , prot_xdr.c ,
461 and
462 .Pa prot_tbl.i .
463 .
464 .Ss Example 2 Sending headers to standard output

512 .sp
513 .in +2
514 .nf

```

```

515 example% \fBrpcgen -T prot.x\fR
516 .fi
517 .in -2
518 .sp
520 .sp
521 .LP
522 generates all the five files: \fBprot.h\fR, \fBprot_clnt.c\fR,
523 \fBprot_svc.c\fR, \fBprot_xdr.c\fR, and \fBprot_tbl.i\fR.
525 .LP
526 \fBExample 2\fR \fRSending headers to standard output
527 .sp
528 .LP
465 The following example sends the C data-definitions (header) to the standard
466 output:
467 .Lp
468 .Dl example% rpcgen -h prot.x
469 .
470 .Ss Example 3 Sending a test version
471 To send the test version of the
472 .Fl DTEST ,
473 server side stubs for all the
474 transport belonging to the class
475 .Sy datagram_n
476 to standard output, use:
477 .Lp
478 .Dl example% rpcgen -s datagram_n -DTEST prot.x
479 .
480 .Ss Example 4 Creating server side stubs
481 To create the server side stubs for the transport indicated by
482 .Ar netid
483 .Sy tcp ,
484 use:
485 .Lp
486 .Dl example% rpcgen -n tcp -o prot_svc.c prot.x
487 .Sh SEE ALSO
488 .Xr inetd 1M ,
489 .Xr listen 1M ,
490 .Xr rpc 3NSL ,
491 .Xr rpc_control 3NSL ,
492 .Xr rpc_svc_calls 3NSL ,
493 .Xr syslog 3C ,
494 .Xr netconfig 4 ,
495 .Xr attributes 5
496 .Rs
497 .%B ONC+ Developer\&#39;s Guide
498 .Re

532 .sp
533 .in +2
534 .nf
535 example% \fBrpcgen -h prot.x\fR
536 .fi
537 .in -2
538 .sp
540 .LP
541 \fBExample 3\fR \fRSending a test version
542 .sp
543 .LP
544 To send the test version of the \fB-DTEST\fR, server side stubs for all the
545 transport belonging to the class \fBdatagram_n\fR to standard output, use:
547 .sp
548 .in +2

```

```

549 .nf
550 example% \fBrpcgen -s datagram_n -DTEST prot.x\fR
551 .fi
552 .in -2
553 .sp
555 .LP
556 \fBExample 4\fR \fRCreating server side stubs
557 .sp
558 .LP
559 To create the server side stubs for the transport indicated by \fInetid\fR
560 \fBtcp\fR, use:
562 .sp
563 .in +2
564 .nf
565 example% \fBrpcgen -n tcp -o prot_svc.c prot.x\fR
566 .fi
567 .in -2
568 .sp
570 .SH EXIT STATUS
571 .sp
572 .ne 2
573 .na
574 \fB\fB0\fR\fR
575 .ad
576 .RS 6n
577 Successful operation.
578 .RE

580 .sp
581 .ne 2
582 .na
583 \fB\fB>0\fR\fR
584 .ad
585 .RS 6n
586 An error occurred.
587 .RE

589 .SH SEE ALSO
590 .sp
591 .LP
592 \fBinetd\fR(1M), \fBlisten\fR(1M), \fBrpc\fR(3NSL), \fBrpc_control\fR(3NSL),
593 \fBrpc_svc_calls\fR(3NSL), \fBsyslog\fR(3C), \fBnetconfig\fR(4),
594 \fBattributes\fR(5)
595 .sp
596 .LP
597 The \fBrpcgen\fR chapter in the \fIONC+ Developer\&#39;s Guide\fR manual.

```