```
**********************************************************
   10166 Sun Aug  3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_clntout.c
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_clntout.c, Client-stub outputter for the RPC protocol compiler
 */
#include <stdio.h>
#include <string.h>
#include <rpc/types.h>
#include "rpc_parse.h"
#include "rpc_util.h"

extern void pdeclaration(char *, declaration *, int, char *);
extern void printarglist(proc_list *, char *, char *, char *);

static void write_program(definition *);
static void printbody(proc_list *);

static char RESULT[] = "clnt_res";

#define DEFAULT_TIMEOUT 25      /* in seconds */

void
write_stubs(void)
{
```



```
        list *l;
        definition *def;

        f_print(fout,
            "\n/* Default timeout can be changed using clnt_control() */\n");
        f_print(fout, "static struct timeval TIMEOUT = { %d, 0 };\n",
            DEFAULT_TIMEOUT);
        for (l = defined; l != NULL; l = l->next) {
                def = (definition *) l->val;
                if (def->def_kind == DEF_PROGRAM) {
                        write_program(def);
                }
        }
}
```

_____***unchanged_portion_omitted_***

```
/*
 * Writes out declarations of procedure's argument list.
 * In either ANSI C style, in one of old rpcgen style (pass by reference),
 * or new rpcgen style (multiple arguments, pass by value);
 */

/* sample addargname = "clnt"; sample addargtype = "CLIENT * " */

void
printarglist(proc_list *proc, char *result, char *addargname, char *addargtype)
{
        bool_t oneway = streq(proc->res_type, "oneway");
        decl_list *l;

        if (!newstyle) {
                /* old style: always pass argument by reference */
                if (Cflag) {    /* C++ style heading */
                f_print(fout, "(");
                ptype(proc->args.decls->decl.prefix,
                    proc->args.decls->decl.type, 1);

                if (mtflag) {   /* Generate result field */
                        f_print(fout, "*argp, ");
                        if (!oneway) {
                                ptype(proc->res_prefix, proc->res_type, 1);
                                ptype(proc->res_prefix,
                                    proc->res_type, 1);
                                f_print(fout, "*%s, ", result);
                        }
                        f_print(fout, "%s%s)\n", addargtype, addargname);
                        f_print(fout, "%s%s)\n",
                            addargtype, addargname);
                } else
                        f_print(fout, "*argp, %s%s)\n", addargtype, addargname);
                        f_print(fout, "*argp, %s%s)\n",
                            addargtype, addargname);
                } else {
                        if (!mtflag)
                                f_print(fout, "(argp, %s)\n", addargname);
                        else {
                                f_print(fout, "(argp, ");
                                if (!oneway) {
                                        f_print(fout, "%s, ",
                                            result);
                                }
                                f_print(fout, "%s)\n",
                                    addargname);
                        }
                        f_print(fout, "\t");
                        ptype(proc->args.decls->decl.prefix,
```

```
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             }
134     } else if (streq(proc->args.decls->decl.type, "void")) {
135             /* newstyle, 0 argument */
136             if (mtflag) {
137                     f_print(fout, "(");

163                     if (Cflag) {
139                     if (!oneway) {
140                             ptype(proc->res_prefix, proc->res_type, 1);
165                                     ptype(proc->res_prefix,
166                                             proc->res_type, 1);
141                             f_print(fout, "*%s, ", result);
142                     }
143                     f_print(fout, "%s%s)\n", addargtype, addargname);
169                             f_print(fout, "%s%s)\n",
170                                 addargtype, addargname);
171                     } else
172                             f_print(fout, "(%s)\n", addargname);

145             } else
175             if (Cflag)
146                     f_print(fout, "(%s%s)\n", addargtype, addargname);
177             else
178                     f_print(fout, "(%s)\n", addargname);
147     } else {
148             /* new style, 1 or multiple arguments */
181             if (!Cflag) {
149             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);

188                     f_print(fout, "%s)\n", addargname);
150             for (l = proc->args.decls; l != NULL; l = l->next) {
151                     pdeclaration(proc->args.argname, &l->decl, 0, ", ");
190                             pdeclaration(proc->args.argname,
191                                 &l->decl, 1, ";\n");
152             }
153             if (mtflag && !oneway) {
194                             f_print(fout, "\t");
154                     ptype(proc->res_prefix, proc->res_type, 1);
196                             f_print(fout, "*%s;\n", result);
197                     }

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);
155                     f_print(fout, "*%s, ", result);

157             }
158             f_print(fout, "%s%s)\n", addargtype, addargname);
159     }
212     }
```



```
214         if (!Cflag)
215                 f_print(fout, "\t%s%s;\n", addargtype, addargname);
160 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   21350 Sun Aug  3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_cout.c
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_cout.c, XDR routine outputter for the RPC protocol compiler
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "rpc_parse.h"
#include "rpc_util.h"

extern void crash(void);

static void print_header(definition *);
static void print_trailer(void);
static void emit_enum(definition *);
static void emit_program(definition *);
static void emit_union(definition *);
static void emit_struct(definition *);
static void emit_typedef(definition *);
static void print_stat(int, declaration *);
static void emit_inline(int, declaration *, int);
static void emit_inline64(int, declaration *, int);
```



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

/*
 * Emit the C-routine for the given definition
 */
void
emit(definition *def)
{
        if (def->def_kind == DEF_CONST)
                return;
        if (def->def_kind == DEF_PROGRAM) {
                emit_program(def);
                return;
        }
        if (def->def_kind == DEF_TYPEDEF) {
                /*
                 * now we need to handle declarations like
                 * struct typedef foo foo;
                 * since we dont want this to be expanded into 2 calls
                 * to xdr_foo
                 */

                if (strcmp(def->def.ty.old_type, def->def_name) == 0)
                        return;
        };
        print_header(def);
        switch (def->def_kind) {
        case DEF_UNION:
                emit_union(def);
                break;
        case DEF_ENUM:
                emit_enum(def);
                break;
        case DEF_STRUCT:
                emit_struct(def);
                break;
        case DEF_TYPEDEF:
                emit_typedef(def);
                break;
        }
        print_trailer();
}
```

_____***unchanged_portion_omitted_***

```
static void
print_generic_header(char *procname, int pointerp)
{
        f_print(fout, "\n");
        f_print(fout, "bool_t\n");
        if (Cflag) {
        f_print(fout, "xdr_%s(", procname);
        f_print(fout, "XDR *xdrs, ");
        f_print(fout, "%s ", procname);
        if (pointerp)
                f_print(fout, "*");
        f_print(fout, "objp)\n{\n\n");
        } else {
                f_print(fout, "xdr_%s(xdrs, objp)\n", procname);
                f_print(fout, "\tXDR *xdrs;\n");
                f_print(fout, "\t%s ", procname);
                if (pointerp)
                        f_print(fout, "*");
                f_print(fout, "objp;\n{\n\n");
```

```
143         }
137 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   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>
 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 parglist(proc_list *, char *);
 59 static void penumdef(definition *);
 60 static void ptypedef(definition *);
```



```
 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         pvname(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                             locase(def->def_name), vers->vers_num);
285                         f_print(fout,
286                             "extern int %s_%s_nproc;\n",
287                             locase(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,
```

```
                                        proc->proc_num, PROCEDURE);
                                f_print(fout, "%s", ext);
                                pprocdef(proc, vers, NULL, 0, 2);

                                if (mtflag) {
                                        f_print(fout, "%s", ext);
                                        pprocdef(proc, vers, NULL, 1, 2);
                                }
                        }
                        pfreeprocdef(def->def_name, vers->vers_num, 2);
                } else {
                        for (i = 1; i < 3; i++) {
                                if (i == 1) {
                                        f_print(fout, "\n#if defined(__STDC__)"
                                            " || defined(__cplusplus)\n");
                                        ext = "extern  ";
                                } else {
                                        f_print(fout, "\n#else /* K&R C */\n");
                                        ext = "extern  ";
                                }

                                for (proc = vers->procs; proc != NULL;
                                    proc = proc->next) {
                        if (!define_printed(proc,
                            def->def.pr.versions)) {
                                puldefine(proc->proc_name,
                                    proc->proc_num, PROCEDURE);
                        }
                        f_print(fout, "extern  ");
                        pprocdef(proc, vers, "CLIENT *", 0);
                        f_print(fout, "extern  ");
                        pprocdef(proc, vers, "struct svc_req *", 1);
                                        f_print(fout, "%s", ext);
                                        pprocdef(proc, vers, "CLIENT *", 0, i);
                                        f_print(fout, "%s", ext);
                                        pprocdef(proc, vers,
                                            "struct svc_req *", 1, i);
                }
                pfreeprocdef(def->def_name, vers->vers_num);
                                pfreeprocdef(def->def_name, vers->vers_num, i);
        }
                        f_print(fout, "#endif /* K&R C */\n");
                }
        }
}

void
pprocdef(proc_list *proc, version_list *vp, char *addargtype, int server_p)
pprocdef(proc_list *proc, version_list *vp, char *addargtype, int server_p,
                                                                int mode)
{
        if (mtflag) {
                /* Print MT style stubs */
                if (server_p)
                        f_print(fout, "bool_t ");
                else
                        f_print(fout, "enum clnt_stat ");
        } else {
                ptype(proc->res_prefix, proc->res_type, 1);
                f_print(fout, "* ");
        }
        if (server_p)
                pvname_svc(proc->proc_name, vp->vers_num);
        else
                pvname(proc->proc_name, vp->vers_num);
```



```
        /*
         *  mode  1 = ANSI-C, mode 2 = K&R C
         */
        if (mode == 1)
        parglist(proc, addargtype);
        else
                f_print(fout, "();\n");
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   31733 Sun Aug  3 11:09:16 2014
new/usr/src/cmd/rpcgen/rpc_main.c
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_main.c, Top level of the RPC protocol compiler.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/file.h>
#include <sys/stat.h>
#include "rpc_parse.h"
#include "rpc_util.h"
#include "rpc_scan.h"


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



```
extern void reinitialize(void);
extern void crash(void);
extern void add_type(int, char *);
extern void add_sample_msg(void);

static void svc_output(char *, char *, int, char *);
static void clnt_output(char *, char *, int, char *);
static void c_output(char *, char *, int, char *);
static void mkfile_output(struct commandline *);
static void c_initialize(void);
static void h_output(char *, char *, int, char *);
static void s_output(int, char *[], char *, char *, int, char *, int, int);
static void l_output(char *, char *, int, char *);
static void t_output(char *, char *, int, char *);
static int do_registers(int, char *[]);
static uint_t parseargs(int, char *[], struct commandline *);
static void usage(void);
static void version_info(void);
static void options_usage(void);

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

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


static char *cmdname;

static char *svcclosetime = "120";
static char *CPP = SUNOS_CPP;
static char CPPFLAGS[] = "-C";
static char pathbuf[MAXPATHLEN + 1];
static char *allv[] = {
        "rpcgen", "-s", "udp", "-s", "tcp",
};
______unchanged_portion_omitted_
static int allnc = sizeof (allnv)/sizeof (allnv[0]);

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


#define ARGLISTLEN      20
#define FIXEDARGS       2

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


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

```
131 int inlinelen = INLINE;
132 /*
133  * Length at which to start doing an inline. INLINE = default
134  * if 0, no xdr_inline code
135  */

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) */
139 int Cflag = 0;          /* ANSI C syntax */
141 int CCflag = 0;         /* C++ files */
142 static int allfiles;   /* generate all files */
143 int tirpcflag = 1;    /* generating code for tirpc, by default */
144 xdrfunc *xdrfunc_head = NULL; /* xdr function list */
145 xdrfunc *xdrfunc_tail = NULL; /* xdr function list */
146 pid_t childpid;


149 int
150 main(int argc, char *argv[])
151 {
152         struct commandline cmd;

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);

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                 }

206                 if (allfiles) {
207                         reinitialize();
208                         svc_output(cmd.infile, "-DRPC_SERVER", EXTEND,
209                             "_server.c");
210                         reinitialize();
211                         clnt_output(cmd.infile, "-DRPC_CLIENT", EXTEND,
212                             "_client.c");

214                 }
215                 if (allfiles || (cmd.makefileflag == 1)) {
216                         reinitialize();
217                         mkfile_output(&cmd);
218                 }

220         }
221         return (nonfatalerrors);
222 }
```

_____***unchanged_portion_omitted_***

```
505 /*
506  * Compile into an XDR header file
507  */


510 static void
511 h_output(char *infile, char *define, int extend, char *outfile)
512 {
513         definition *def;
514         char *outfilename;
515         long tell;
516         char *guard;
517         list *l;
518         xdrfunc *xdrfuncp;
518         int i;

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_";

529         f_print(fout, "#ifndef _%s\n#define     _%s\n\n", guard, guard);

531         f_print(fout, "#include <rpc/rpc.h>\n");

533         if (mtflag) {
534                 f_print(fout, "#ifndef _KERNEL\n");
535                 f_print(fout, "#include <synch.h>\n");
536                 f_print(fout, "#include <thread.h>\n");
537                 f_print(fout, "#endif /* !_KERNEL */\n");
538         };

540         /* put the C++ support */
```

```
541         if (!CCflag) {
541         if (Cflag && !CCflag) {
542                 f_print(fout, "\n#ifdef __cplusplus\n");
543                 f_print(fout, "extern \"C\" {\n");
544                 f_print(fout, "#endif\n\n");
545         }

547         /* put in a typedef for quadprecision. */
547         /* put in a typedef for quadprecision. Only with Cflag */

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                 f_print(fout, rpcgen_table_dcl2);
560         }

562         tell = ftell(fout);

564         /* print data definitions */
565         while (def = get_definition())
566                 print_datadef(def);

568         /*
569          * print function declarations.
570          *  Do this after data definitions because they might be used as
571          *  arguments for functions
572          */
573         for (l = defined; l != NULL; l = l->next)
574                 print_funcdef(l->val);
575         /* Now  print all xdr func declarations */
576         if (xdrfunc_head != NULL) {
577                 f_print(fout, "\n/* the xdr functions */\n");

579                 if (CCflag) {
580                         f_print(fout, "\n#ifdef __cplusplus\n");
581                         f_print(fout, "extern \"C\" {\n");
582                         f_print(fout, "#endif\n");
583                 }

585                 f_print(fout, "\n");
585                 if (!Cflag) {
586                         xdrfuncp = xdrfunc_head;
587                         while (xdrfuncp != NULL) {
588                                 print_xdr_func_def(xdrfuncp->name,
589                                     xdrfuncp->pointerp, 2);
590                                 xdrfuncp = xdrfuncp->next;
591                         }
592                 } else {
593                         for (i = 1; i < 3; i++) {
594                                 if (i == 1)
595                                         f_print(fout,
596 "\n#if defined(__STDC__) || defined(__cplusplus)\n");
597                                 else
598                                         f_print(fout, "\n#else /* K&R C */\n");

587                 xdrfuncp = xdrfunc_head;
588                 while (xdrfuncp != NULL) {
589                         print_xdr_func_def(xdrfuncp->name, xdrfuncp->pointerp);
602                                         print_xdr_func_def(xdrfuncp->name,
```



```
603                                             xdrfuncp->pointerp, i);
590                         xdrfuncp = xdrfuncp->next;
591                 }
592                 f_print(fout, "\n");
593         }
607                         f_print(fout, "\n#endif /* K&R C */\n");
608                 }
609         }

595         if (extend && tell == ftell(fout)) {
596                 (void) unlink(outfilename);
597         }

615         if (Cflag) {
599         f_print(fout, "\n#ifdef __cplusplus\n");
600         f_print(fout, "}\n");
601         f_print(fout, "#endif\n");
619         }

603         f_print(fout, "\n#endif /* !_%s */\n", guard);
604 }

606 /*
607  * Compile into an RPC service
608  */
609 static void
610 s_output(int argc, char *argv[], char *infile, char *define, int extend,
611                                         char *outfile, int nomain, int netflag)
612 {
613         char *include;
614         definition *def;
615         int foundprogram = 0;
616         char *outfilename;

618         open_input(infile, define);
619         outfilename = extend ? extendfile(infile, outfile) : outfile;
620         open_output(infile, outfilename);
621         add_warning();
622         if (infile && (include = extendfile(infile, ".h"))) {
623                 f_print(fout, "#include \"%s\"\n", include);
624                 free(include);
625         } else
626                 f_print(fout, "#include <rpc/rpc.h>\n");

628         f_print(fout, "#include <stdio.h>\n");
629         f_print(fout, "#include <stdlib.h> /* getenv, exit */\n");
630         f_print(fout, "#include <signal.h>\n");

650         if (Cflag) {
632         f_print(fout,
633         "#include <rpc/pmap_clnt.h> /* for pmap_unset */\n");
634         f_print(fout, "#include <string.h> /* strcmp */\n");

654         }
636         if (strcmp(svcclosetime, "-1") == 0)
637                 indefinitewait = 1;
638         else if (strcmp(svcclosetime, "0") == 0)
639                 exitnow = 1;
640         else if (inetdflag || pmflag)
641                 timerflag = 1;

643         if (!tirpcflag && inetdflag)
644                 f_print(fout, "#include <sys/termios.h> /* TIOCNOTTY */\n");
645         if (inetdflag || pmflag)
664         if (Cflag && (inetdflag || pmflag))
646                 if (tirpcflag)
```

```
647                         f_print(fout, "#include <unistd.h> /* setsid */\n");
648         if (tirpcflag)
649                 f_print(fout, "#include <sys/types.h>\n");

670         f_print(fout, "#include <memory.h>\n");
651         f_print(fout, "#include <stropts.h>\n");
652         if (inetdflag || !tirpcflag) {
653                 f_print(fout, "#include <sys/socket.h>\n");
654                 f_print(fout, "#include <netinet/in.h>\n");
655                 f_print(fout, "#include <rpc/svc_soc.h>\n");
656         }

658         if ((netflag || pmflag) && tirpcflag && !nomain)
659                 f_print(fout, "#include <netconfig.h>\n");
660         if (tirpcflag)
661                 f_print(fout, "#include <sys/resource.h> /* rlimit */\n");
662         if (logflag || inetdflag || pmflag)
663                 f_print(fout, "#include <syslog.h>\n");

665         /* for ANSI-C */
666         f_print(fout, "\n#ifndef SIG_PF\n"
667             "#define    SIG_PF void(*)(int)\n#endif\n");
686         if (Cflag)
687                 f_print(fout,
688                     "\n#ifndef SIG_PF\n#define  SIG_PF void(*)\
689 (int)\n#endif\n");

669         f_print(fout, "\n#ifdef DEBUG\n#define  RPC_SVC_FG\n#endif\n");
670         if (timerflag)
671                 f_print(fout, "\n#define        _RPCSVC_CLOSEDOWN %s\n",
672                     svcclosetime);
673         while (def = get_definition())
674                 foundprogram |= (def->def_kind == DEF_PROGRAM);
675         if (extend && !foundprogram) {
676                 (void) unlink(outfilename);
677                 return;
678         }
679         write_most(infile, netflag, nomain);
680         if (!nomain) {
681                 if (!do_registers(argc, argv)) {
682                         if (outfilename)
683                                 (void) unlink(outfilename);
684                         usage();
685                 }
686                 write_rest();
687         }
688 }

690 /*
691  * generate client side stubs
692  */
693 static void
694 l_output(char *infile, char *define, int extend, char *outfile)
695 {
696         char *include;
697         definition *def;
698         int foundprogram = 0;
699         char *outfilename;

701         open_input(infile, define);
702         outfilename = extend ? extendfile(infile, outfile) : outfile;
703         open_output(infile, outfilename);
704         add_warning();
727         if (Cflag)
705         f_print(fout, "#include <memory.h> /* for memset */\n");
706         if (infile && (include = extendfile(infile, ".h"))) {
```



```
707                 f_print(fout, "#include \"%s\"\n", include);
708                 free(include);
709         } else
710                 f_print(fout, "#include <rpc/rpc.h>\n");

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");

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;

1019         cmdname = argv[0];
1020         cmd->infile = cmd->outfile = NULL;
1021         if (argc < 2)
1022                 return (0);
1023         allfiles = 0;
1024         flag['c'] = 0;
1025         flag['h'] = 0;
1026         flag['l'] = 0;
1027         flag['m'] = 0;
1028         flag['o'] = 0;
1029         flag['s'] = 0;
1030         flag['n'] = 0;
1031         flag['t'] = 0;
1032         flag['S'] = 0;
1033         flag['C'] = 0;
1034         flag['M'] = 0;

1036         for (i = 1; i < argc; i++) {
1037                 if (argv[i][0] != '-') {
1038                         if (cmd->infile) {
1039                                 f_print(stderr,
1040         "Cannot specify more than one input file.\n");

1042                                 return (0);
1043                         }
1044                         cmd->infile = argv[i];
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                            case 'h':
1054                            case 'l':
1055                            case 'm':
1056                            case 't':
1057                                    if (flag[c])
1058                                            return (0);
1059                                    flag[c] = 1;
1060                                    break;
1061                            case 'S':
1062                                    /*
1063                                     * sample flag: Ss or Sc.
1064                                     *  Ss means set flag['S'];
1065                                     *  Sc means set flag['C'];
1066                                     *  Sm means set flag['M'];
1067                                     */
1068                                    ch = argv[i][++j]; /* get next char */
1069                                    if (ch == 's')
1070                                            ch = 'S';
1071                                    else if (ch == 'c')
1072                                            ch = 'C';
1073                                    else if (ch == 'm')
1074                                            ch = 'M';
1075                                    else
1076                                            return (0);

1078                                    if (flag[ch])
1079                                            return (0);
1080                                    flag[ch] = 1;
1081                                    break;
1082                            case 'C': /* ANSI C syntax (default) */
1105                            case 'C': /* ANSI C syntax */
1106                                    Cflag = 1;
1083                                    ch = argv[i][j+1]; /* get next char */

1085                                    if (ch != 'C')
1086                                            break;
1087                                    /* Undocumented C++ mode */
1088                                    CCflag = 1;
1089                                    break;
1090                            case 'b':
1091                                    /*
1092                                     *  Turn TIRPC flag off for
1093                                     *  generating backward compatible
1094                                     *  code
1095                                     */
1096                                    tirpcflag = 0;
1097                                    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':
```



```
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] != '-' ||
1131                                        argv[i][j + 1] != 0)
1132                                            return (0);
1133                                    flag[c] = 1;
1134                                    if (++i == argc)
1135                                            return (0);
1136                                    if (c == 'o') {
1137                                            if (cmd->outfile)
1138                                                    return (0);
1139                                            cmd->outfile = argv[i];
1140                                    }
1141                                    goto nextarg;
1142                            case 'D':
1143                                    if (argv[i][j - 1] != '-')
1144                                            return (0);
1145                                    (void) addarg(argv[i]);
1146                                    goto nextarg;
1147                            case 'v':
1148                                    version_info();
1149                                    return (0);
1150                            case 'Y':
1151                                    if (++i == argc)
1152                                            return (0);
1153                                    (void) strcpy(pathbuf, argv[i]);
1154                                    (void) strcat(pathbuf, "/cpp");
1155                                    CPP = pathbuf;
1156                                    cppDefined = 1;
1157                                    goto nextarg;
1158                            case 'r':
1159                                    rflag = !rflag;
1160                                    break;
1161                            default:
1162                                    return (0);
1163                            }
1164                    }
1165            nextarg:
1166                    ;
1167            }
1168    }

1170    cmd->cflag = flag['c'];
1171    cmd->hflag = flag['h'];
1172    cmd->lflag = flag['l'];
1173    cmd->mflag = flag['m'];
1174    cmd->nflag = flag['n'];
1175    cmd->sflag = flag['s'];
1176    cmd->tflag = flag['t'];
1177    cmd->Ssflag = flag['S'];
1178    cmd->Scflag = flag['C'];
1179    cmd->makefileflag = flag['M'];

1181    if (tirpcflag) {
1182            if (inetdflag) {
```

```
1183                        f_print(stderr,
1184                            "Cannot use -I flag without -b flag.\n");
1185                        return (0);
1186                }
1187                pmflag = 1;
1188        } else {                /* 4.1 mode */
1189                pmflag = 0;     /* set pmflag only in tirpcmode */
1190                inetdflag = 1;  /* inetdflag is TRUE by default */
1191                if (cmd->nflag) { /* netid needs TIRPC */
1192                        f_print(stderr,
1193                            "Cannot use netid flag without TIRPC.\n");
1194                        return (0);
1195                }
1196        }

1198        if (newstyle && (tblflag || cmd->tflag)) {
1199                f_print(stderr, "Cannot use table flags with newstyle.\n");
1200                return (0);
1201        }

1203        /* check no conflicts with file generation flags */
1204        nflags = cmd->cflag + cmd->hflag + cmd->lflag + cmd->mflag +
1205            cmd->sflag + cmd->nflag + cmd->tflag + cmd->Ssflag +
1206            cmd->Scflag + cmd->makefileflag;

1208        if (nflags == 0) {
1209                if (cmd->outfile != NULL || cmd->infile == NULL)
1210                        return (0);
1211        } else if (cmd->infile == NULL &&
1212            (cmd->Ssflag || cmd->Scflag || cmd->makefileflag)) {
1213                f_print(stderr, "\"infile\" is required for template"
1214                    " generation flags.\n");
1215                return (0);
1216        }
1217        if (nflags > 1) {
1218                f_print(stderr,
1219                    "Cannot have more than one file generation flag.\n");
1220                return (0);
1221        }
1222        return (1);
1223 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8516 Sun Aug  3 11:09:17 2014
new/usr/src/cmd/rpcgen/rpc_sample.c
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_sample.c, Sample client-server code outputter
 * for the RPC protocol compiler
 */

#include <stdio.h>
#include <string.h>
#include "rpc_parse.h"
#include "rpc_util.h"


static char RQSTP[] = "rqstp";

extern void printarglist(proc_list *, char *, char *, char *);

static void write_sample_client(char *, version_list *);
static void write_sample_server(definition *);
static void return_type(proc_list *);

void
write_sample_svc(definition *def)
{
        if (def->def_kind != DEF_PROGRAM)
                return;
        write_sample_server(def);
}
```



```
_____unchanged_portion_omitted_

static void
write_sample_client(char *program_name, version_list *vp)
{
        proc_list *proc;
        int i;
        decl_list *l;

        f_print(fout, "\n\nvoid\n");
        pvname(program_name, vp->vers_num);
        if (Cflag)
        f_print(fout, "(char *host)\n{\n");
        else
                f_print(fout, "(host)\n\tchar *host;\n{\n");
        f_print(fout, "\tCLIENT *clnt;\n");

        i = 0;
        for (proc = vp->procs; proc != NULL; proc = proc->next) {
                f_print(fout, "\t");
                if (mtflag) {
                        f_print(fout, "enum clnt_stat retval_%d;\n", ++i);
                        if (!streq(proc->res_type, "oneway")) {
                                f_print(fout, "\t");
                                if (!streq(proc->res_type, "void"))
                                        ptype(proc->res_prefix,
                                            proc->res_type, 1);
                                else
                                        f_print(fout, "void *");
                                f_print(fout, "result_%d;\n", i);
                        }
                } else {
                        ptype(proc->res_prefix, proc->res_type, 1);
                        f_print(fout, " *result_%d;\n", ++i);
                }
                /* print out declarations for arguments */
                if (proc->arg_num < 2 && !newstyle) {
                        f_print(fout, "\t");
                        if (!streq(proc->args.decls->decl.type, "void"))
                                ptype(proc->args.decls->decl.prefix,
                                    proc->args.decls->decl.type, 1);
                        else
                                /* cannot have "void" type */
                                f_print(fout, "char * ");
                        f_print(fout, " ");
                        pvname(proc->proc_name, vp->vers_num);
                        f_print(fout, "_arg;\n");
                } else if (!streq(proc->args.decls->decl.type, "void")) {
                        for (l = proc->args.decls; l != NULL; l = l->next) {
                                f_print(fout, "\t");
                                ptype(l->decl.prefix, l->decl.type, 1);
                                if (strcmp(l->decl.type, "string") == 1)
                                        f_print(fout, " ");
                                pvname(proc->proc_name, vp->vers_num);
                                f_print(fout, "_%s;\n", l->decl.name);
                        }
                }
        }

        /* generate creation of client handle */
        f_print(fout, "\n#ifndef\tDEBUG\n");
        f_print(fout, "\tclnt = clnt_create(host, %s, %s, \"%s\");\n",
```

```
140             program_name, vp->vers_name, tirpcflag? "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\t/* DEBUG */\n\n");
145
146         /* generate calls to procedures */
147         i = 0;
148         for (proc = vp->procs; proc != NULL; proc = proc->next) {
149                 if (mtflag)
150                         f_print(fout, "\tretval_%d = ", ++i);
151                 else
152                         f_print(fout, "\tresult_%d = ", ++i);
153                 pvname(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                         pvname(proc->proc_name, vp->vers_num);
161                         if (mtflag) {
162                                 if (streq(proc->res_type, "oneway"))
163                                         f_print(fout, "_arg, 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 (mtflag) {
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                                 pvname(proc->proc_name, vp->vers_num);
183                                 f_print(fout, "_%s, ", l->decl.name);
184                         }
185                         if (mtflag) {
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 (mtflag) {
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\tDEBUG\n");
204         f_print(fout, "\tclnt_destroy(clnt);\n");
205         f_print(fout, "#endif\t /* DEBUG */\n");
```



```
206         f_print(fout, "}\n");
207 }
208
209 static void
210 write_sample_server(definition *def)
211 {
212         version_list *vp;
213         proc_list *proc;
214
215         for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
216                 for (proc = vp->procs; proc != NULL; proc = proc->next) {
217                         f_print(fout, "\n");
218                         if (!mtflag) {
219                                 return_type(proc);
220                                 f_print(fout, "*\n");
221                         } else {
222                                 f_print(fout, "bool_t\n");
223                         }
225                         if (Cflag || mtflag)
224                         pvname_svc(proc->proc_name, vp->vers_num);
227                         else
228                                 pvname(proc->proc_name, vp->vers_num);
225                         printarglist(proc, "result", RQSTP, "struct svc_req *");
226
227                         f_print(fout, "{\n");
228
229                         if (!mtflag) {
230                                 f_print(fout, "\tstatic ");
231                                 if ((!streq(proc->res_type, "void")) &&
232                                     (!streq(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\n");
242
243                         if (!mtflag)
244                                 if (!streq(proc->res_type, "void"))
245                                         f_print(fout,
246                                             "\treturn (&result);\n}\n");
247                                 else /* cast back to void * */
248                                         f_print(fout, "\treturn((void *) "
249                                             "&result);\n}\n");
250                         else
251                                 f_print(fout, "\treturn (retval);\n}\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);
261                         if (Cflag)
257                         f_print(fout, "_freeresult(SVCXPRT *transp,"
258                             " xdrproc_t xdr_result,"
259                             " caddr_t result)\n");
265                         else {
266                                 f_print(fout, "_freeresult(transp, xdr_result,"
267                                     " result)\n");
268                                 f_print(fout, "\tSVCXPRT *transp;\n");
269                                 f_print(fout, "\txdrproc_t xdr_result;\n");
270                                 f_print(fout, "\tcaddr_t result;\n");
271                         }
260                         f_print(fout, "{\n"
```

```
261                             "\t(void) xdr_free(xdr_result, result);\n"
262                             "\n\t/*\n\t * Insert additional freeing"
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");
305         if (Cflag)
293         f_print(fout, "int\nmain(int argc, char *argv[])\n{\n");
307         else
308                 f_print(fout, "int\nmain(argc, argv)\n\tint argc;\n"
309                     "\tchar *argv[];\n{\n");

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

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

*unchanged_portion_omitted_*

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

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_svcout.c, Server-skeleton outputter for the RPC protocol compiler
 */
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include "rpc_parse.h"
#include "rpc_util.h"

extern int nullproc(proc_list *);

static char RQSTP[] = "rqstp";
static char TRANSP[] = "transp";
static char ARG[] = "argument";
static char RESULT[] = "result";
static char ROUTINE[] = "local";
static char RETVAL[] = "retval";

#define ERRBUFLEN       256

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



```
static void write_programs(char *);
static void write_program(definition *, char *);
static void printerr(char *, char *);
static void write_svc_aux(int);
static void printif(char *, char *, char *, char *);
static void write_inetmost(char *);
static void print_return(char *);
static void print_pmapunset(char *);
static void print_err_message(const char *, const char *, ...);
static void write_msg_out(void);
static void write_timeout_func(void);
static void write_pm_most(char *, int);
static void write_rpc_svc_fg(char *, char *);
static void open_log_file(char *, char *);

static void
p_xdrfunc(char *rname, char *typename)
{
        if (Cflag) {
        f_print(fout, "\t\t_xdr_%s = (xdrproc_t)\n", rname);
        f_print(fout, "\t\t    xdr_%s;\n", stringfix(typename));
        } else {
                f_print(fout, "\t\t_xdr_%s = xdr_%s;\n",
                    rname, stringfix(typename));
        }
}
```

*unchanged_portion_omitted_*

```
/*
 * write the rest of the service
 */
void
write_rest(void)
{
        f_print(fout, "\n");
        if (inetdflag) {
                f_print(fout, "\tif (%s == (SVCXPRT *)NULL) {\n", TRANSP);
                print_err_message("\t\t", "could not create a handle");
                f_print(fout, "\t\texit(1);\n");
                f_print(fout, "\t}\n");
                if (timerflag) {
                        f_print(fout, "\tif (_rpcpmstart) {\n");
                        if (mtflag) {
                                f_print(fout,
"\t\tif (thr_create(NULL, 0, closedown, NULL, 0, NULL) != 0) {\n");
                                print_err_message("\t\t\t",
                                    "cannot create closedown thread");
                                f_print(fout, "\t\t\texit(1);\n");
                                f_print(fout, "\t\t}\n");
                                f_print(fout, "\t}\n");
                        } else {
                                f_print(fout,
                                "\t\t(void) signal(SIGALRM, %s closedown);\n",
                                    "(SIG_PF)");
                                    Cflag? "(SIG_PF)":"(void(*)())");
                                f_print(fout,
"\t\t(void) alarm(_RPCSVC_CLOSEDOWN/2);\n");
                                f_print(fout, "\t}\n");
                        }
                }
        }
        f_print(fout, "\tsvc_run();\n");
        print_err_message("\t", "svc_run returned");
        f_print(fout, "\texit(1);\n");
        f_print(fout, "\t/* NOTREACHED */\n");
        f_print(fout, "}\n");
```

```
312 }
_____unchanged_portion_omitted_

337 /*
338  * write out definition of internal function (e.g. _printmsg_1(...))
339  *  which calls server's defintion of actual function (e.g. printmsg_1(...)).
340  *  Unpacks single user argument of printmsg_1 to call-by-value format
341  *  expected by printmsg_1.
342  */
343 static void
344 write_real_program(definition *def)
345 {
346         version_list *vp;
347         proc_list *proc;
348         decl_list *l;

350         if (!newstyle)
351                 return;  /* not needed for old style */
352         for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
353                 for (proc = vp->procs; proc != NULL; proc = proc->next) {
354                         int oneway = streq(proc->res_type, "oneway");

356                         f_print(fout, "\n");
357                         if (proc->arg_num < 2 &&
358                             streq(proc->args.decls->decl.type, "void")) {
359                                 f_print(fout, "/* ARGSUSED */\n");
360                         }
361                         if (!mtflag)
362                                 internal_proctype(proc);
363                         else
364                                 f_print(fout, "int");
365                         f_print(fout, "\n_");
366                         pvname(proc->proc_name, vp->vers_num);

370                         if (Cflag) {
368                         f_print(fout, "(\n");
369                         f_print(fout, "    ");
370                         /* arg name */
371                         if (proc->arg_num > 1)
372                                 /* LINTED variable format */
373                                 f_print(fout, proc->args.argname);
374                         else
375                                 ptype(proc->args.decls->decl.prefix,
376                                     proc->args.decls->decl.type, 0);
377                         f_print(fout, " *argp,\n");
378                         if (mtflag) {
379                                 f_print(fout, "    ");
380                                 ptype(proc->res_prefix, proc->res_type, 1);
383                                         ptype(proc->res_prefix,
384                                             proc->res_type, 1);
381                                 f_print(fout, "*%s,\n", RESULT);
382                         }
383                         f_print(fout, "    struct svc_req *%s)\n", RQSTP);
387                                 f_print(fout, "    struct svc_req *%s)\n",
388                                     RQSTP);

390                         } else {
391                                 if (mtflag)
392                                         f_print(fout, "(argp, %s, %s)\n",
393                                             RESULT, RQSTP);
394                                 else
395                                         f_print(fout, "(argp, %s)\n", RQSTP);
396                                 /* arg name */
397                                 if (proc->arg_num > 1)
398                                         f_print(fout, "\t%s *argp;\n",
399                                             proc->args.argname);
```



```
400                                 else {
401                                         f_print(fout, "\t");
402                                         ptype(proc->args.decls->decl.prefix,
403                                             proc->args.decls->decl.type, 0);
404                                         f_print(fout, " *argp;\n");
405                                 }
406                                 if (mtflag)
407                                         f_print(fout, "\tvoid *%s;\n", RESULT);
408                                 f_print(fout, "\tstruct svc_req *%s;\n", RQSTP);
409                         }

385                         f_print(fout, "{\n");
386                         f_print(fout, "\treturn (");
387                         /* for mtflag, arguments are different */
414                         if (Cflag || mtflag)
388                         pvname_svc(proc->proc_name, vp->vers_num);
416                         else
417                                 pvname(proc->proc_name, vp->vers_num);
389                         f_print(fout, "(");
390                         if (proc->arg_num < 2) { /* single argument */
391                                 /* only print if non-void */
392                                 if (!streq(proc->args.decls->decl.type, "void"))
393                                         f_print(fout, "*argp, ");
394                         } else {
395                                 f_print(fout, "\n");
396                                 for (l = proc->args.decls;  l != NULL;
397                                     l = l->next)
398                                         f_print(fout, "\t    argp->%s,\n",
399                                             l->decl.name);
400                                 f_print(fout, "\t    ");
401                         }
402                         if (mtflag && !oneway)
403                                 f_print(fout, "%s, ", RESULT);
404                         f_print(fout, "%s));\n}\n", RQSTP);
405                 }
406         }
407 }

409 static void
410 write_program(definition *def, char *storage)
411 {
412         version_list *vp;
413         proc_list *proc;
414         int filled;

416         for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
417                 f_print(fout, "\n");
418                 if (storage != NULL) {
419                         f_print(fout, "%s ", storage);
420                 }
421                 f_print(fout, "void\n");
422                 pvname(def->def_name, vp->vers_num);

453                 if (Cflag) {
424                 f_print(fout, "(struct svc_req *%s, ", RQSTP);
425                 f_print(fout, "register SVCXPRT *%s)\n", TRANSP);
456                 } else {
457                         f_print(fout, "(%s, %s)\n", RQSTP, TRANSP);
458                         f_print(fout, " struct svc_req *%s;\n", RQSTP);
459                         f_print(fout, " register SVCXPRT *%s;\n", TRANSP);
460                 }

427                 f_print(fout, "{\n");

429                 filled = 0;
430                 f_print(fout, "\tunion {\n");
```

```
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                                 pvname(proc->proc_name, vp->vers_num);
442                                 f_print(fout, "_arg;\n");

444                         } else {
445                                 filled = 1;
446                                 f_print(fout, "\t\t%s", proc->args.argname);
447                                 f_print(fout, " ");
448                                 pvname(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} %s;\n", ARG);

457                 if (mtflag) {
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                                 pvname(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} %s;\n", RESULT);
474                         f_print(fout, "\tbool_t %s;\n", RETVAL);

476                 } else
477                         f_print(fout, "\tchar *%s;\n", RESULT);

479                 f_print(fout, "\txdrproc_t _xdr_%s, _xdr_%s;\n", ARG, RESULT);
514                 if (Cflag) {
515                         f_print(fout, "\txdrproc_t _xdr_%s, _xdr_%s;\n",
516                             ARG, RESULT);
480                 if (mtflag)
481                         f_print(fout, "\tbool_t "
482                             "(*%s)(char *, void *, struct svc_req *);\n",
518                             f_print(fout,
519 "\tbool_t (*%s)(char *, void *, struct svc_req *);\n",
483                             ROUTINE);
484                 else
485                         f_print(fout, "\tchar *(*%s)"
486                             "(char *, struct svc_req *);\n",
522                             f_print(fout,
523 "\tchar *(*%s)(char *, struct svc_req *);\n",
487                             ROUTINE);
525                 } else {
526                         f_print(fout,
```



```
527                             "\tbool_t (*_xdr_%s)(), (*_xdr_%s)();\n",
528                             ARG, RESULT);
529                         if (mtflag)
530                                 f_print(fout, "\tbool_t (*%s)();\n", ROUTINE);
531                         else
532                                 f_print(fout, "\tchar *(*%s)();\n", ROUTINE);
533                 }
488                 f_print(fout, "\n");

490                 if (timerflag) {
491                         if (mtflag)
492                                 f_print(fout,
493 "\t(void) mutex_lock(&_svcstate_lock);\n");

495                         f_print(fout, "\t_rpcsvccount++;\n");
496                         if (mtflag)
497                                 f_print(fout,
498 "\t(void) mutex_unlock(&_svcstate_lock);\n");
499                 }

501                 f_print(fout, "\tswitch (%s->rq_proc) {\n", RQSTP);
502                 if (!nullproc(vp->procs)) {
503                         f_print(fout, "\tcase NULLPROC:\n");
504                         f_print(fout,
505 "\t\t(void) svc_sendreply(%s,\n\t\t    (xdrproc_t)xdr_void, NULL);\n",
551                             Cflag ?
552 "\t\t(void) svc_sendreply(%s,\n\t\t    (xdrproc_t)xdr_void, NULL);\n" :
553 "\t\t(void) svc_sendreply(%s, xdr_void,\n\t\t\tNULL);\n",
506                             TRANSP);
507                         print_return("\t\t");
508                         f_print(fout, "\n");
509                 }
510                 for (proc = vp->procs; proc != NULL; proc = proc->next) {
511                         f_print(fout, "\tcase %s:\n", proc->proc_name);
512                         if (proc->arg_num < 2) { /* single argument */
513                                 p_xdrfunc(ARG, proc->args.decls->decl.type);
514                         } else {
515                                 p_xdrfunc(ARG, proc->args.argname);
516                         }

518                         if (streq(proc->res_type, "oneway")) {
519                                 /* One-way call */
520                                 f_print(fout, "\t\t_xdr_%s = NULL;\n", RESULT);
521                         } else {
522                                 p_xdrfunc(RESULT, proc->res_type);
523                         }
572                         if (Cflag) {
524                         if (mtflag) {
525                                 f_print(fout,
526                                     "\t\t%s = (bool_t (*) (char *,  "
527                                     "void *,  struct svc_req *))",
528                                     ROUTINE);
529                         } else {
530                                 f_print(fout,
531                                     "\t\t%s = (char *(*)(char *, "
532                                     "struct svc_req *))",
533                                     ROUTINE);
534                         }
584                         } else {
585                                 if (mtflag) {
586                                         f_print(fout,
587                                             "\t\t%s = (bool_t (*)())",
588                                             ROUTINE);
589                                 } else {
590                                         f_print(fout, "\t\t%s = (char *(*)())",
591                                             ROUTINE);
```

```
592                                 }
593                         }

536                         f_print(fout, "\n\t\t    ");
537                         if (newstyle) { /* new style: calls internal routine */
538                                 f_print(fout, "_");
539                         }
540                         if (!newstyle)
599                         if ((Cflag || mtflag) && !newstyle)
541                                 pvname_svc(proc->proc_name, vp->vers_num);
542                         else
543                                 pvname(proc->proc_name, vp->vers_num);
544                         f_print(fout, ";\n");
545                         f_print(fout, "\t\tbreak;\n\n");
546                 }
547                 f_print(fout, "\tdefault:\n");
548                 printerr("noproc", TRANSP);
549                 print_return("\t\t");
550                 f_print(fout, "\t}\n");

552                 f_print(fout,
553                     "\t(void) memset((char *)&%s, 0, sizeof (%s));\n",
554                     ARG, ARG);
555                 printif("getargs", TRANSP, "(caddr_t)&", ARG);
556                 printerr("decode", TRANSP);
557                 print_return("\t\t");
558                 f_print(fout, "\t}\n");

560                 if (!mtflag)
620                         if (Cflag)
561                         f_print(fout,
562                             "\t%s = (*%s)((char *)&%s, %s);\n",
563                             RESULT, ROUTINE, ARG, RQSTP);
564                 else
625                         f_print(fout, "\t%s = (*%s)(&%s, %s);\n",
626                             RESULT, ROUTINE, ARG, RQSTP);
627                 else
628                         if (Cflag)
565                         f_print(fout,
566                             "\t%s = (bool_t)(*%s)"
567                             "((char *)&%s, (void *)&%s, %s);\n",
630 "\t%s = (bool_t)(*%s)((char *)&%s, (void *)&%s, %s);\n",
568                             RETVAL, ROUTINE, ARG, RESULT, RQSTP);
632                         else
633                                 f_print(fout,
634 "\t%s = (bool_t)(*%s)(&%s, &%s, %s);\n",
635                                 RETVAL, ROUTINE, ARG, RESULT, RQSTP);




572                 if (mtflag)
573                         f_print(fout,
574 "\tif (_xdr_%s && %s > 0 &&\n"
575 "\t    !svc_sendreply(%s, _xdr_%s, (char *)&%s)) {\n",
576                             RESULT, RETVAL, TRANSP, RESULT, RESULT);
577                 else
578                         f_print(fout,
579 "\tif (_xdr_%s && %s != NULL &&\n"
580 "\t    !svc_sendreply(%s, _xdr_%s, %s)) {\n",
581                             RESULT, RESULT, TRANSP, RESULT, RESULT);

583                 printerr("systemerr", TRANSP);
584                 f_print(fout, "\t}\n");

586                 printif("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}\n");
590                 /* print out free routine */
591                 if (mtflag) {
592                         f_print(fout, "\tif (_xdr_%s != NULL) {\n", RESULT);
593                         f_print(fout, "\t\tif (!");

595                         pvname(def->def_name, vp->vers_num);
596                         f_print(fout, "_freeresult(%s, _xdr_%s,\n",
597                             TRANSP, RESULT);
598                         f_print(fout, "\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}\n");
603                 };
604                 print_return("\t");
605                 f_print(fout, "}\n");
606         }
607 }
```

*_____unchanged_portion_omitted_*

```
734 /*
735  * Write the RPC_MSGOUT function
736  *
737  * Note that while we define RPC_MSGOUT to be printf-like, all existing
738  * calls are of the form "%s","<msg>" and this implementation assumes that
739  * trivial case.  If in the future it's desirable to generate richer calls
740  * this implementation can change to match.  This way we don't (yet) have
741  * to introduce varargs into the generated code.
742  */
743 static void
744 write_msg_out(void)
745 {
746         f_print(fout, "\n");
747         f_print(fout, "#if\tdefined(RPC_MSGOUT)\n");
816         if (!Cflag) {
817                 f_print(fout, "extern void RPC_MSGOUT();\n");
818         } else {
748         f_print(fout, "extern void RPC_MSGOUT(const char *, ...);\n");
820         }
749         f_print(fout, "#else\t/* defined(RPC_MSGOUT) */\n");
750         f_print(fout, "static ");
823         if (!Cflag) {
824                 f_print(fout, "void\nRPC_MSGOUT(fmt, msg)\n");
825                 f_print(fout, "\tchar *fmt;\n");
826                 f_print(fout, "\tchar *msg;\n");
827         } else {
751         f_print(fout, "void\nRPC_MSGOUT(const char *fmt, char *msg)\n");
829         }
752         f_print(fout, "{\n");
753         f_print(fout, "#ifdef RPC_SVC_FG\n");
754         if (inetdflag || pmflag)
755                 f_print(fout, "\tif (_rpcpmstart)\n");
756         f_print(fout, "\t\tsyslog(LOG_ERR, fmt, msg);\n");
757         f_print(fout, "\telse {\n");
758         f_print(fout, "\t\t(void) fprintf(stderr, fmt, msg);\n");
759         f_print(fout, "\t\t(void) putc('\\n', stderr);\n");
760         f_print(fout, "\t}\n");
761         f_print(fout, "#else\n");
762         f_print(fout, "\tsyslog(LOG_ERR, fmt, msg);\n");
763         f_print(fout, "#endif\n");
764         f_print(fout, "}\n");
765         f_print(fout, "#endif\t/* defined(RPC_MSGOUT) */\n");
766 }
```

```
768 /*
769  * Write the timeout function
770  */
771 static void
772 write_timeout_func(void)
773 {
774         if (!timerflag)
775                 return;

777         f_print(fout, "\n");
778         if (mtflag) {
779                 f_print(fout, "/*ARGSUSED*/\n");
780                 f_print(fout, "static void *\n");
859                 if (!Cflag) {
860                         f_print(fout, "closedown(arg)\n");
861                         f_print(fout, "\tvoid *arg;\n");
862                 } else
781                 f_print(fout, "closedown(void *arg)\n");
782                 f_print(fout, "{\n");
783                 f_print(fout, "\t/*CONSTCOND*/\n");
784                 f_print(fout, "\twhile (1) {\n");
785                 f_print(fout, "\t\t(void) sleep(_RPCSVC_CLOSEDOWN/2);\n\n");
786                 f_print(fout,
787 "\t\tif (mutex_trylock(&_svcstate_lock) != 0)\n");
788                 f_print(fout, "\t\t\tcontinue;\n\n");
789                 f_print(fout,
790 "\t\tif (_rpcsvcstate == _IDLE && _rpcsvccount == 0) {\n");
791                 if (tirpcflag) {
792                         f_print(fout, "\t\t\tint size;\n");
793                 } else {
794                         f_print(fout, "\t\t\textern fd_set svc_fdset;\n");
795                         f_print(fout, "\t\t\tstatic int size;\n");
796                 }
797                 f_print(fout, "\t\t\tint i, openfd = 0;\n\n");
798                 if (tirpcflag) {
799                         f_print(fout, "\t\t\tsize = svc_max_pollfd;\n");
800                 } else {
801                         f_print(fout, "\t\t\tif (size == 0) {\n");
802                         f_print(fout, "\t\t\t\tsize = getdtablesize();\n");
803                         f_print(fout, "\t\t\t}\n");
804                 }
805                 f_print(fout,
806 "\t\t\tfor (i = 0; i < size && openfd < 2; i++)\n");
807                 if (tirpcflag) {
808                         f_print(fout, "\t\t\t\tif (svc_pollfd[i].fd >= 0)\n");
809                 } else {
810                         f_print(fout, "\t\t\t\tif (FD_ISSET(i, &svc_fdset))\n");
811                 }
812                 f_print(fout, "\t\t\t\t\topenfd++;\n");
813                 f_print(fout, "\t\t\t\tif (openfd <= 1)\n");
814                 f_print(fout, "\t\t\t\t\texit(0);\n");
815                 f_print(fout, "\t\t\t} else\n");
816                 f_print(fout, "\t\t\t\t_rpcsvcstate = _IDLE;\n\n");
817                 f_print(fout, "\t\t(void) mutex_unlock(&_svcstate_lock);\n");
818                 f_print(fout, "\t}\n");
819                 f_print(fout, "}\n");
820                 return;
821         }

823         f_print(fout, "static void\n");
906         if (!Cflag) {
907                 f_print(fout, "closedown(sig)\n");
908                 f_print(fout, "\tint sig;\n");
909         } else
824         f_print(fout, "closedown(int sig)\n");
```



```
825         f_print(fout, "{\n");
826         f_print(fout, "\tif (_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\tstatic int size;\n");
832         }
833         f_print(fout, "\t\tint i, openfd = 0;\n\n");
834         if (tirpcflag) {
835                 f_print(fout, "\t\tsize = 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\t\topenfd++;\n");
849         f_print(fout, "\t\tif (openfd <= 1)\n");
850         f_print(fout, "\t\t\texit(0);\n");
851         f_print(fout, "\t} else\n");
852         f_print(fout, "\t\t_rpcsvcstate = _IDLE;\n\n");

854         f_print(fout, "\t(void) signal(SIGALRM, (SIG_PF) closedown);\n");
940         f_print(fout, "\t(void) signal(SIGALRM, %s closedown);\n",
941             Cflag? "(SIG_PF)" : "(void(*)())");
855         f_print(fout, "\t(void) alarm(_RPCSVC_CLOSEDOWN/2);\n");
856         f_print(fout, "}\n");
857 }

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;

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\tSVCXPRT *%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  *  f_print(fout, "\t\textern char *getenv();\n");
887  */
888         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");

 894         if (timerflag) {
 895                 f_print(fout, "\t\t/* started from inetd */\n");
 896                 f_print(fout, "\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\t", "cannot get transport info");
 901         if (timerflag)
 902                 f_print(fout,
 903                     "\n\t\t\tpmclose = (t_getstate(0) != T_DATAXFER);\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\tif (!svc_reg(%s, %s, %s,\n",
 921                             TRANSP, def->def_name, vp->vers_name);
 922                         f_print(fout, "\t\t    ");
 923                         pvname(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    0, NULL) != 0) {\n");
 937                         print_err_message("\t\t\t\t",
 938                             "cannot create closedown thread");
 939                         f_print(fout, "\t\t\t\texit(1);\n");
 940                         f_print(fout, "\t\t\t}\n");
 941                 } else {
 942                         f_print(fout,
 943 "\t\t\t(void) signal(SIGALRM, (SIG_PF) closedown);\n");
1030 "\t\t\t(void) signal(SIGALRM, %s closedown);\n",
1031                             Cflag? "(SIG_PF)" : "(void(*)())");
 944                         f_print(fout,
 945 "\t\t\t(void) alarm(_RPCSVC_CLOSEDOWN/2);\n");
 946                 }
 947                 f_print(fout, "\t\t}\n");
 948         }
 949         f_print(fout, "\t\tsvc_run();\n");
 950         f_print(fout, "\t\texit(1);\n");
 951         f_print(fout, "\t\t/* NOTREACHED */\n");
 952         f_print(fout, "\t}");
```



```
 953 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    4924 Sun Aug  3 11:09:17 2014
new/usr/src/cmd/rpcgen/rpc_tblout.c
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * rpc_tblout.c, Dispatch table outputter for the RPC protocol compiler
 */
#include <stdio.h>
#include <string.h>
#include "rpc_parse.h"
#include "rpc_util.h"

extern int nullproc(proc_list *);

static void write_table(definition *);
static void printit(char *, char *);

#define TABSIZE         8
#define TABCOUNT        5
#define TABSTOP         (TABSIZE*TABCOUNT)

static char tabstr[TABCOUNT+1] = "\t\t\t\t\t";

static char tbl_hdr[] = "struct rpcgen_table %s_table[] = {\n";
static char tbl_end[] = "};\n";
```



```
static char null_entry_b[] = "\n\t(char *(*)())0,\n"
                        " \t(xdrproc_t)xdr_void,\t\t\t0,\n"
                        " \t(xdrproc_t)xdr_void,\t\t\t0,\n";

static char null_entry[] = "\n\t(void *(*)())0,\n"
                        " \t(xdrproc_t)xdr_void,\t\t\t0,\n"
                        " \t(xdrproc_t)xdr_void,\t\t\t0,\n";


static char tbl_nproc[] = "int %s_nproc =\n\tsizeof(%s_table)"
                                "/sizeof(%s_table[0]);\n\n";

void
write_tables(void)
{
        list *l;
        definition *def;

        f_print(fout, "\n");
        for (l = defined; l != NULL; l = l->next) {
                def = (definition *)l->val;
                if (def->def_kind == DEF_PROGRAM) {
                        write_table(def);
                }
        }
}

static void
write_table(definition *def)
{
        version_list *vp;
        proc_list *proc;
        int current;
        int expected;
        char progvers[100];
        int warning;

        for (vp = def->def.pr.versions; vp != NULL; vp = vp->next) {
                warning = 0;
                (void) snprintf(progvers, sizeof (progvers), "%s_%s",
                    locase(def->def_name), vp->vers_num);
                /* print the table header */
                f_print(fout, tbl_hdr, progvers);

                if (nullproc(vp->procs)) {
                        expected = 0;
                } else {
                        expected = 1;
                        if (tirpcflag)
                                f_print(fout, null_entry);
                        else
                                f_print(fout, null_entry_b);
                }
                for (proc = vp->procs; proc != NULL; proc = proc->next) {
                        current = atoi(proc->proc_num);
                        if (current != expected++) {
                                f_print(fout,
                        "\n/*\n * WARNING: table out of order\n */\n");
                                if (warning == 0) {
                                        f_print(stderr,
                                    "WARNING %s table is out of order\n",
                                            progvers);
                                        warning = 1;
                                        nonfatalerrors = 1;
                                }
                                expected = current + 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(");
135
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
147                         /* argument info */
148                         if (proc->arg_num > 1)
149                                 printit(NULL, proc->args.argname);
150                         else
151                         /* do we have to do something special for newstyle */
152                                 printit(proc->args.decls->decl.prefix,
153                                     proc->args.decls->decl.type);
154                         /* result info */
155                         printit(proc->res_prefix, proc->res_type);
156                 }
157
158                 /* print the table trailer */
159                 f_print(fout, tbl_end);
160                 f_print(fout, tbl_nproc, progvers, progvers, progvers);
161         }
162 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    4722 Sun Aug  3 11:09:17 2014
new/usr/src/cmd/rpcgen/rpc_util.h
rpcgen should only produce ANSI code
**********************************************************
```

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

/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */
/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

#ifndef _RPC_UTIL_H
#define _RPC_UTIL_H

#include <sys/types.h>
#include <stdlib.h>
#include "rpc_scan.h"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * rpc_util.h, Useful definitions for the RPC protocol compiler
 */


/* Current version number of rpcgen. */
#define RPCGEN_MAJOR 1
#define RPCGEN_MINOR 1

#define f_print (void) fprintf
```



```
struct list {
        definition *val;
        struct list *next;
};
```

*_____unchanged_portion_omitted_*

```
#define PUT 1
#define GET 2

/*
 * Global variables
 */
#define MAXLINESIZE 1024
extern char curline[MAXLINESIZE];
extern char *where;
extern int linenum;

extern char *infilename;
extern FILE *fout;
extern FILE *fin;

extern list *defined;

extern bas_type *typ_list_h;
extern bas_type *typ_list_t;
extern xdrfunc *xdrfunc_head, *xdrfunc_tail;

/*
 * All the option flags
 */
extern int inetdflag;
extern int pmflag;
extern int tblflag;
extern int logflag;
extern int newstyle;
extern int Cflag;       /* ANSI-C/C++ flag */
extern int CCflag;      /* C++ flag */
extern int tirpcflag;   /* flag for generating tirpc code */
extern int inlinelen;   /* if this is 0, then do not generate inline code */
extern int mtflag;
extern int mtauto;
extern int rflag;

/*
 * Other flags related with inetd jumpstart.
 */
extern int indefinitewait;
extern int exitnow;
extern int timerflag;

extern int nonfatalerrors;

extern pid_t childpid;

/*
 * rpc_util routines
 */
extern void storeval(list **, definition *);

#define STOREVAL(list, item)    \
        storeval(list, item)

extern definition *findval(list *, char *, int (*)());

#define FINDVAL(list, item, finder) \
        findval(list, item, finder)
```

```
150 extern char *fixtype(char *);
151 extern char *stringfix(char *);
152 extern char *locase(char *);
153 extern void pvname_svc(char *, char *);
154 extern void pvname(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_nettype_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_

```
**********************************************************
   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 '\" 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
```



```
  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]
```

```
.fi

.LP
.nf
\fBrpcgen\fR [\fB-n\fR \fInetid\fR] [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBrpcgen\fR utility is a tool that generates C code to implement an
\fBRPC\fR protocol. The input to \fBrpcgen\fR is a language similar to C known
as \fBRPC\fR Language (Remote Procedure Call Language).
.sp
.LP
The \fBrpcgen\fR utility is normally used as in the first synopsis where it
takes an input file and generates four output files. If the \fIinfile\fR is
named \fBproto.x\fR, then \fBrpcgen\fR generates a header in \fBproto.h\fR,
\fBXDR\fR routines in \fBproto_xdr.c\fR, server-side stubs in
\fBproto_svc.c\fR, and client-side stubs in \fBproto_clnt.c\fR. With the
\fB-T\fR option, it also generates the \fBRPC\fR dispatch table in
\fBproto_tbl.i\fR.
.sp
.LP
\fBrpcgen\fR can also generate sample client and server files that can be
customized to suit a particular application. The \fB-Sc\fR, \fB-Ss\fR, and
\fB-Sm\fR options generate sample client, server and makefile, respectively.
The \fB-a\fR option generates all files, including sample files. If the infile
is \fBproto.x\fR, then the client side sample file is written to
\fBproto_client.c\fR, the server side sample file to \fBproto_server.c\fR and
the sample makefile to \fBmakefile.proto\fR.
.sp
.LP
The server created can be started both by the port monitors (for example,
.Xr inetd 1M
or
.Xr listen 1M )
or by itself. When it is started by a port
\fBinetd\fR or \fBlisten\fR) or by itself. When it is started by a port
monitor, it creates servers only for the transport for which the file
descriptor 0 was passed. The name of the transport must be specified by
setting up the environment variable
.Ev PM_TRANSPORT .
When the server
generated by
.Nm
is executed, it creates server handles for all the
transports specified in the
.Ev NETPATH
environment variable, or if it is
descriptor \fB0\fR was passed. The name of the transport must be specified by
setting up the environment variable \fBPM_TRANSPORT\fR. When the server
generated by \fBrpcgen\fR is executed, it creates server handles for all the
transports specified in the \fBNETPATH\fR environment variable, or if it is
unset, it creates server handles for all the visible transports from the
.Pa /etc/netconfig
file. Note: the transports are chosen at run time and not
\fB/etc/netconfig\fR file. Note: the transports are chosen at run time and not
at compile time. When the server is self-started, it backgrounds itself by
default. A special define symbol
.Dv RPC_SVC_FG
can be used to run the server process in foreground.
.Lp
default. A special define symbol \fBRPC_SVC_FG\fR can be used to run the server
process in foreground.
.sp
```



```
.LP
The second synopsis provides special features which allow for the creation of
more sophisticated RPC servers. These features include support for
user-provided
.Li #defines
and RPC dispatch tables. The entries in the
RPC dispatch table contain:
.Bl -bullet -offset indent
.It
more sophisticated \fBRPC\fR servers. These features include support for
user-provided \fB#defines\fR and \fBRPC\fR dispatch tables. The entries in the
\fBRPC\fR dispatch table contain:
.RS +4
.TP
.ie t \(bu
.el o
pointers to the service routine corresponding to that procedure
.It
.RE
.RS +4
.TP
.ie t \(bu
.el o
a pointer to the input and output arguments
.It
.RE
.RS +4
.TP
.ie t \(bu
.el o
the size of these routines
.El
.Lp
.RE
.sp
.LP
A server can use the dispatch table to check authorization and then to execute
the service routine. A client library can use the dispatch table to deal with
the details of storage management and XDR data conversion.
.Lp
the details of storage management and \fBXDR\fR data conversion.
.sp
.LP
The other three synopses shown above are used when one does not want to
generate all the output files, but only a particular one. See the
.Sx EXAMPLES
section below for examples of
.Nm
usage. When
.Nm
is executed with the
.Fl s
option, it creates servers for that particular class of
transports. When executed with the
.Fl n
option, it creates a server for the
transport specified by
.Ar netid .
If
.Ar infile
is not specified,
.Nm
accepts the standard input.
.Lp
generate all the output files, but only a particular one. See the EXAMPLES
section below for examples of \fBrpcgen\fR usage. When \fBrpcgen\fR is executed
```

```
111 with the \fB-s\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 \fInetid\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 .Ic 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
```

```
.RS 18n
Size at which to start generating inline code. This option is useful for
optimization. The default
.Ar size
is 5.
.
.It Fl I
Compiles support for
.Xr inetd 1M
in the server side stubs. Such servers can
be self-started or can be started by
.Xr inetd 3C .
When the server is
optimization. The default \fIsize\fR is 5.
.RE

.sp
.ne 2
.na
\fB\fB-I\fR\fR
.ad
.RS 18n
Compiles support for \fBinetd\fR(1M) in the server side stubs. Such servers can
be self-started or can be started by \fBinetd\fR. When the server is
self-started, it backgrounds itself by default. A special define symbol
.Dv RPC_SVC_FG
can be used to run the server process in foreground, or the
user can simply compile without the
.Fl I
option.
.Lp
If there are no pending client requests, the
.Xr inetd 1M
servers exit after 120
seconds (default). The default can be changed with the
.Fl -K
option. All of
the error messages for
.Xr inetd 1M
servers are always logged with
.Xr syslog 3C .
.Lp
.Em Note:
This option is supported for backward compatibility only. It should
always be used in conjunction with the
.Fl b
option which generates backward
compatibility code. By default (that is, when
.Fl b
is not specified),
.Nm
generates servers that can be invoked through portmonitors.
.
.It Fl K Ar seconds
By default, services created using
.Nm
and invoked through port
\fBRPC_SVC_FG\fR can be used to run the server process in foreground, or the
user can simply compile without the \fB-I\fR option.
.sp
If there are no pending client requests, the \fBinetd\fR servers exit after 120
seconds (default). The default can be changed with the \fB-K\fR option. All of
the error messages for \fBinetd\fR servers are always logged with
\fBsyslog\fR(3C).
.sp
\fBNote:\fR This option is supported for backward compatibility only. It should
```



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

.sp
.ne 2
.na
\fB\fB-K\fR \fIseconds\fR\fR
.ad
.RS 18n
By default, services created using \fBrpcgen\fR and invoked through port
monitors wait 120 seconds after servicing a request before exiting. That
interval can be changed using the
.Fl K
flag. To create a server that exits
immediately upon servicing a request, use
.Fl K Li 0 .
To create a server
that never exits, the appropriate argument is
.Fl K Li \(mi1 .
.Lp
When monitoring for a server, some portmonitors, like
.Xr listen 1M ,
.Em always
spawn a new process in response to a service request. If it is
interval can be changed using the \fB-K\fR flag. To create a server that exits
immediately upon servicing a request, use \fB-K\fR \fB0\fR. To create a server
that never exits, the appropriate argument is \fB-K\fR \fB\(mi1\fR\&.
.sp
When monitoring for a server, some portmonitors, like \fBlisten\fR(1M),
\fBalways\fR spawn a new process in response to a service request. If it is
known that a server are used with such a monitor, the server should exit
immediately on completion. For such servers,
.Nm
should be used with
.Fl K Li 0 .
.
.It Fl l
immediately on completion. For such servers, \fBrpcgen\fR should be used with
\fB-K\fR \fB0\fR.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 18n
Compiles into client-side stubs.
.
.It Fl L
When the servers are started in foreground, uses
.Xr syslog 3C
to log the server errors instead of printing them on the standard error.
.
.It Fl m
.RE

.sp
.ne 2
.na
\fB\fB-L\fR\fR
.ad
.RS 18n
When the servers are started in foreground, uses \fBsyslog\fR(3C) to log the
```

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

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

.sp
.ne 2
.na
\fB\fB-M\fR\fR
.ad
.RS 18n
Generates multithread-safe stubs for passing arguments and results between
code generated by
.Nm rpcgen
and user written code. This option is useful for
\fBrpcgen\fR-generated code and user written code. This option is useful for
users who want to use threads in their code.
.
.It Fl N
.RE

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

.sp
.ne 2
.na
\fB\fB-n\fR \fInetid\fR\fR
.ad
.RS 18n
Compiles into server-side stubs for the transport specified by \fInetid\fR.
There should be an entry for \fInetid\fR in the \fBnetconfig\fR database. This
```



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

.sp
.ne 2
.na
\fB\fB-o\fR \fIoutfile\fR\fR
.ad
.RS 18n
Specifies the name of the output file. If none is specified, standard output is
used
.Po
.Fl c , h , l , m , n , s , "Sc" , "Sm" , "Ss" ,
and
.Fl t
modes only
.Pc .
.
.It Fl s Ar netttype
used (\fB-c\fR, \fB-h\fR, \fB-l\fR, \fB-m\fR, \fB-n\fR, \fB-s\fR, \fB-Sc\fR,
\fB-Sm\fR, \fB-Ss\fR, and \fB-t\fR modes only).
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fInettype\fR\fR
.ad
.RS 18n
Compiles into server-side stubs for all the transports belonging to the class
.Ar nettype .
The supported classes are
.Sy netpath ,
.Sy visible ,
.Sy circuit_n ,
.Sy circuit_v ,
.Sy datagram_n ,
.Sy datagram_v ,
.Sy tcp ,
and
.Sy udp
(see
.Xr rpc 3NSL
for the meanings associated with
these classes). This option can be specified more than once.
.Em Note:
The transports are chosen at run time and not at compile time.
.
.It Fl "Sc"
\fInettype\fR. The supported classes are \fBnetpath\fR, \fBvisible\fR,
\fBcircuit_n\fR, \fBcircuit_v\fR, \fBdatagram_n\fR, \fBdatagram_v\fR,
\fBtcp\fR, and \fBudp\fR (see \fBrpc\fR(3NSL) for the meanings associated with
these classes). This option can be specified more than once. \fBNote:\fR The
transports are chosen at run time and not at compile time.
.RE

.sp
.ne 2
.na
\fB\fB-Sc\fR\fR
.ad
.RS 18n
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 \fRGenerating 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
```

```
example% \fBrpcgen -T prot.x\fR
.fi
.in -2
.sp

.sp
.LP
generates all the five files: \fBprot.h\fR, \fBprot_clnt.c\fR,
\fBprot_svc.c\fR, \fBprot_xdr.c\fR, and \fBprot_tbl.i\fR.

.LP
\fBExample 2 \fRSending headers to standard output
.sp
.LP
```

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

```
.Lp
.Dl example% rpcgen -h prot.x
.
.Ss Example 3 Sending a test version
To send the test version of the
.Fl DTEST ,
server side stubs for all the
transport belonging to the class
.Sy datagram_n
to standard output, use:
.Lp
.Dl example% rpcgen -s datagram_n -DTEST prot.x
.
.Ss Example 4 Creating server side stubs
To create the server side stubs for the transport indicated by
.Ar netid
.Sy tcp ,
use:
.Lp
.Dl example% rpcgen -n tcp -o prot_svc.c prot.x
.Sh SEE ALSO
.Xr inetd 1M ,
.Xr listen 1M ,
.Xr rpc 3NSL ,
.Xr rpc_control 3NSL ,
.Xr rpc_svc_calls 3NSL ,
.Xr syslog 3C ,
.Xr netconfig 4 ,
.Xr attributes 5
.Rs
.%B ONC+ Developer\&’s Guide
.Re

.sp
.in +2
.nf
example% \fBrpcgen -h prot.x\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRSending a test version
.sp
.LP
To send the test version of the \fB-DTEST\fR, server side stubs for all the
transport belonging to the class \fBdatagram_n\fR to standard output, use:

.sp
.in +2
```



```
.nf
example% \fBrpcgen -s datagram_n -DTEST prot.x\fR
.fi
.in -2
.sp

.LP
\fBExample 4 \fRCreating server side stubs
.sp
.LP
To create the server side stubs for the transport indicated by \fInetid\fR
\fBtcp\fR, use:

.sp
.in +2
.nf
example% \fBrpcgen -n tcp -o prot_svc.c prot.x\fR
.fi
.in -2
.sp

.SH EXIT STATUS
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 6n
Successful operation.
.RE

.sp
.ne 2
.na
\fB\fB>0\fR\fR
.ad
.RS 6n
An error occurred.
.RE

.SH SEE ALSO
.sp
.LP
\fBinetd\fR(1M), \fBlisten\fR(1M), \fBrpc\fR(3NSL), \fBrpc_control\fR(3NSL),
\fBrpc_svc_calls\fR(3NSL), \fBsyslog\fR(3C), \fBnetconfig\fR(4),
\fBattributes\fR(5)
.sp
.LP
The \fBrpcgen\fR chapter in the \fIONC+ Developer\&’s Guide\fR manual.
```