IBM i
7.2

Programming
Using Sun TI-RPC to develop distributed applications

Note

Before using this information and the product it supports, read the information in “Notices” on page 33.

Contents

Using Sun TI-RPC to develop distributed applications

Remote procedure call (RPC) provides a high-level paradigm, which allows distributed applications to communicate with one another.

Sun Microsystems developed open networking computers (ONC) RPC to easily separate and distribute a client application from a server mechanism. Transport independent remote procedure call (TI-RPC) or ONC+ RPC is the latest version of RPC to be released. By providing a method for abstracting the underlying protocol used at the network layer, TI-RPC can provide a more seamless transition from one protocol to another.

For detailed information about designing, implementing, and maintaining distributed applications by using TI-RPC, refer to the *ONC+ Developer's Guide* by Sun Microsystems, Inc. on the Sun Microsystems Documentation Web site.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 30.

PDF file for Using Sun TI-RPC to develop distributed applications

You can view and print a PDF file of this information.

To view or download the PDF version of this document, select Using Sun TI-RPC to develop distributed applications (about 330 KB).

Other information

For detailed information about designing, implementing, and maintaining distributed applications by using TI-RPC, refer to the *ONC+ Developer's Guide* by Sun Microsystems, Inc. on the Sun Microsystems Documentation Web site.

For more information about TI-RPC service application programming interfaces (APIs), see the Application programming interfaces.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html).

Using the rpcbind daemon

When a client wants to connect to a remote procedure call (RPC) service, it contacts the rpcbind daemon and requests the address of the service. In this way, addresses can be dynamic and the client does not need to know which port the service is waiting on.

Services need to be registered with the rpcbind daemon in order to be useful. If the rpcbind daemon is inactive, the services are unable to start, and the clients are not able to find any of the services.

Ensuring that the rpcbind daemon is running on i5/OS

To use the transport independent remote procedure call (TI-RPC) application programming interfaces (APIs), you need to ensure that the rpcbind daemon job (QNFSRPCD) is running on the i5/OS operating system.

To ensure that the rpcbind daemon is running, complete the following steps:

1. On the i5/OS command line, type: WRKACTJOB
2. Look in the subsystem QSYSWRK for the existence of the following job:

```
QNFSRPCD The rpcbind daemon
```

Starting and ending the rpcbind daemon on i5/OS

You need to start the rpcbind daemon on i5/OS operating system if it is not already running. The rpcbind daemon (RPCBIND) command starts the rpcbind daemon job (QNFSRPCD).

Starting the rpcbind daemon job

To start the rpcbind daemon job, type the following command:

```
RPCBIND RTVRPCREG(*YES)
```

Note: An optional parameter for this command is RTVRPCREG, which specifies whether to retrieve previously recorded registration information when the rpcbind daemon is started. The default for this parameter is *NO. Select *YES if you want the rpcbind daemon to retrieve registration information when it starts. For more information about the parameter and value descriptions for this command, refer to the online help text.

Ending the rpcbind daemon job

To end the rpcbind daemon job (QNFSRPCD), type the following command:

```
ENDRPCBIND
```

Using the rpcgen compiler

The RPCGEN command generates C code from an input file that is written in the remote procedure call language (RPCL). You can use the generated C code to implement an RPC protocol.

To use the rpcgen compiler on the i5/OS operating system, complete the following tasks:

1. Create your source input file in RPCL.

   For details about using the rpcgen compiler, refer to the Sun Microsystems Documentation Web site  .
2. Type the following command to run the rpcgen compiler on the i5/OS operating system:

   ```
   RPCGEN
   ```

   Note: To see the parameter and value descriptions for this command, refer to the online help text.
3. Use a C language compiler on the i5/OS operating system to compile the output from the rpcgen compiler.

   Note: If you are using the Integrated Language Environment® (ILE) C compiler on the i5/OS operating system, you need to store the output files as source members.

Using the network selection mechanism

The network selection mechanism allows you to choose the transport on which an application should run.

The /etc/netconfig file is a database that lists the transports that are available to the host and identifies them by type. Transports are available in the /etc/netconfig file in the specified order in which they appear.

If you want to access the /etc/netconfig file on the i5/OS operating system, complete the following steps:

1. Open System i® Navigator.
2. Expand **Network** > **Servers**.
3. Select the i5/OS server.
4. Right-click **RPC** and from the menu, select **Properties**.
5. Select the **RPC Transports** tab.

   Note: You must have *IOSYSCFG authority to view this information.

Related reference

Application programming interfaces

Using data conversion support

All transport independent remote procedure call (TI-RPC) application programming interfaces (APIs) are enabled for National Language Support (NLS) on the i5/OS operating system.

This support has been added to a list of eXternal Data Representation (XDR) functions. These functions allow data to be communicated between clients and services that are in different code pages. The system administrator maintains the /etc/rpcnls file to associate the code pages with a remote client. The XDR functions use the information in the /etc/rpcnls file to provide an implicit data conversion. The following XDR functions have implicit data conversion routines built in:

- `xdr_char() (single byte only)`
- `xdr_u_char() (single byte only)`
- `xdr_double_char (single and double byte)`
- `xdr_string() (single and double byte)`
- `xdr_wrapstring() (single and double byte)`

To access the /etc/rpcnls file on the i5/OS operating system in System i Navigator, complete the following steps:

1. Open System i Navigator.
2. Expand **Network** > **Servers**.
3. Select the i5/OS server.
4. Right-click **RPC** and from the menu, select **Properties**.
5. Select the **Data Conversion Support** tab.

   Note: You must have *IOSYSCFG authority to view this information.

Related reference

Application programming interfaces

Examples: Developing service applications based on TI-RPC code

Transport independent remote procedure call (TI-RPC) programming provides an effective method for developing distributed client-server based applications on the i5/OS operating system.

Related reference

Examples: Developing client applications based on TI-RPC code

To develop client applications on the i5/OS operating system, use these code examples as a guideline.

Example: TI-RPC simplified-level service API

This code example illustrates one of the simplified-level service APIs that are used in developing TI-RPC services.

In this code example, notice how each procedure is registered independently from the rest. At the simplified level, a service might have multiple procedures, but each one must be registered separately. If the service is unregistered, all of the procedures are unregistered at once. There is no method that is provided to unregister an individual procedure while leaving the remaining procedures intact.

This level is a good choice for services with a small number of procedures. It is also useful for prototyping a much larger service by using a limited number of the final procedures. As with all the service levels, the final call in the service should be to svc_run(), which goes into Select Wait (waiting for a connection from a client).

Note: By using the code example, you agree to the terms of the "Code license and disclaimer information" on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    bool_t rslt; /* return value for rpc_call() */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_UID, myapp_get_uid,
            xdr_wrapstring, xdr_u_int, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_UID");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
                PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_UID_STRING, myapp_get_uid_string,
            xdr_wrapstring, xdr_wrapstring, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_UID_STRING");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
                PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_SIZE, myapp_get_size,
            xdr_wrapstring, xdr_int, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_SIZE");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
```

```
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_MTIME, myapp_get_mtime,
            xdr_wrapstring, xdr_long, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_MTIME");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_MTIME_STRING, myapp_get_mtime_string,
            xdr_wrapstring, xdr_wrapstring, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_MTIME_STRING");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_CODEPAGE, myapp_get_codepage,
            xdr_wrapstring, xdr_u_short, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_CODEPAGE");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_OBJTYPE, myapp_get_objtype,
            xdr_wrapstring, xdr_wrapstring, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_OBJTYPE");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, GET_FILETYPE, myapp_get_filetype,
            xdr_wrapstring, xdr_wrapstring, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "GET_FILETYPE");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
            PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
```

```
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* (bool_t)rpc_reg(prognum, versnum, procnum, procname, */
    /*                 xdr_in, xdr_out, nettype)            */
    rslt = rpc_reg(PROGNUM, VERSNUM, END_SERVER, myapp_end_server,
            (xdrproc_t)xdr_void, (xdrproc_t)xdr_void, NETTYPE);

    /* check for errors calling rpc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling rpc_reg for %s\n", "END_SERVER");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
                PROGNUM, VERSNUM, NETTYPE);
        /* clean up before exiting */
        svc_unreg(PROGNUM, VERSNUM);
        return 1;
        }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned.  ERROR has occurred.\n");
    fprintf(stderr, "errno: %d\n", errno);

    /* clean up by unregistering.  then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;

} /* end of main() */
```

Example: TI-RPC top-level service API

This code example illustrates a top-level service API used in developing TI-RPC services.

The development of a service is more complicated at the top level, because it requires the developer to write a dispatch routine. At this level, when a service request comes in, a dispatch routine is called. The dispatch routine must collect the arguments and call the correct local procedure, catch all errors and results, and return that information to the client. After a dispatch function is written, it can be readily copied and used in other services with only slight modifications.

The top, intermediate, and expert layers can use the same dispatch function without modification. In the following example, the dispatch function is bundled with the other local functions in this file. Both files need to be compiled and linked together before the service runs. The advantage of the top level over the other layers is the ability to specify the NETTYPE as a string, instead of using the network selection APIs. After calling the top-level API, the service is created, bound to the dispatch function, and registered with the rpcbind service.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"
int main(int argc, char *argv[]) {

    int num_svc; /* return value for the svc_create() API */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (int)svc_create(dispatch, prognum, versnum, nettype); */
    num_svc = svc_create(myapp_dispatch, PROGNUM, VERSNUM, NETTYPE);

    /* check for errors calling svc_create() */
    if (num_svc == 0) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.\n", "svc_create");
        fprintf(stderr, "PROG: %lu\nVERS: %lu\tNET: %s\n",
```

```
                PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned.  ERROR has occurred.\n");
    fprintf(stderr, "errno: %d\n", errno);

    /* clean up by unregistering.  then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;

} /* end of main() */
```

```
/* This is an example of the dispatch function */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <pwd.h>
#include <rpc/rpc.h>
#include <time.h>
#include "myapp.h"

char * myapp_get_uid(char *in) {

    u_int retval;              /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (u_int)-1;
        }
    else {
        retval = (u_int)(sbuf.st_uid);
        }

    return (char *)&retval;
}

char *myapp_get_uid_string(char *in) {

    char *retval;              /* return value for this procedure() */
    struct passwd *pbuf;       /* return value for getpwuid() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
        }
    else {

        pbuf = (struct passwd *)getpwuid((uid_t)(sbuf.st_uid));

        if (pbuf == NULL) {
            retval = (char *)NULL;
            }
        else {
            retval = (char *)(pbuf->pw_name);
            }
        }

    return (char *)&retval;
}

char * myapp_get_size(char *in) {
```

```
    int retval;                /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (int)-1;
        }
    else {
        retval = (int)(sbuf.st_size);
        }

    return (char *)&retval;
}

char * myapp_get_mtime(char *in) {

    long retval;               /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (long)-1;
        }
    else {
        retval = (long)(sbuf.st_mtime);
        }

    return (char *)&retval;

}

char *myapp_get_mtime_string(char *in) {

    char *retval;              /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
        }

    else {
        retval = (char *)ctime((time_t *)&(sbuf.st_mtime));
        }

    return (char *)&retval;
}

char * myapp_get_codepage(char *in) {

    u_short retval;            /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (u_short)-1;
    }

    else {
        retval = (u_short)(sbuf.st_codepage);
    }

    return (char *)&retval;
}


char *myapp_get_objtype(char *in) {
```

```
    char *retval;              /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        retval = (char *)NULL;
    }

    else {
        retval = (char *)(sbuf.st_objtype);
    }

    return (char *)&retval;

}

char *myapp_get_filetype(char *in) {

    char *result = NULL;       /* return value for this procedure() */
    struct stat sbuf;          /* data storage area for stat() */
    int stat_ret;              /* return value for stat() */
    char *file = *(char **)in; /* input value for stat() */

    /* (int)stat(filename, struct stat *) */
    stat_ret = stat(file, &sbuf);

    if (stat_ret == -1) {
        return (char *)NULL;
        }

    if (S_ISDIR(sbuf.st_mode)) {
        result = "Directory";
        }

    if (S_ISREG(sbuf.st_mode)) {
        result = "Regulare File";
        }

    if (S_ISLNK(sbuf.st_mode)) {
        result = "Symbolic Link";
        }

    if (S_ISSOCK(sbuf.st_mode)) {
        result = "Socket";
        }

    if (S_ISNATIVE(sbuf.st_mode)) {
        result = "System i Native Object";
        }

    if (S_ISFIFO(sbuf.st_mode)) {
        result = "FIFO";
        }

    if (S_ISCHR(sbuf.st_mode)) {
        result = "Character Special";
        }

    if (S_ISBLK(sbuf.st_mode)) {
        result = "Block Special";
        }

    return (char *)&result;

}

char * myapp_end_server(char *empty) {

    /* char *empty is not used      */
    /* function always returns NULL */

    svc_unreg(PROGNUM, VERSNUM);

    return (char *)NULL;

}
```

```
void myapp_dispatch(struct svc_req *request, SVCXPRT *svc) {

    union {
        /* all of the procedures take a string */
        /* if there were other procedures, it */
        /* might look like this:              */
        /* int set_codepage_arg               */
        char *  filename_arg;
    } argument;

    char *result;            /* pointer to returned data from proc */
    xdrproc_t xdr_argument;  /* decodes data from client call */
    xdrproc_t xdr_result;    /* encodes data to return to client */
    char *(*proc)(char *);   /* pointer to local procedure to call */

    switch (request->rq_proc) {

        case NULLPROC:
            /* a special case.  always return void */
                        (void)svc_sendreply((SVCXPRT *)svc,
                                            (xdrproc_t)xdr_void,
                                            (char *)NULL);
            return;

        case GET_UID:
            /* takes a string argument (filename) */
            /* returns an u_int (uid of file ownder) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_u_int;
            proc         = (char *(*)(char *))myapp_get_uid;
            break;

        case GET_UID_STRING:
            /* takes a string argument (filename) */
            /* returns a string (owner's name in string format) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_wrapstring;
            proc         = (char *(*)(char *))myapp_get_uid_string;
            break;

        case GET_SIZE:
            /* takes a string argument (filename) */
            /* returns an int (size of file in bytes) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_int;
            proc         = (char *(*)(char *))myapp_get_size;
            break;

        case GET_MTIME:
            /* takes a string argument (filename) */
            /* returns a long (time last modified) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_long;
            proc         = (char *(*)(char *))myapp_get_mtime;
            break;

        case GET_MTIME_STRING:
            /* takes a string argument (filename) */
            /* returns a string (time last modified, string format) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_wrapstring;
            proc         = (char *(*)(char *))myapp_get_mtime_string;
            break;

        case GET_CODEPAGE:
            /* takes a string argument (filename) */
            /* returns an u_short (codepage of file) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_u_short;
            proc         = (char *(*)(char *))myapp_get_codepage;
            break;

        case GET_OBJTYPE:
            /* takes a string argument (filename) */
            /* returns a string (object type) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_wrapstring;
            proc         = (char *(*)(char *))myapp_get_objtype;
            break;

        case GET_FILETYPE:
            /* takes a string argument (filename) */
```

```
            /* returns a string (file type) */
            xdr_argument = xdr_wrapstring;
            xdr_result   = xdr_wrapstring;
            proc         = (char *(*)(char *))myapp_get_filetype;
            break;

        case END_SERVER:
            /* takes no arguments */
            /* returns no data */
            /* unregisters service with local rpcbind daemon */
            xdr_argument = (xdrproc_t)xdr_void;
            xdr_result   = (xdrproc_t)xdr_void;
            proc         = (char *(*)(char *))myapp_end_server;
            break;

        default:
            /* fall through case.  return error to client */
            svcerr_noproc(svc);
            return;

    } /* end switch(request->rq_proc) */


    /* clear the argument */
    memset((char *)&argument, (int)0, sizeof(argument));

    /* decode argument from client using xdr_argument() */
    if (svc_getargs(svc, xdr_argument, (char *)&argument) == FALSE) {
        /* if svc_getargs() fails, return RPC_CANTDECODEARGS to client */
        svcerr_decode(svc);
        return;
    }

    /* call local procedure, passing in pointer to argument */
    result = (char *)(*proc)((char *)&argument);

    /* check first that result isn't NULL */
    /* try to send results back to client.  check for failure */
    if ((result != NULL) && (svc_sendreply(svc, xdr_result, result) == FALSE))
{
        /* send error message back to client */
        svcerr_systemerr(svc);
    }

    /* free the decoded argument's space */
    if (svc_freeargs(svc, xdr_argument, (char *)&argument) == FALSE) {
        /* if unable to free, print error and exit */
        (void)fprintf(stderr, "unable to free arguments\n");
        exit(1);
    }

} /* end of myapp_dispatch() */
```

Example: TI-RPC intermediate-level service API

This code example illustrates an intermediate-level service API that is used in developing TI-RPC services.

The dispatch function can be reused without any changes. The only difference between the top and intermediate levels is the use of the network selection APIs. This level also creates, binds, and registers the service with the rpcbind daemon.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    struct netconfig *nconf;  /* pointer to nettype data */
    SVCXPRT *svc;             /* pointer to service handle */

    /* unregister any existing copy of this service */
```

```
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (struct netconfig *)getnetconfigent(nettype) */
    nconf = getnetconfigent(NETTYPE);
    if (nconf == (struct netconfig *)NULL) {
        fprintf(stderr, "Error calling getnetconfigent(%s)\n", NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }

    /* (SVCXPRT *)svc_tp_create(dispatch, prognum, versnum, netconf) */
    svc = svc_tp_create(myapp_dispatch, PROGNUM, VERSNUM, nconf);

        /* check for errors calling svc_tp_create() */
    if (svc == (SVCXPRT *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.\n", "svc_tp_create");
        fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
                PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned.  ERROR has occurred.\n");
    fprintf(stderr, "errno: %d\n", errno);

    /* clean up by unregistering.  then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;

} /* end of main() */
```

Example: TI-RPC expert-level service API

This code example illustrates an expert-level service API that is used in developing TI-RPC services.

The expert level allows the programmer to specify Send and Receive buffer sizes for services. Calling svc_tli_create() creates the service handle, but it does not register or bind it with the rpcbind daemon. The programmer must call svc_reg() before the service works properly. Similar to the intermediate level, you have the option to use the network selection APIs to retrieve the transport information for the service.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <rpc/rpc.h>
#include <errno.h>
#include "myapp.h"

int main(int argc, char *argv[]) {

    struct netconfig *nconf; /* pointer to nettype data */
    SVCXPRT *svc;            /* pointer to service handle */
    bool_t rslt;             /* return value for svc_reg() */

    /* unregister any existing copy of this service */
    /* (void)svc_unreg(program, version) */
    svc_unreg(PROGNUM, VERSNUM);

    /* (struct netconfig *)getnetconfigent(nettype) */
    nconf = getnetconfigent(NETTYPE);

    /* check for errors calling getnetconfigent() */
    if (nconf == (struct netconfig *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling getnetconfigent(%s)\n", NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }
```

```
    /* (SVCXPRT *)svc_tli_create(filedes, netconfig, bindaddr, sendsz, recvsz) */
    svc = svc_tli_create(RPC_ANYFD, nconf, NULL, 0, 0);

    /* check for errors calling svc_tli_create() */
    if (svc == (SVCXPRT *)NULL) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling %s.\n", "svc_tli_create");
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }

    /* (bool_t)svc_reg(svcxprt, prognum, versnum, dispatch, netconf) */
    rslt = svc_reg(svc, PROGNUM, VERSNUM, myapp_dispatch, nconf);

    /* check for errors calling svc_reg() */
    if (rslt == FALSE) {
        /* print error messages and exit */
        fprintf(stderr, "Error calling svc_reg\n");
        fprintf(stderr, "PROG: %lu\nVERS: %lu\tNET: %s\n",
                PROGNUM, VERSNUM, NETTYPE);
        fprintf(stderr, "errno: %d\n", errno);
        return 1;
        }

    /* this should loop indefinitely waiting for client connections */
    svc_run();

    /* if we get here, svc_run() returned */
    fprintf(stderr, "svc_run() returned.  ERROR has occurred.\n");
    fprintf(stderr, "errno: %d\n", errno);

    /* clean up by unregistering.  then, exit */
    svc_unreg(PROGNUM, VERSNUM);

    return 1;

} /* end of main() */
```

Example: Adding authentication to the TI-RPC service

These code snippets display how the authentication system works in RPC.

System is the only authentication method that is provided on the i5/OS operating system. The following information is set up and passed from the client to the service with every clnt_call(). In the following code snippets, notice that rpc_call() is not sufficient when using authentication information because it uses authnone (an empty authentication token) as the default:

- aup_time - authentication information timestamp
- aup_machname - the hostname of the remote client
- aup_uid - the UID of the remote user of the client
- aup_gid - the primary GID of the remote user
- aup_gids - an array of the secondary groups of the remote user

The authentication information comes directly into the service as part of the remote request. It is up to the server to parse this information and verify that the client is from a trusted machine and a trusted user. If the authentication type is incorrect, or too weak for the server to accept, it sends back an error message, using svcerr_weakauth(), to indicate this to the client.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <sys/types.h> /* needed for gid_t and uid_t */
#include <stdlib.h>    /* misc. system auth APIs     */
#include <errno.h>

struct authsys_parms *credentials;  /* authentication information */
char *remote_machine;               /* machine name (from the credentials) */
uid_t remote_user;                  /* remote user's UID (from credentials) */

/* make sure we got the correct flavor of authentication */
if (request->rq_cred.oa_flavor != AUTH_UNIX) {
   /* if not, send back a weak authentication message and return */
```

```
   svcerr_weakauth(svc);
   return;
}

/* get our credentials */
credentials = (struct authsys_parms *)(request->rq_clntcred);

/* get the remote user's GID */
remote_user = credentials->aup_uid;

/* get the remote hostname of the client */
remote_machine = credentials->aup_machname;

/* check to see if this machine is "trusted" by us */
if ((strcmpi("remote1", remote_machine) != 0) &&
    (strcmpi("remote2", remote_machine) != 0)) {

   /* not from a machine we trust */
   /* send back an authentication error the client */
   svcerr_weakauth(svc);
   return;

} /* end of if (!trusted hostname) */

else {

   /* now check the user id for one we trust */
   /* information can be gotten from DSPUSRPRF */
   if ((remote_user != 568) &&
       (remote_user != 550) &&
       (remote_user != 528)) {

      /* not a user id we trust */
      /* send back an authentication error the client */
      svcerr_weakauth(svc);
      return;

   } /* end of if (!trusted uid) */

} /* end of else (trusted hostname) */

/* we fall out of the loop if the hostname and uid are trusted */
```

Examples: Developing client applications based on TI-RPC code

To develop client applications on the i5/OS operating system, use these code examples as a guideline.

Related concepts

Examples: Developing service applications based on TI-RPC code
Transport independent remote procedure call (TI-RPC) programming provides an effective method for developing distributed client-server based applications on the i5/OS operating system.

Related reference

Application programming interfaces

Example: TI-RPC simplified-level client API

This code example illustrates a simplified-level client API that is used in developing TI-RPC applications.

The simplified-level client API is the quickest and shortest set of code, because the client creation, control, use, and destruction are all in one call. This is convenient, but it does not allow the customization that can be done with a client handle. Defaults are accepted for timeout and buffer sizes, which is the most significant difference between the simplified level and the other levels.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <stdio.h>
#include <errno.h>
#include "myapp.h"

#define EXIT 100
```

```
int main(void) {

   enum clnt_stat rslt;    /* return value of rpc_call() */
   char hostname[256];     /* buffer for remote service's hostname */
   unsigned long procnum;  /* procedure to call */
   char filename[512];     /* buffer for filename */
   char *arg = filename;   /* pointer to filename buffer */

   union {
      u_int   myapp_get_uid_result;
      char *  myapp_get_uid_string_result;
      int     myapp_get_size_result;
      long    myapp_get_mtime_result;
      char *  myapp_get_mtime_string_result;
      u_short myapp_get_codepage_result;
      char *  myapp_get_objtype_result;
      char *  myapp_get_filetype_result;
   } result; /* a union of all the possible results */

   /* get the hostname from the user */
   printf("Enter the hostname where the remote service is running: \n");
   scanf("%s", (char *)&hostname);

   myapp_print_menu(); /* print out the menu choices */

   /* get the procedure number to call from the user */
   printf("\nEnter a procedure number to call: \n");
   scanf("%lu", &procnum);

   /* get the filename from the user */
   printf("\nEnter a filename to stat: \n");
   scanf("%s", (char *)&arg);

   /* switch on the input */
   switch (procnum) {

      case NULLPROC:

         /* rpc_call(host, prognum, versnum, procnum,    */
         /*          xdr_in, in, xdr_out, out, nettype); */
         rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                         (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in  */
                         (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                         NETTYPE);

         /* check return value of rpc_call() */
         if (rslt != RPC_SUCCESS) {
            fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
            fprintf(stderr, "clnt_stat: %d\n", rslt);
            fprintf(stderr, "errno: %d\n", errno);
            return 1;
         }

         /* print results and exit */
         printf("NULLRPOC call succeeded\n");
         break;

      case GET_UID:

         /* rpc_call(host, prognum, versnum, procnum,    */
         /*          xdr_in, in, xdr_out, out, nettype); */
         rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                         xdr_wrapstring, (char *)&arg, /* xdr_in  */
                         xdr_u_int, (char *)&result,        /* xdr_out */
                         NETTYPE);

         /* check return value of rpc_call() */
         if (rslt != RPC_SUCCESS) {
            fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
            fprintf(stderr, "clnt_stat: %d\n", rslt);
            fprintf(stderr, "errno: %d\n", errno);
            return 1;
         }

         /* print results and exit */
         printf("uid of %s: %u\n",
                filename, result.myapp_get_uid_result);
         break;

      case GET_UID_STRING:

         /* rpc_call(host, prognum, versnum, procnum,    */
```

```
   /*          xdr_in, in, xdr_out, out, nettype); */
   rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                   xdr_wrapstring, (char *)&arg, /* xdr_in  */
                   xdr_wrapstring, (char *)&result, /* xdr_out */
                   NETTYPE);

   /* check return value of rpc_call() */
   if (rslt != RPC_SUCCESS) {
      fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
      fprintf(stderr, "clnt_stat: %d\n", rslt);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* print results and exit */
   printf("owner of %s: %s\n",
          filename, result.myapp_get_uid_string_result);
   break;

case GET_SIZE:

   /* rpc_call(host, prognum, versnum, procnum,    */
   /*          xdr_in, in, xdr_out, out, nettype); */
   rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                   xdr_wrapstring, (char *)&arg, /* xdr_in  */
                   xdr_int, (char *)&result, /* xdr_out */
                   NETTYPE);

   /* check return value of rpc_call() */
   if (rslt != RPC_SUCCESS) {
      fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
      fprintf(stderr, "clnt_stat: %d\n", rslt);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* print results and exit */
   printf("size of %s: %d\n",
          filename, result.myapp_get_size_result);
   break;

case GET_MTIME:

   /* rpc_call(host, prognum, versnum, procnum,    */
   /*          xdr_in, in, xdr_out, out, nettype); */
   rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                   xdr_wrapstring, (char *)&arg, /* xdr_in  */
                   xdr_long, (char *)&result, /* xdr_out */
                   NETTYPE);

   /* check return value of rpc_call() */
   if (rslt != RPC_SUCCESS) {
      fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
      fprintf(stderr, "clnt_stat: %d\n", rslt);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* print results and exit */
   printf("last modified time of %s: %ld\n",
          filename, result.myapp_get_mtime_result);
   break;

case GET_MTIME_STRING:

   /* rpc_call(host, prognum, versnum, procnum,    */
   /*          xdr_in, in, xdr_out, out, nettype); */
   rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                   xdr_wrapstring, (char *)&arg, /* xdr_in  */
                   xdr_wrapstring, (char *)&result, /* xdr_out */
                   NETTYPE);

   /* check return value of rpc_call() */
   if (rslt != RPC_SUCCESS) {
      fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
      fprintf(stderr, "clnt_stat: %d\n", rslt);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* print results and exit */
   printf("last modified time of %s: %s\n",
```

```
            filename, result.myapp_get_mtime_string_result);
      break;

   case GET_CODEPAGE:

      /* rpc_call(host, prognum, versnum, procnum,    */
      /*          xdr_in, in, xdr_out, out, nettype); */
      rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                     xdr_wrapstring, (char *)&arg, /* xdr_in  */
                     xdr_u_short, (char *)&result, /* xdr_out */
                     NETTYPE);

      /* check return value of rpc_call() */
      if (rslt != RPC_SUCCESS) {
         fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
         fprintf(stderr, "clnt_stat: %d\n", rslt);
         fprintf(stderr, "errno: %d\n", errno);
         return 1;
      }

      /* print results and exit */
      printf("codepage of %s: %d\n",
            filename, result.myapp_get_codepage_result);
      break;

   case GET_OBJTYPE:

      /* rpc_call(host, prognum, versnum, procnum,    */
      /*          xdr_in, in, xdr_out, out, nettype); */
      rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                     xdr_wrapstring, (char *)&arg, /* xdr_in  */
                     xdr_wrapstring, (char *)&result, /* xdr_out */
                     NETTYPE);

      /* check return value of rpc_call() */
      if (rslt != RPC_SUCCESS) {
         fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
         fprintf(stderr, "clnt_stat: %d\n", rslt);
         fprintf(stderr, "errno: %d\n", errno);
         return 1;
      }

      /* print results and exit */
      printf("object type of %s: %s\n",
            filename, result.myapp_get_objtype_result);
      break;

   case GET_FILETYPE:

      /* rpc_call(host, prognum, versnum, procnum,    */
      /*          xdr_in, in, xdr_out, out, nettype); */
      rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                     xdr_wrapstring, (char *)&arg, /* xdr_in  */
                     xdr_wrapstring, (char *)&result, /* xdr_out */
                     NETTYPE);

      /* check return value of rpc_call() */
      if (rslt != RPC_SUCCESS) {
         fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
         fprintf(stderr, "clnt_stat: %d\n", rslt);
         fprintf(stderr, "errno: %d\n", errno);
         return 1;
      }

      /* print results and exit */
      printf("file type of %s: %s\n",
            filename, result.myapp_get_filetype_result);
      break;

   case END_SERVER:

      /* rpc_call(host, prognum, versnum, procnum,    */
      /*          xdr_in, in, xdr_out, out, nettype); */
      rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                     (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in  */
                     (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                     NETTYPE);

      /* check return value of rpc_call() */
      if (rslt != RPC_SUCCESS) {
         fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
         fprintf(stderr, "clnt_stat: %d\n", rslt);
```

```
            fprintf(stderr, "errno: %d\n", errno);
            return 1;
         }

         /* print results and exit */
         printf("Service has been unregistered.\n");
         printf("You must still kill the job in QBATCH\n");
         break;

      case EXIT:

         /* do nothing and exit */
         printf("Exiting program now.\n");
         return 1;
         break;

      default:

         /* an invalid procedure number was entered */
         /* we could just exit here */
         printf("Invalid choice.  Issuing NULLRPOC instead.\n");
         procnum = NULLPROC;

         /* rpc_call(host, prognum, versnum, procnum,    */
         /*          xdr_in, in, xdr_out, out, nettype); */
         rslt = rpc_call(hostname, PROGNUM, VERSNUM, procnum,
                         (xdrproc_t)xdr_void, (char *)NULL, /* xdr_in  */
                         (xdrproc_t)xdr_void, (char *)NULL, /* xdr_out */
                         NETTYPE);

         /* check return value of rpc_call() */
         if (rslt != RPC_SUCCESS) {
            fprintf(stderr, "Error calling rpc_call(%lu)\n", procnum);
            fprintf(stderr, "clnt_stat: %d\n", rslt);
            fprintf(stderr, "errno: %d\n", errno);
            return 1;
         }

         /* print results and exit */
         printf("NULLRPOC call succeeded\n");
         break;

   } /* end of switch(procnum) */

   /* no cleanup is required for rpc_call() */
   return 0;

}

void myapp_print_menu(void) {

   /* print out the procedure choices */
   printf("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

Example: TI-RPC top-level client API

This code example illustrates a top-level client API that is used in developing TI-RPC applications.

At the top level, you must create a client handle before you can use it or modify it. Top-level APIs are easy to use, and they allow more manipulation and error handling than the simplified level.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>
```

```
#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

   enum clnt_stat rslt;    /* return value of clnt_call() */
   char hostname[256];    /* buffer for remote service's hostname */
   unsigned long procnum;  /* procedure to call */
   char filename[512];     /* buffer for filename */
   xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
   xdrproc_t xdr_result;   /* xdr procedure to decode results */
   CLIENT *clnt;           /* pointer to client handle */
   struct timeval tout;    /* timeout for clnt_call() */
   char *arg = filename;   /* pointer to filename buffer */

   union {
      u_int   myapp_get_uid_result;
      char *  myapp_get_uid_string_result;
      int     myapp_get_size_result;
      long    myapp_get_mtime_result;
      char *  myapp_get_mtime_string_result;
      u_short myapp_get_codepage_result;
      char *  myapp_get_objtype_result;
      char *  myapp_get_filetype_result;
   } result; /* a union of all the possible results */

   tout.tv_sec  = 30;  /* set default timeout to 30.00 seconds */
   tout.tv_usec =  0;

   /* get the hostname from the user */
   printf("Enter the hostname where the remote service is running: \n");
   scanf("%s", (char *)&hostname);

   myapp_print_menu(); /* print out the menu choices */

   /* get the procedure number to call from the user */
   printf("\nEnter a procedure number to call: \n");
   scanf("%lu", &procnum);

   /* get the filename from the user */
   printf("\nEnter a filename to stat: \n");
   scanf("%s", (char *)&filename);

   /* clnt_create(host, prognum, versnum, nettype);    */
   clnt = clnt_create(hostname, PROGNUM, VERSNUM, NETTYPE);

   /* check to make sure clnt_create() didn't fail */
   if (clnt == (CLIENT *)NULL) {
      /* if we failed, print out all appropriate error messages and exit */
      fprintf(stderr, "Error calling clnt_create()\n");
      fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
              PROGNUM, VERSNUM, NETTYPE);
      fprintf(stderr, "clnt_stat: %d\n", rpc_createerr.cf_stat);
      fprintf(stderr, "errno: %d\n", errno);
      fprintf(stderr, "re_errno: %d\n", rpc_createerr.cf_error.re_errno);
      return 1;
   }

   /* switch on the input */
   switch (procnum) {

      case NULLPROC:
         /* set the encode procedure */
         xdr_argument = (xdrproc_t)xdr_void;
         /* set the decode procedure */
         xdr_result   = (xdrproc_t)xdr_void;
         break;

      case GET_UID:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_u_int;
         break;

      case GET_UID_STRING:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_wrapstring;
```

```
      break;

   case GET_SIZE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_int;
      break;

   case GET_MTIME:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_long;
      break;

   case GET_MTIME_STRING:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case GET_CODEPAGE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_u_short;
      break;

   case GET_OBJTYPE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case GET_FILETYPE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case END_SERVER:
      /* set the encode procedure */
      xdr_argument = (xdrproc_t)xdr_void;
      /* set the decode procedure */
      xdr_result   = (xdrproc_t)xdr_void;
      break;

   case EXIT:
      /* we're done.  clean up and exit */
      clnt_destroy(clnt);
      return 1;
      break;

   default:
      /* invalid procedure number entered.  defaulting to NULLPROC */
      printf("Invalid choice.  Issuing NULLRPOC instead.\n");
      procnum = NULLPROC;
      /* set the encode procedure */
      xdr_argument = (xdrproc_t)xdr_void;
      /* set the decode procedure */
      xdr_result   = (xdrproc_t)xdr_void;
      break;

} /* end of switch(procnum) */

/* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                 xdr_result, (char *)&result, tout);

/* check to make sure clnt_call() succeeded */
if (rslt != RPC_SUCCESS) {
   /* if clnt_call() failed, print errors and exit */
   printf("An error occurred calling %lu procedure\n", procnum);
   printf("clnt_stat: %d\terrno: %d\n", rslt, errno);
   clnt_destroy(clnt);
   return 1;
}
```

```
   /* clnt_call() succeeded.  switch on procedure and print results */
   switch (procnum) {

      case NULLPROC:
         /* print results and exit */
         printf("NULLRPOC call succeeded\n");
         break;

      case GET_UID:
         /* print results and exit */
         printf("uid of %s: %u\n",
                filename, result.myapp_get_uid_result);
         break;

      case GET_UID_STRING:
         /* print results and exit */
         printf("owner of %s: %s\n",
                filename, result.myapp_get_uid_string_result);
         break;

      case GET_SIZE:
         /* print results and exit */
         printf("size of %s: %d\n",
                filename, result.myapp_get_size_result);
         break;

      case GET_MTIME:
         /* print results and exit */
         printf("last modified time of %s: %ld\n",
                filename, result.myapp_get_mtime_result);
         break;

      case GET_MTIME_STRING:
         /* print results and exit */
         printf("last modified time of %s: %s\n",
                filename, result.myapp_get_mtime_string_result);
         break;

      case GET_CODEPAGE:
         /* print results and exit */
         printf("codepage of %s: %d\n",
                filename, result.myapp_get_codepage_result);
         break;

      case GET_OBJTYPE:
         /* print results and exit */
         printf("object type of %s: %s\n",
                filename, result.myapp_get_objtype_result);
         break;

      case GET_FILETYPE:
         /* print results and exit */
         printf("file type of %s: %s\n",
                filename, result.myapp_get_filetype_result);
         break;

      case END_SERVER:
         /* print results and exit */
         printf("Service has been unregistered.\n");
         printf("You must still kill the job in QBATCH\n");
         break;

      default:
         /* we should never get the default case.  */
         /* the previous switch should catch it.   */
         break;

   } /* end of switch(procnum) */


   /* clean up and exit */
   clnt_destroy(clnt);

   return 0;

}

void myapp_print_menu(void) {

   /* print out the procedure choices */
   printf("%.2ld - GET_UID            %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
```

```
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

Example: TI-RPC intermediate-level client API

This code example illustrates an intermediate-level client API that is used in developing TI-RPC applications.

The intermediate level for the client follows the same path as it does for the service. For example, the programmer is responsible for using the network selection APIs to obtain transport information instead of passing a simple text string.

Note: By using the code example, you agree to the terms of the "Code license and disclaimer information" on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>
#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

   enum clnt_stat rslt;    /* return value of clnt_call() */
   char hostname[256];    /* buffer for remote service's hostname */
   unsigned long procnum;  /* procedure to call */
   char filename[512];     /* buffer for filename */
   xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
   xdrproc_t xdr_result;   /* xdr procedure to decode results */
   CLIENT *clnt;           /* pointer to client handle */
   struct timeval tout;    /* timeout for clnt_call() */
   struct netconfig *nconf;/* transport information */
   char *arg = filename;   /* pointer to filename buffer */

   union {
      u_int   myapp_get_uid_result;
      char *  myapp_get_uid_string_result;
      int     myapp_get_size_result;
      long    myapp_get_mtime_result;
      char *  myapp_get_mtime_string_result;
      u_short myapp_get_codepage_result;
      char *  myapp_get_objtype_result;
      char *  myapp_get_filetype_result;
   } result; /* a union of all the possible results */


   tout.tv_sec  = 30;  /* set default timeout to 30.00 seconds */
   tout.tv_usec =  0;

   /* get the hostname from the user */
   printf("Enter the hostname where the remote service is running: \n");
   scanf("%s", (char *)&hostname);

   myapp_print_menu(); /* print out the menu choices */

   /* get the procedure number to call from the user */
   printf("\nEnter a procedure number to call: \n");
   scanf("%lu", &procnum);

   /* get the filename from the user */
   printf("\nEnter a filename to stat: \n");
   scanf("%s", (char *)&filename);

   /* getnetconfigent(nettype) */
   nconf = getnetconfigent(NETTYPE);

   /* check to make sure getnetconfigent() didn't fail */
```

```
   if (nconf == NULL) {
      /* if getnetconfigent() failed, print error messages and exit */
      fprintf(stderr, "Error calling getnetconfigent(%s)\n", NETTYPE);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* clnt_tp_create(host, prognum, versnum, netconf) */
   clnt = clnt_tp_create(hostname, PROGNUM, VERSNUM, nconf);

   /* check to make sure clnt_tp_create() didn't fail */
   if (clnt == (CLIENT *)NULL) {
      fprintf(stderr, "Error calling clnt_tp_create()\n");
      fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
              PROGNUM, VERSNUM, NETTYPE);
      fprintf(stderr, "clnt_stat: %d\n", rpc_createerr.cf_stat);
      fprintf(stderr, "errno: %d\n", errno);
      fprintf(stderr, "re_errno: %d\n", rpc_createerr.cf_error.re_errno);
      return 1;
   }

   /* switch on the input */
   switch (procnum) {

      case NULLPROC:
         /* set the encode procedure */
         xdr_argument = (xdrproc_t)xdr_void;
         /* set the decode procedure */
         xdr_result   = (xdrproc_t)xdr_void;
         break;

      case GET_UID:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_u_int;
         break;

      case GET_UID_STRING:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_wrapstring;
         break;

      case GET_SIZE:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_int;
         break;

      case GET_MTIME:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_long;
         break;

      case GET_MTIME_STRING:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_wrapstring;
         break;

      case GET_CODEPAGE:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_u_short;
         break;

      case GET_OBJTYPE:
         /* set the encode procedure */
         xdr_argument = xdr_wrapstring;
         /* set the decode procedure */
         xdr_result   = xdr_wrapstring;
         break;

      case GET_FILETYPE:
         /* set the encode procedure */
```

```
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result   = xdr_wrapstring;
        break;

     case END_SERVER:
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result   = (xdrproc_t)xdr_void;
        break;

     case EXIT:
        /* we're done.  clean up and exit */
        clnt_destroy(clnt);
        return 1;
        break;

     default:
        /* invalid procedure number entered.  defaulting to NULLPROC */
        printf("Invalid choice.  Issuing NULLRPOC instead.\n");
        procnum = NULLPROC;
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result   = (xdrproc_t)xdr_void;
        break;

  } /* end of switch(procnum) */

  /* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
  rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                   xdr_result, (char *)&result, tout);

  /* check to make sure clnt_call() succeeded */
  if (rslt != RPC_SUCCESS) {
     /* if clnt_call() failed, print errors and exit */
     printf("An error occurred calling %lu procedure\n", procnum);
     printf("clnt_stat: %d\terrno: %d\n", rslt, errno);
     clnt_destroy(clnt);
     return 1;
  }

  /* clnt_call() succeeded.  switch on procedure and print results */
  switch (procnum) {

     case NULLPROC:
        /* print results and exit */
        printf("NULLRPOC call succeeded\n");
        break;

     case GET_UID:
        /* print results and exit */
        printf("uid of %s: %u\n",
               filename, result.myapp_get_uid_result);
        break;

     case GET_UID_STRING:
        /* print results and exit */
        printf("owner of %s: %s\n",
               filename, result.myapp_get_uid_string_result);
        break;

     case GET_SIZE:
        /* print results and exit */
        printf("size of %s: %d\n",
               filename, result.myapp_get_size_result);
        break;

     case GET_MTIME:
        /* print results and exit */
        printf("last modified time of %s: %ld\n",
               filename, result.myapp_get_mtime_result);
        break;

     case GET_MTIME_STRING:
        /* print results and exit */
        printf("last modified time of %s: %s\n",
               filename, result.myapp_get_mtime_string_result);
        break;

     case GET_CODEPAGE:
```

```
         /* print results and exit */
         printf("codepage of %s: %d\n",
                filename, result.myapp_get_codepage_result);
         break;

      case GET_OBJTYPE:
         /* print results and exit */
         printf("object type of %s: %s\n",
                filename, result.myapp_get_objtype_result);
         break;

      case GET_FILETYPE:
         /* print results and exit */
         printf("file type of %s: %s\n",
                filename, result.myapp_get_filetype_result);
         break;

      case END_SERVER:
         /* print results and exit */
         printf("Service has been unregistered.\n");
         printf("You must still kill the job in QBATCH\n");
         break;

      default:
         /* we should never get the default case.  */
         /* the previous switch should catch it.   */
         break;

   } /* end of switch(procnum) */


   /* clean up and exit */

   /* free the netconfig struct */
   freenetconfigent(nconf);
   /* free the universal address buffer */
   free(svcaddr.buf);
   /* destroy the client handle */
   clnt_destroy(clnt);


   return 0;

}

void myapp_print_menu(void) {

   /* print out the procedure choices */
   printf("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

Example: TI-RPC expert-level client API

This code example illustrates an expert-level client API that is used in developing TI-RPC applications.

The expert level for the development of a client API is the most complicated. It also offers the most customization. This is the only level where the buffer size can be tuned for the client API. This level requires the programmer to set up the universal address for the client to connect to, either by using the name-to-address translation APIs or one of the other expert-level APIs. Either way, this level requires more work, but it allows the programmer to tailor the client application to the environment it runs in.

Note: By using the code example, you agree to the terms of the “Code license and disclaimer information” on page 30.

```
#include <stdio.h>
#include <netconfig.h>
#include <netdir.h>
```

```
#include <errno.h>
#include "myapp.h"

#define EXIT 100

int main(void) {

   enum clnt_stat rslt;    /* return value of clnt_call() */
   char hostname[256];    /* buffer for remote service's hostname */
   unsigned long procnum;  /* procedure to call */
   char filename[512];     /* buffer for filename */
   xdrproc_t xdr_argument; /* xdr procedure to encode arguments */
   xdrproc_t xdr_result;   /* xdr procedure to decode results */
   CLIENT *clnt;           /* pointer to client handle */
   struct timeval tout;    /* timeout for clnt_call() */
   struct netconfig *nconf;/* transport information */
   struct netbuf svcaddr;  /* universal address of remote service */
   bool_t rpcb_rslt;       /* return value for rpcb_getaddr() */
   char *arg = filename;   /* pointer to filename buffer */

   union {
      u_int   myapp_get_uid_result;
      char *  myapp_get_uid_string_result;
      int     myapp_get_size_result;
      long    myapp_get_mtime_result;
      char *  myapp_get_mtime_string_result;
      u_short myapp_get_codepage_result;
      char *  myapp_get_objtype_result;
      char *  myapp_get_filetype_result;
   } result; /* a union of all the possible results */

   /* initialize the struct netbuf space */
   svcaddr.maxlen = 16;
   svcaddr.buf = (char *)malloc(svcaddr.maxlen);

   if (svcaddr.buf == (char *)NULL) {
      /* if malloc() failed, print error messages and exit */
      fprintf(stderr, "Error calling malloc() for struct netbuf\n");
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   tout.tv_sec  = 30;  /* set default timeout to 30.00 seconds */
   tout.tv_usec =  0;

   /* get the hostname from the user */
   printf("Enter the hostname where the remote service is running: \n");
   scanf("%s", (char *)&hostname);

   myapp_print_menu(); /* print out the menu choices */

   /* get the procedure number to call from the user */
   printf("\nEnter a procedure number to call: \n");
   scanf("%lu", &procnum);

   /* get the filename from the user */
   printf("\nEnter a filename to stat: \n");
   scanf("%s", (char *)&filename);

   /* getnetconfigent(nettype) */
   nconf = getnetconfigent(NETTYPE);

   /* check to make sure getnetconfigent() didn't fail */
   if (nconf == NULL) {
      /* if getnetconfigent() failed, print error messages and exit */
      fprintf(stderr, "Error calling getnetconfigent(%s)\n", NETTYPE);
      fprintf(stderr, "errno: %d\n", errno);
      return 1;
   }

   /* rpcb_getaddr(prognum, versnum, nconf, output netbuf, hostname) */
   /* this sets the universal address svcaddr */
   rpcb_rslt = rpcb_getaddr(PROGNUM, VERSNUM, nconf, &svcaddr, hostname);

   /* check to make sure rpcb_getaddr() didn't fail */
   if (rpcb_rslt == FALSE) {
      /* if rpcb_getaddr() failed, print error messages and exit */
      fprintf(stderr, "Error calling rpcb_getaddr()\n");
      fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
              PROGNUM, VERSNUM, NETTYPE);
      fprintf(stderr, "clnt_stat: %d\n", rpc_createerr.cf_stat);
      fprintf(stderr, "errno: %d\n", errno);
```

```
   fprintf(stderr, "re_errno: %d\n", rpc_createerr.cf_error.re_errno);
   return 1;
}

/* clnt_tli_create(filedes, netconfig, netbuf,           */
/*                 prognum, versnum, sendsz, recvsz);    */
clnt = clnt_tli_create(RPC_ANYFD, nconf, &svcaddr,
                       PROGNUM, VERSNUM, 0, 0);

/* check to make sure clnt_tli_create() didn't fail */
if (clnt == (CLIENT *)NULL) {
   /* if we failed, print out all appropriate error messages and exit */
   fprintf(stderr, "Error calling clnt_tli_create()\n");
   fprintf(stderr, "PROG: %lu\tVERS: %lu\tNET: %s\n",
           PROGNUM, VERSNUM, NETTYPE);
   fprintf(stderr, "clnt_stat: %d\n", rpc_createerr.cf_stat);
   fprintf(stderr, "errno: %d\n", errno);
   fprintf(stderr, "re_errno: %d\n", rpc_createerr.cf_error.re_errno);
   return 1;
}

/* switch on the input */
switch (procnum) {

   case NULLPROC:
      /* set the encode procedure */
      xdr_argument = (xdrproc_t)xdr_void;
      /* set the decode procedure */
      xdr_result   = (xdrproc_t)xdr_void;
      break;

   case GET_UID:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_u_int;
      break;

   case GET_UID_STRING:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case GET_SIZE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_int;
      break;

   case GET_MTIME:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_long;
      break;

   case GET_MTIME_STRING:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case GET_CODEPAGE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_u_short;
      break;

   case GET_OBJTYPE:
      /* set the encode procedure */
      xdr_argument = xdr_wrapstring;
      /* set the decode procedure */
      xdr_result   = xdr_wrapstring;
      break;

   case GET_FILETYPE:
      /* set the encode procedure */
```

```
        xdr_argument = xdr_wrapstring;
        /* set the decode procedure */
        xdr_result   = xdr_wrapstring;
        break;

     case END_SERVER:
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result   = (xdrproc_t)xdr_void;
        break;

     case EXIT:
        /* we're done.  clean up and exit */
        clnt_destroy(clnt);
        return 1;
        break;

     default:
        /* invalid procedure number entered.  defaulting to NULLPROC */
        printf("Invalid choice.  Issuing NULLRPOC instead.\n");
        procnum = NULLPROC;
        /* set the encode procedure */
        xdr_argument = (xdrproc_t)xdr_void;
        /* set the decode procedure */
        xdr_result   = (xdrproc_t)xdr_void;
        break;

  } /* end of switch(procnum) */

  /* clnt_call(client, procnum, xdr_inproc, in, xdr_outproc, out, timeout) */
  rslt = clnt_call(clnt, procnum, xdr_argument, (char *)&arg,
                   xdr_result, (char *)&result, tout);

  /* check to make sure clnt_call() succeeded */
  if (rslt != RPC_SUCCESS) {
     /* if clnt_call() failed, print errors and exit */
     printf("An error occurred calling %lu procedure\n", procnum);
     printf("clnt_stat: %d\terrno: %d\n", rslt, errno);
     clnt_destroy(clnt);
     return 1;
  }

  /* clnt_call() succeeded.  switch on procedure and print results */
  switch (procnum) {

     case NULLPROC:
        /* print results and exit */
        printf("NULLRPOC call succeeded\n");
        break;

     case GET_UID:
        /* print results and exit */
        printf("uid of %s: %u\n",
               filename, result.myapp_get_uid_result);
        break;

     case GET_UID_STRING:
        /* print results and exit */
        printf("owner of %s: %s\n",
               filename, result.myapp_get_uid_string_result);
        break;

     case GET_SIZE:
        /* print results and exit */
        printf("size of %s: %d\n",
               filename, result.myapp_get_size_result);
        break;

     case GET_MTIME:
        /* print results and exit */
        printf("last modified time of %s: %ld\n",
               filename, result.myapp_get_mtime_result);
        break;

     case GET_MTIME_STRING:
        /* print results and exit */
        printf("last modified time of %s: %s\n",
               filename, result.myapp_get_mtime_string_result);
        break;

     case GET_CODEPAGE:
```

```
         /* print results and exit */
         printf("codepage of %s: %d\n",
                filename, result.myapp_get_codepage_result);
         break;

      case GET_OBJTYPE:
         /* print results and exit */
         printf("object type of %s: %s\n",
                filename, result.myapp_get_objtype_result);
         break;

      case GET_FILETYPE:
         /* print results and exit */
         printf("file type of %s: %s\n",
                filename, result.myapp_get_filetype_result);
         break;

      case END_SERVER:
         /* print results and exit */
         printf("Service has been unregistered.\n");
         printf("You must still kill the job in QBATCH\n");
         break;

      default:
         /* we should never get the default case.  */
         /* the previous switch should catch it.   */
         break;

   } /* end of switch(procnum) */


   /* clean up and exit */

   /* free the netconfig struct */
   freenetconfigent(nconf);
   /* free the universal address buffer */
   free(svcaddr.buf);
   /* destroy the client handle */
   clnt_destroy(clnt);


   return 0;

}

void myapp_print_menu(void) {

   /* print out the procedure choices */
   printf("%.2ld - GET_UID             %.2ld - GET_UID_STRING\n",
          GET_UID, GET_UID_STRING);
   printf("%.2ld - GET_SIZE            %.2ld - GET_MTIME\n",
          GET_SIZE, GET_MTIME);
   printf("%.2ld - GET_MTIME_STRING    %.2ld - GET_CODEPAGE\n",
          GET_MTIME_STRING, GET_CODEPAGE);
   printf("%.2ld - GET_OBJTYPE         %.2ld - GET_FILETYPE\n",
          GET_OBJTYPE, GET_FILETYPE);
   printf("%.2ld - END_SERVER          %.2d - EXIT\n",
          END_SERVER, EXIT);

}
```

Example: Adding authentication to the TI-RPC client

These code snippets display how the authentication system works in RPC.

The system method is the only authentication method that is provided on the i5/OS operating system. The following information is set up and passed from the client to the service with every clnt_call(). In the following code snippets, notice that rpc_call() is not sufficient when using authentication information because it uses `authnone` (an empty authentication token) as the default:

- `aup_time - authentication information timestamp`
- `aup_machname - the hostname of the remote client`
- `aup_uid - the UID of the remote user of the client`
- `aup_gid - the primary GID of the remote user`

- aup_gids - an array of the secondary groups of the remote user

It is up to the client to set up the authentication information and make it part of the client handle. After that, all subsequent calls to clnt_call() will pass that authentication information along. It is up to the server to report on unauthorized clients. RPC only provides a simple method of communicating the information. The data that is sent by the client is authenticated, but not encrypted. The reply from the service is not encrypted either. Authentication provides a simple way of verifying the remote host name and the user identification. It cannot be considered a secure and private method of communication.

Note: By using the code example, you agree to the terms of the "Code license and disclaimer information" on page 30.

```
#include <sys/types.h> /* needed for gid_t and uid_t */
#include <stdlib.h>    /* misc. system auth APIs     */
#include <unistd.h>    /* misc. system auth APIs     */
#include <errno.h>

#ifndef NGROUPS_MAX
   #define NGROUPS_MAX 16
#endif

char hostname[256];         /* hostname for credentials */
int rslt;                   /* return value of gethostname() */
gid_t groups[NGROUPS_MAX];  /* array of groups set by getgroups() */
gid_t *aup_gids;            /* pointer to array of gid_t */
uid_t uid;                  /* uid, return value for geteuid() */
gid_t gid;                  /* gid, return value for getegid() */
int num_groups;             /* return value for getgroups(), number of groups set
*/

aup_gids = groups;          /* point to the array of groups */
uid = geteuid();            /* get the effective uid of the user */
gid = getegid();            /* get the effect primary gid of the user */

/* get a list of other groups the user is a member of */
/* (int)getgroups(maxgropus, array) */
num_groups = getgroups(NGROUPS_MAX, groups);

/* check return value of getgroups() for error */
if (num_groups == -1) {
    /* print error message and exit */
    fprintf(stderr, "getgroups() failed for %d\n", uid);
    fprintf(stderr, "errno: %d\n", errno);
    return 1;
}

/* (int)gethostname(buffer, buflen) */
rslt = gethostname(hostname, 256);

/* check return value of gethostname() for error */
if (rslt == -1) {
    /* print error message and exit */
    fprintf(stderr, "gethostname() failed\n");
    fprintf(stderr, "errno: %d\n", errno);
    return 1;
}

/* insert just before clnt_call() */
/* (AUTH *)authsys_create(hostname, uid, gid, num_groups, gid[]); */
clnt->cl_auth = authsys_create(hostname, uid, gid, num_groups, aup_gids);

if (clnt->cl_auth == NULL) {
    /* print error messages and exit */
    fprintf(stderr, "authsys_create() failed\n");
    fprintf(stderr, "errno: %d\n", errno);
    /* clean up */
    clnt_destroy(clnt);
    return 1;
}
```

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Using Sun TI-RPC to develop distributed applications publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Other product and service names might be trademarks of IBM® or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Product Number: 5770-SS1