RPC +

User's Guide

Version 1.4

England Technical Services, Inc.
210 Main Street
Elliott, IA 51532
Phone: (712) 767-2270
Fax: (712) 767-2227
e-mail: ets@netins.net
Web: http://www.netins.net/showcase/etsinc

CONTENTS

Chapter 1: Installation

Introduction

To use RPC+ it must be installed on both a client and a server. Unix and 32-bit Windows operating systems are supported for both the client and server. For testing or development purposes the same machine may be used as both client and server, although there is little need for this in a production setting.

The RPC+ installation CD or zip file installs under Windows. It creates a Unix subdirectory which will contain the installation file for Unix. After installing RPC+ on your client and server as described below, see Chapter 2 for information on executing the example programs.

Windows

The Windows installation starts with SETUP.EXE. Find SETUP.EXE on your installation media and execute it. If you are installing from CD this process should happen automatically. The installation process will then ask you the following questions.

Choose Destination Location

This lets you choose the location on your computer where RPC+ is installed. RPC+ will create several subdirectories under this location to install various types of files. A bin directory will be created to hold the executable and DLL files. A source directory will be created to hold files needed when compiling application programs. An example directory will be created to hold samples of how to use RPC+ with different languages and architectures. The DLL files will also be installed into the Windows directory, so that they can be found by the operating system when RPC+ applications are executed.

Setup Type

This lets you choose Typical (all files will be installed), Compact (no example or source files will be installed), or Custom (you choose what is installed) installations. If this is your first experience with using RPC+ we recommend the Typical installation.

Select Program Folder

When the example programs are installed, RPC+ creates a number of icons used to execute the samples. This dialog lets you choose the program folder the icons will be created in.

Configure TCP/IP Ports

RPC+ uses TCP/IP to communicate between the client and server. This communication takes place on a logical TCP/IP port number that must be specified the same on the client and server. This dialog lets you choose the port numbers to use for the different types of services supported by RPC+. Be sure the numbers entered here match between the client and the server and are not used by any other service on the server. The command "`netstat -a`" can be used to see what port numbers are currently in use on your system.

Specify Server

The Windows installation of RPC+ installs both the client and server components. If you are going to use the client components, you need to identify the server the client should communicate with. This server name or IP address will be placed in the rpcplus.ini files used by the example programs. If you do not yet know the IP address of the server you will be accessing, you can add this information later directly into the DefaultServer entry in the [ClientConfig] section of the rpcplus.ini file used by the client. Each sample has an rpcplus.ini file in its directory.

Server License

If you are installing the software on a machine that will function as an RPC+ server, you will want to enter your RPC+ license number here. Entering a valid RPC+ server license will allow the server to have full functionality. You can install and use an RPC+ server without a license number, but the server will only allow a single user at a time and the clients will display demonstration messages. If you do not have an RPC+ license number now, you can add it later by placing it in the License entry of the [ServerConfig] section of the rpcplus.ini file used by the server. Each RPC+ service on the server will have an rpcplus.ini file in its directory.

Unix

The Unix installation file is created as part of the Windows installation and is placed in the Unix subdirectory of the Windows installation directory. This file should be transferred to the Unix server with a binary file transfer. The Unix installation process differs depending on the language you are using to build your server applications. There is a separate installation routine for each application language. The following sections describe the installation process for each of the server languages. Following the language dependent sections are notes which apply to all Unix server installations.

MicroFocus

Create a new directory and extract the installations files into that directory with the following command:

```
cpio -icvBd <xxx
```

where xxx is your installation file. Then execute the installation script mfinstall.

You will be asked to enter your RPC+ license number. Entering a valid RPC+ server license will allow the server to have full functionality. You can install and use an RPC+ server without a license number, but the server will only allow a single user at a time, and the clients will display demonstration messages. If you do not have an RPC+ license number now, you can add it later by placing it in the License entry of the [ServerConfig] section of the rpcplus.ini file used by the server. Each RPC+ service on the server will have an rpcplus.ini file in its directory.

You will also be asked to enter port numbers to use for the fat and thin client services. RPC+ uses TCP/IP to communicate between client and server. This communication takes place on a logical TCP/IP port number that must be specified the same on the client and server. Be sure the numbers entered here match between the client and the server, and are not used by any other service on the server. The command “netstat –a” can be used to see what port numbers are currently in use on your system. It may also be helpful to look at the services already listed in the /etc/services file.

The installation will update the /etc/services and /etc/inetd.conf files. Two new services, rpcplus and rpcplustc will be created and assigned to the port numbers you specified.

The installation will build the RPC+ object file rpcplus.o. If the mkrts command is located, the installation routine will also create a new rts32 file by linking the MicroFocus runtime with rpcplus.o. If the mkrts command is not located you will have to manually relink the MicroFocus runtime.

The MicroFocus runtime system can be linked with RPC+ as follows:

mkrts rpcplus.o

The rpcplus.o file is created automatically at installation. If it is somehow lost or overwritten it can be recreated by executing the following command in the etssrc subdirectory of your RPC+ installation:

```
cc -c rpcplus.c
```

If you have your own subroutines that you normally link into the MicroFocus runtime, just include them on the mkrts command line along with rpcplus.o

If the installation routine completed successfully you are ready to run. Otherwise, see the appropriate sections below to resolve any installation issues.

RM/COBOL

Create a new directory and extract the installations files into that directory with the following command:

```
cpio –icvBd <xxx
```

where xxx is your installation file. Then execute the installation script rminstall

You will be asked to enter the location of your RM/COBOL runtime. This is so the install routine can locate the necessary files to relink the runtime, or create a shared object library, depending on which version of RM/COBOL you are using. There are more details on this later in this section.

You will be asked to enter your RPC+ license number. Entering a valid RPC+ server license will allow the server to have full functionality. You can install and use an RPC+ server without a license number, but the server will only allow a single user at a time, and the clients will display demonstration messages. If you do not have an RPC+ license number now, you can add it later by placing it in the License entry of the [ServerConfig] section of the rpcplus.ini file used by the server. Each RPC+ service on the server will have an rpcplus.ini file in its directory.

You will also be asked to enter port numbers to use for the fat and thin client services. RPC+ uses TCP/IP to communicate between client and server. This communication takes place on a logical TCP/IP port number that must be specified the same on the client and server. Be sure the numbers entered here match between the client and the server, and are not used by any other service on the server. The command "netstat –a" can be used to see what port numbers are currently in use on your system. It may also be helpful to look at the services already listed in the /etc/services file.

The installation will update the /etc/services and /etc/inetd.conf files. Two new services, rpcplus and rpcplustc will be created and assigned to the port numbers you specified.

The final step of the installation process is the creation of a shared object library or the relinking of the RM/COBOL runtime, depending on what version of RM/COBOL you are using. The following paragraphs describe this process. The process is controlled by a script named rmrelink, which can be run again if there is a problem during the initial execution. The rmrelink script is automatically executed by rminstall.

The rmrelink script will look at the files in your RM/COBOL directory and determine which version of RM/COBOL you are using. If it is v7 or higher the script will create a shared object library. If it is v6 or earlier the script will relink a new runcobol.

The rmrelink script assumes that the entire RM/COBOL runtime has been installed into the directory you identify. If only part of the runtime has been installed the rmrelink script may fail, and you should install the complete RM/COBOL runtime and reexecute the rmrelink script.

If you are using RM/COBOL v7 or higher, when rmrelink successfully completes it will have created a file named libetsrpc.so and have copied it to the RM/COBOL installation directory. If you are using an earlier version of RM/COBOL, the script will have created a new runcobol and placed it in the RPC+ installation directory.

Once the relinking process is complete you may want to delete the relink subdirectory which was created underneath the RPC+ directory to perform the link. The files in that directory are no longer required.

If you have your own 'C' routines which you link with the RM/COBOL runtime, you will have to combine those with the routines inserted by RPC+. When the rmrelink script executes, it modifies the makefile and sub.c files to include the RPC+ routines. You can easily perform a diff command using the original and modified files and see the entries added by RPC+. The function declarations and LIBTABLE entries required by RPC+ are declared in the files rpcdecl.h and rpcfuncs.h. These files can be found in the relink subdirectory and the etssrc/rm subdirectory.

Acucobol

Create a new directory and extract the installations files into that directory with the following command:

```
cpio –icvBd <xxx
```

where xxx is your installation file. Then execute the installation script acuinstall.

You will be asked to enter your RPC+ license number. This will allow the server to have full functionality. You can install and use the product without a license number, but the server will only allow a single user at a time, and the clients will display demonstration messages. If you do not have an RPC+ license number now, you can add it later by placing it in the License entry of the [ServerConfig] section of the rpcplus.ini file used by the server.

You will also be asked to enter port numbers to use for the fat and thin client services. RPC+ uses TCP/IP to communicate between client and server. The client and server must both use the same port number to communicate. Be sure the numbers entered here match between the client and the server, and are not used by any other service on the server.

The installation will update the /etc/services and /etc/inetd.conf files. Two new services, rpcplus and rpcplustc will be created and assigned to the port numbers you specified.

The final step of the installation process is the relinking of the Acucobol runtime. The process is controlled by a script named acurelink, which can be run again if there is a problem during the initial execution. The acurelink script is automatically executed by acuinstall.

The acurelink script uses the files installed by the Acucobol runtime in the lib subdirectory of your Acucobol installation directory. If these files have not been installed the acurelink script will fail, and you should install the complete Acucobol runtime and reexecute the acurelink script.

The script will create a new runcbl and place it in the RPC+ installation directory as runcblrpc, so it is obvious that it has been relinked with RPC+.

Once the relinking process is complete you may want to delete the relink subdirectory which was created underneath the RPC+ plus directory to perform the link. The files in that directory are no longer required.

If you have your own ‘C’ routines that you link with the Acucobol runtime, you will have to combine those with the routines inserted by RPC+. When the acurelink script executes it modifies the Makefile, sub.c, and sub85.c files to include the RPC+ routines. You can easily perform a diff command using the original and modified files and see the entries added by RPC+. The function declarations and LIBTABLE entries required by RPC+ are declared in the files rpcdecl.h and rpcfuncs.h. These files can be found in the relink subdirectory or the etssrc/acu subdirectory.

All Languages

Configuration file location:

The execution of RPC+ on the server is controlled by the inetd superserver and the rpcstart and rpcthin scripts created in the RPC+ installation directory. These scripts execute a cd command to set the working directory before invoking the Acucobol runtime. This allows the multiple samples to each use their own rpcplus.ini file. Some of the following sections refer to entries in the rpcplus.ini file. You will find the rpcplus.ini file for each sample in the directory containing the sample program.

Changing the default port number

The port number used for RPC+ is controlled by the rpcplus and rpcplustc entries in the /etc/services file. To change the port number, edit the /etc/services file and change the port number on the rpcplus and/or rpcplustc line.

After changing the port number it is necessary to “refresh” the inetd service. This can be done with the following command:

```
kill –HUP pid
```

where pid is the process id of the inetd service.

Setting the RPC+ license number

The RPC+ license number is specified in the following entry in the [ServerConfig] section of the rpcplus.ini file:

License=xxxxxx-xxxxxx-xxxxxx

If you need to change the license number, edit this entry in the rpcplus.ini file and enter the correct value.

Configuring the environment

Each RPC+ service launches the server process by invoking a shell script. The shell script is listed in the entries added to /etc/services. These shell scripts can be edited to set environment variables, change the working directory, or perform other functions required before the server application is launched.

Chapter 2: Executing the Examples

Introduction

Once RPC+ has been installed on the client and the server, you can start running the example programs. These programs are installed in the examples subdirectory of your RPC+ installation. Each sample is in it's own unique directory, separated by application architecture and language.

Depending on the language you are using, there may be some additional setup required. Be sure and the following section describing setup issues related to any of the languages you will be using.

If the server you have installed is a Windows server, you will need to start a server daemon before executing the client. This daemon listens for the connection from the client, and then starts the server process. If you run the client without first starting the server daemon, you will get an error message indicating that the RPC+ service cannot be found. Before running the Fat Client sample, you need to execute the Fat Client Server icon on the server. This will launch the Fat Client server daemon and start it listening for requests.

To avoid having to start the server daemon manually under Windows, see Chapter 9 on Using a Windows Service.

If you installed a Unix server, it should already be listening and you do not need to start anything else. The inetd superserver, which is part of the Unix operating system, was configured automatically on installation.

MicroFocus COBOL

When using MicroFocus COBOL, it is important to have the required MicroFocus environment variables set and the MicroFocus executables located in the PATH. No special entries are required for RPC+ usage that are not required any time you run a MicroFocus program. This requirement applies to both client and server programs executed with MicroFocus COBOL.

Windows

The samples are built to execute in the .int format and use runw.exe to execute. This was chosen as a format to provide maximum portability. The samples are compatible with MicroFocus Object COBOL and with NetExpress. The samples and the compiled MicroFocus programs we supply can be compiled into .gnt or .obj format, and linked into executables. To link to an executable, simply include the rpcplus.lib file in the cbllink command.

If you link to an executable, be sure to pay attention to situations where an RPC+ program must be executed before your application code. This is the case with the Fat Client server. The rpcinimf program must be the first program specified in the cbllink command, before any of your programs. The command to link should look like this:

```
cbllink  rpcinimf+myprog1+myprog2+rpcplus.lib
```

Of course, before you do this you have to compile rpcinimf.int to rpcinimf as follows:

ccbl rpcinimf.int;

You may want to use the animator to execute the samples. This can be done by replacing runw.exe with animw.exe on the command line or in the StartupCommand of the rpcplus.ini file.

Unix

When RPC+ installs on Unix it attempts to compile the RPC+ object code and relink the MicroFocus runtime (rts32). If this process fails because the commands were not found or failed,

you will have to complete this process manually. This will require a 'C' compiler and the MicroFocus mkrts command.

If you establish the RPC+ installation directory as your working directory, you can use the following command to compile the RPC+ code:

```
cc -c rpcplus.c
```

This will create the file rpcplus.o. Then you can link this file to the MicroFocus runtime with the following command:

```
mkrts rpcplus.o
```

The mkrts command is supplied by MicroFocus and is typically installed in the following directory (assuming MicroFocus COBOL was installed in the /mfcobol directory):

```
/mfcobol/src/rts
```

Once a new rts32 containing RPC+ has been linked, it should be placed in the RPC+ installation directory.

Acucobol

Windows

To execute the samples, you must have a 32-bit Acucobol runtime for Windows included in your PATH.

Acucobol can be used in linked or unlinked form. A separate set of examples is supplied for each because the linked version allows the use of a more sophisticated programming technique. Both are compiled using the v3.10 compiler, and the compiled programs are built both with the .acu extension and with no extension. The calls made in the sample programs are made without an extension for compatibility with other languages, and the older versions of Acucobol default to using no extension, while the newer versions default to using the .acu extension.

The unlinked samples can be executed without any special setup, as long as the Acucobol runtime is located in the PATH so that wrun32.exe can be found. The linked version for Acucobol requires the Acucobol runtime to be relinked as follows:

v3.1 to v4.2

With these versions you create a replacement for wrun32.exe that includes the RPC+ routines. The files needed to relink the runtime are located in the examples\acucobol\linked\relinking subdirectory of your RPC+ installation. You have to have the Microsoft Visual C compiler to relink the runtime. To relink the runtime, simply copy the files from your Acucobol lib subdirectory into the correct RPC+ relinking directory, based on the version number of your Acucobol installation. Then go to a command prompt in that directory and type the following:

```
nmake -f wrun32rpc.mak
```

This will create a file called wrun32rpc.exe. This is an executable you will use to launch your RPC+ programs, rather than the standard wrun32.exe. Copy this file to wherever you want to execute it from, and then edit the rpcplus.ini files in the FatServer, ThinClient, and ThinServer directories. Each of these rpcplus.ini files contains a reference to wrun32. Replace this reference with the path and name of your wrun32rpc.

With some versions of Acucobol a license file is required. Acucobol supplies a wrun32.alc file, but since we are using wrun32rpc instead of wrun32.alc, the wrun32.alc license file must be copied to the name wrun32rpc.alc and placed in the same directory as wrun32rpc.exe.

v4.3 to v5.2

With these versions you create a replacement for wrun32.dll that includes the RPC+ routines. The files needed to relink the runtime are located in the examples\acucobol\linked\relinking subdirectory of your RPC+ installation. You have to have the Microsoft Visual C compiler to relink the runtime. To relink the runtime, simply copy the files from your Acucobol lib subdirectory into the correct RPC+ relinking directory, based on the version number of your Acucobol installation. Then go to a command prompt in that directory and type the following:

```
nmake -f wrun32rpc.mak
```

This will create a file called wrun32.dll. This is a modified runtime you will use to execute your RPC+ programs, rather than the standard wrun32.dll. You must place this wrun32.dll in the same directory as the wrun32.exe that you use to invoke the runtime, otherwise the relinked wrun32.dll is not found.

Unix

The Unix samples are automatically configured to run by the installation process and no further action is required.

RM/COBOL

Windows

To execute the samples you must have a 32-bit RM/COBOL runtime for Windows included in your PATH.

Unix

The Unix samples are automatically configured to run by the installation process and no further actions is required.

'C'

The 'C' samples can be executed without any additional setup. The sample projects were built using Visual C++ 6.0.

VisualBasic

To execute the Visual Basic samples, Visual Basic 5.0 or higher must be installed on the system.

Java

To execute the java samples you must have the java interpreter included in your PATH.

Chapter 3: Basics

The demand for client/server and distributed applications has placed new demands on applications. In the past, applications were monolithic and executed on a single system. Today's applications must be able to execute with part of the application on a client, and the rest on a server. Some implementations require multiple servers. Other implementations may use different languages on the client and the server.

RPC+ provides a simple solution for all these situations. With RPC+, programs running on clients can call programs executing on servers. Programs running on a server can call Cobol programs running on other servers. Data can be passed between these programs. The calls are made using standard techniques.

But, before we examine how this is accomplished, let's review the concepts of client/server and distributed computing.

Client/Server and Distributed Processing

Client/server applications combine the capabilities of two computer systems. The client computer provides the user interface and some or all of the processing. The server computer stores the application's data and may do some of the processing. The most common type of client/server application is based on an architecture where the client system handles the user interface and business logic and the server handles the data. The advantage of this type of architecture is its simplicity. The disadvantage is that the client is responsible for the business logic as well as the user interface. This is commonly called a fat client architecture, because the client is doing a lot of the work.

In a fat client architecture, calls are made from the client to the server since the client is requesting services from the server.

A more sophisticated type of client/server application is based on an architecture where the client system handles the user interface, the business logic executes on a server, and the data resides on that or another server. This architecture has the advantage of placing business logic in a central location where it can be more easily administered, has faster access to data, and is scaleable to faster systems as user demands increase without affecting the clients. This is often referred to as a thin client architecture, because the client has a limited role.

In a thin client architecture the client makes the initial connection to the server. Calls are then made from the server to the client since the server is requesting services from the server such as user interface presentation.

A third type of architecture involves the distribution of an application across multiple servers. This type of architecture is sometimes referred to as a three-tier architecture. RPC+ works well in this architecture because multiple levels of calling are supported. The server can call another server while being called by the client.

How RPC+ Works

Client/Server is based on the ability for one computer to CALL another computer. RPC+ implements this through a RemoteProgram function. This function allows a system to call another system, pass it parameters, and receive modified data when control is returned. This RemoteProgram function is available in each supported language, and is tailored to be as similar as possible to the language's native calling mechanism. Let's step through the process.

Let's begin with the client. At some point in its execution the client application wants to call a program on a server. The client application calls RPC+ and passes the name of the program it wants to call and the data to pass to the program.

RPC+ checks its configuration file to see where that program is available. Then it contacts the appropriate server with a request to execute the program.

Now we move to the server. When the server receives the request, it checks to see if it already has a conversation established with the client. If not, it starts one by spawning a server process. This server process remains dedicated to the client throughout multiple calls for optimum performance.

After ensuring the server has a conversation with the client, the server receives the data items from the client and executes the requested program. When the server program has completed execution, the server returns the updated values of the data items to the client.

When the client receives the returned data items from the server, the client program continues execution.

Note that with RPC+, multiple levels of remote program calls are supported. The server program could have called a program on a completely different server while servicing the call from the client! This would not affect the client in any way.

In most cases, RPC+ server programs require no special coding. They are called by a remote client in the same way they are called locally. Many existing programs can function as server programs without modification.

RPC+ client programs require little or no special coding. With some implementations, server programs are called with the standard syntax, just as if they were local. With other implementations, a special RPC+ function must be used to call server programs.

Application Architectures

Most of the time RPC+ is used in a concurrent architecture. In a concurrent architecture, there is a server process associated with every client process. When a client connects to the server, a new process is spawned on the server to provide services to the client. This server process persists until the client disconnects. This means there are as many processes on the server as there are clients connected to the server.

The other available type of architecture is an iterative architecture. In an iterative architecture a single server process handles requests from multiple clients. The complexity of this approach is that the server program must be written in a way to be aware of the fact that it is talking to potentially a different client on each request. It cannot, for example, store an account number selected by the client in one call to be used on the next call. That next call may well be from a different client. Instead, the server must treat each request as a completely discreet event.

Chapter 4: Configuration Options

Introduction

The behavior of RPC+ is largely controlled by options specified in a configuration file. This file is a Windows .ini file. While the Windows operating system has migrated away from .ini files, towards specifying options in the Windows registry, the .ini file remains a preferable way to configure RPC+. The options are easy to edit, without the possibility of damaging the registry, and it is easy to maintain multiple configurations.

The configuration options are generally specified in a file named rpcplus.ini. Applications which use RPC+ generally will look for rpcplus.ini file in the current working directory, then the logical Windows directory on Windows, or the /etc/directory on Unix. On Unix the rpcplus.ini file must be named in all lower case letters.

While rpcplus.ini is the default name for the configuration file, there are ways to specify a different name. On either Windows or Unix, setting an environment variable named "RPCPLUSINI" will override the default name. This can be done as follows:

RPCPLUSINI=myfile.ini

On Windows, the file name can also be overridden by including the following on the command line tail for the application:

Myapplication.exe RPCPLUSINI=myfile.ini

[AutoVersionControl]

The [AutoVersionControl] section is used to configure the operation of the automatic version control features. These features allow files to automatically be transferred between client and server, keeping both systems in synchronization.

Destination

Default: FALSE

Indicates whether or not this system will function as a destination for automatic version control files. Setting this option to TRUE indicates that this system will accept files from the remote system when they connect, if the remote system will send them. The client will accept all files sent by the server, but can control their location through the AutoVersionDestFiles section. A file will only be received if its size does not match that on the remote system or its date/time stamp is newer on the remote system. The remote system must have Source=TRUE.

Sample: Destination=TRUE

DownloadMessageText

Default: names of files being transferred

Specifies text to be placed in the download message window. If no text is supplied, the names of the individual files will be displayed as they are transferred.

Sample: DownloadMessageText=Please Wait…

DownloadMessageTitle

Default: Updating Files

Controls the title displayed in the window created by setting ShowDownloadMessage to TRUE.

Sample: DownloadMessageTitle=My Title

QuitOnMissingSourceFile

Default: FALSE

Setting this option to TRUE will cause the application to quit if one of the files listed in the VersionControlSourceFiles section could not be opened for transfer.

Sample: QuitOnMissingSourceFile=TRUE

ShowDownloadMessage

Restrictions: Windows Only

Default: FALSE

Setting this option to TRUE will cause a message to be displayed while files are being transferred via Automatic Version Control. The text of the message can be configured through other entries.

Sample: ShowDownloadMessage=TRUE

Source

Default: FALSE

Indicates whether or not this system will function as a source of automatic version control files. Setting this option to TRUE indicates that it will transfer files to the remote system when they connect, if the remote system will accept them. The files to be sent are indicated in the AutoVersionControlSourceFiles section. A file will only be sent if its size does not match that on the remote system or its date/time stamp is older on the remote system. The remote system must have Destination=TRUE.

Sample: Source=TRUE

SourceMasterVersionFile

This entry can be used to name one of the files listed in the AutoVersionControlSourceFiles section as the master indicator of whether version checking should be performed on the rest of the files listed in the AutoVersionControlSourceFiles section. If a large number of files are listed in this section, a substantial amount of data must be transferred just to check the version information. This can cause a perceptible delay in startup, particularly over low speed dial-up connections. Specifying a SourceMasterVersionFile will cause version checking to be performed on this file first. If this file does not need updating, no version checking will be performed on the rest of the files listed in the AutoVersionControlSourceFiles section. If this file does need updating, it will be transferred, and then all the files listed will also be version checked and updated if needed.

If you use a SourceMasterVersionFile, you will always want to update this file after any other files have changed, giving it a date/time stamp later than any others. If this file is “older” than any of the files listed in the AutoVersionControlSourceFiles section, those “newer” files will not be transferred.

Sample: SourceMasterVersionFile=VERSION.DAT

VersionList

This entry can be used to name a file, which will contain a specific version label for files transferred using AutoVersionControl. This allows the decision on transferring files to be based on version numbers you assign, rather than file size and system date/time stamping. The format of the entries in this file is discussed in the [VERSION] section documentation.

NOTE: This entry must include a fully qualified pathname for the file.

Sample: VersionList=c:\myapp\MYVERSIONS.INI

[AutoVersionControlDestFiles]

Allows selection of the location to place files transferred from the server. Any file not named in this section will be placed in the working directory. The first part of the entry is the name of the file. The second part of the entry is the location the file should be placed in when downloaded.

Samples:

FILE1=C:\INFILES\

FILE2=C:\INFILES2\

[AutoVersionControlSourceFiles]

Indicates the files that should be transferred to the remote system when a connection is established. The file name is indicated first, followed by an "=", then an optional path. This separation of the filename from the path allows the remote system to override the location in it's own configuration information. Files without a path specified will be retrieved from the working directory.

Samples:

FILE1=C:\INFILES\

FILE2=C:\INFILES2\

[CharacterConversion]

The entries in this section control the translation of characters between the local and remote system.

Enabled

Default: FALSE

This entry specifies whether or not character translation is to be performed.

Sample: Enabled=TRUE

Characters 0 – 255

Each character in the character set can be used to specify a translation. Any characters for which a translation is not defined will not be altered. The entries for each character can be made using decimal or hex notation. The entry on the left side of the equal sign is the local value. The entry on the right of the equal sign is the remote value.

Samples:

0x43=0x55

65=43

0x33=74

85=0x32

[ClientConfig]

AutoDisconnect

Default: FALSE

Setting this option to TRUE will cause the client to automatically disconnect from the server after every remote call.

Sample: AutoDisconnect=TRUE

CommLog

This entry requires a file name. If a file name is specified, all communications associated with client behavior will be recorded in the file. This is useful in some debugging situations.

Sample: CommLog=client.dat

CmdLineDllName

This entry specifies the name of a DLL that should be used to encrypt parameters passed on the command line to RPCPlusThinClient. This is an advanced feature that is described in detail in the section titled User Validation.

Sample: CmdLineDllName=MyEncryption.DLL

CmdLineDllFunc

This entry specifies the name of a function in the CmdLineDllName DLL that should be used to decode encrypted parameters passed on the command line to RPCPlusThinClient. This is an advanced feature that is described in detail in the section titled User Validation.

Sample: CmdLineDllFunc=DecodeParameters

DataCompression

Default: TRUE

Determines whether or not the server will allow data compression in its communications with clients. For any type of data compression to be utilized this option must be set to TRUE for both the client and server. In this way, either side of the conversation can suppress compression. This option being set to TRUE will trigger a first level of basic data compression. More advanced compression can be added by setting the DeltaCompression and PersistentDelta options to TRUE as well. However, setting DataCompression to FALSE will eliminate all compression regardless of how other options are set.

Sample: DataCompression=TRUE

DataLog

Specifies a file name to which compression statistics should be written. This gives an opportunity to view the actual amount of data being transferred with and without compression. This will only be written to if DataCompression=TRUE. LogDetail or LogTotalInterval must also be specified.

Sample: DataLog=datalog.txt

DefaultExtension

Specifies the default extension to be used for program names in the RemotePrograms section. Rather then placing the extension on each program name in the RemotePrograms section it can be specified here and RPC+ will add the extension when it loads the list of program names. If the REMOTEPROGRAM function is used to call a program and no extension is included in the program name, RPC+ will append the default extension to the name at this time as well.

Sample: DefaultExtension=.COB

DefaultServer

Specifies a default server used to call any programs listed in the RemotePrograms section, which do not specify a server.

Sample: DefaultServer=167.142.103.220

DeltaCompression

Default: TRUE

Restrictions: Requires DataCompression=TRUE

Determines whether or not a compression algorithm, which compares the data, passed to the server to the data it will return should be used. If set to TRUE, the server can return just the data that has been modified. This requires that DataCompression and DeltaCompression are both set to TRUE on the server and client. Setting this option to TRUE will generally result in higher compression then DataCompression alone. It cannot result in less compression.

Sample: DeltaCompression=FALSE

DetailedMessages

Default: TRUE

If this entry is set to TRUE error messages will include additional information besides just the error description. If set to FALSE, only the error description will be displayed.

Sample: DetailedMessages=FALSE

LogActivity

Default: FALSE

Restrictions: Requires setting LogFileName.

If LogActivity is set to TRUE, a record of any errors encountered will be created and written to the file specified in the LogFileName entry.

Sample: LogActivity=TRUE

LogBuffer

Default: FALSE

Restrictions: Requires setting DataLog

If LogBuffer is set to TRUE, a record of buffered argument usage will be written to the file named in the DataLog entry.

Sample: LogBuffer=TRUE

LogDetail

Default: FALSE

Setting this option to TRUE will result in statistics being written to the DataLog file for each remote program call received. Otherwise, only summary information for the number of calls specified in the LogTotalInterval will be written.

Sample: LogDetail=TRUE

LogFileName

This entry names the file to which any communication or RPC+ related errors are written.

Sample: LogFileName=rpcplus.log

LogTotal

Default: FALSE

Setting this option to TRUE will result in summary statistics being written to the DataLog file after the number of calls specified in LogTotalInterval has been reached.

Sample: LogTotal=TRUE

LogTotalInterval

Restrictions: Requires LogTotal=TRUE

Setting this option indicates the number of calls, which should be combined to provide summary statistics in the DataLog.

Sample: LogTotalInterval=10

MessageOnError

Default: TRUE

Specifies whether or not RPC+ should generate an error message when it encounters an error. This can be used in conjunction with the StopOnError entry to give an application complete control over error handling.

Sample: MessageOnError=FALSE

Password

Indicates a password that will be supplied to the server. If the server specifies a password this value will be compared to the server password and the connection will only succeed if the passwords match. If the server does not specify a password, this value will be ignored. Case is not significant in the password. Length is limited to 30 characters.

Sample: Password=MyPassword

PersistentDelta

Restrictions: Requires DataCompression=TRUE and DeltaCompression=TRUE

Default: FALSE

Determines whether or not a compression algorithm, which compares the data passed to the server in previous calls to the data passed in this call, should be used. If set to TRUE, the server can optimize data transmission from one call to the next. This requires that DataCompression, DeltaCompression, and PersistentDelta are all set to TRUE on the server and client. Setting this option to TRUE will generally result in higher compression then DataCompression and DeltaCompression. It cannot result in less compression.

Sample: PersistentDelta=TRUE

Port

Specifies the port number that will be used to connect to the server.

Sample: Port=5000

RetryOnConnect

Default: FALSE

Setting this option to TRUE will cause RPC+ to automatically retry if a server connection fails up to the time specified in the TimeOut entry.

Sample: RetryOnConnect=TRUE

ServerCountMax

Default: Number of entries in RemotePrograms section

The default behavior of RPC+ is to count the number of programs listed in the RemotePrograms section of the RPCPlus.INI file and assumes there could be no more servers used than there are programs. This was a valid assumption until the implementation of the SETPROGRAMSERVER function. By calling this function a single program can be called on multiple servers. This created the possibility of exceeding the number of servers RPC+ expects to be used.

If the SETPROGRAMSERVER function is used to specify more servers than RPC+ expects the SETPROGRAMSERVER function will fail, returning a GIVING value or RETURN-CODE of 1. This condition can be avoided by specifically telling RPC+ the maximum number of servers, which will be used through this configuration option.

This value will only be used if it is greater than the number of programs listed in the RemotePrograms section.

Sample: ServerCountMax=20

ShowConfiguration

Default: FALSE

Setting ShowConfiguration to TRUE will cause the configuration information loaded by RPC+ to be written to the file specified in the LogFileName entry. This allows the user to confirm that the desired configuration options are actually being loaded and utilized.

Sample: ShowConfiguration=TRUE

ShowErrorFunction

Default: TRUE

The ShowErrorFunction entry determines whether or not the name of the function in which an error occurred is included in the display/written error message. The function name can be useful for debugging purposes, but may be distracting to end users of an application utilizing RPC+.

Sample: ShowErrorFunction=TRUE

StopOnError

Restrictions: Will not stop a Merant COBOL, VisualBasic, or 'C' application

Default: TRUE

Controls one aspect of how RPC+ handles errors encountered when making a call. The default behavior of RPC+ is to display an error message and halt the application. If the user wants to handle the errors inside their application, they can set this option to FALSE. A GIVING clause or RETURN-CODE variable can then be used to determine the success or failure of the call and respond appropriately.

Sample: StopOnError=FALSE

TimeOut

Setting a non-zero value for this option will cause the client to only wait the specified time, in seconds, for a response from the server when making a remote call. If RetryOnConnect=TRUE this value will also control how long the client will retry a connection request.

Sample: TimeOut=60

UseKeepAlive

Default: TRUE

Setting this option to FALSE will prevent the TCP/IP KEEPALIVE socket option from being used. KEEPALIVE periodically tests the socket connection during periods of inactivity. This is the only way that some connection terminations can be detected.

For this option to work, the TCP/IP parameters KeepAliveTime and KeepAliveInterval must be set in the system registry or Unix kernel.

Sample: UseKeepAlive=FALSE

[ManualVersionControl]

DefaultDir

Indicates the default location to be used for files that are referenced when using the manual version control checking function. If no entry is included, these files will be accessed in the working directory.

Sample: DefaultDir=c:\myfiles\

[MessageConfig]

Entries in this section override the default text associated with error conditions. They have the format:

Name=Value

where Name is the internally defined error message name dictated by RPC+ and Value is the user defined text that should be displayed to describe the error.

Sample: ArgCountWrong=The calling program supplied the wrong number of arguments to the function.

To prevent a particular error message from being displayed, assign it the special value shown in the following sample:

Name=*SUPPRESS_MESSAGE*

ArgCountWrong

Default: The specified argument count does not match the supplied parameters

ArgSizeWrong

Default: One of the parameters in the call to the following program is smaller than specified

AttemptToCopyRejected

Default: Server rejected attempted copy

BadServerPassword

Default: Server rejected connection because server password was invalid

BuiltWithoutLogin

Default: RPC+ built without User Login routines

CantAcceptOnSocket

Default: Could not accept on socket

CantAccessVCSF

Default: Cannot access the following file listed in AutoVersionControlSourceFiles

CantAddServer

Default: Cannot add server to list

CantBindSocket

Default: Could not bind socket

CantConnect

Default: Could not connect to RPC+ service on the following server

CantConnectSocket

Default: Could not connect socket

CantConvertIPToASCII

Default: Could not convert the IP address to ASCII

CantCreateFile

Debug: Cannot create the following file

CantCreateSocket

Default: Could not create socket

CantDownloadFile

Default: Cannot download the following file

CantGetClientIP

Default: The socket function getpeername failed to retrieve the client IP

CantGetHost

Default: The sockets function gethostbyname failed for the following server

CantGetSockName

Default: Could not getsockname on socket

CantInitSockets

Default: Cannot initialize sockets

CantIoctlOnSocket

Default: Could not ioctl on socket

CantListenOnSocket

Default: Could not listen on socket

CantOpenFile

Default: Cannot open the following file

CantReachHost

Default: TCP/IP cannot locate the following server

CantReceiveOnSocket

Default: Could not receive on socket

CantReopenIni

Default: Cannot reopen the RPC+ initialization file

CantSendOnSocket

Default: Could not send on socket

CantStartServerProcess

Default: Could not start server process

CantWriteToFile

Default: Cannot write to following file

ChecksumFailure

Default: Checksum failure on remote call

ClientCantReceive

Default: Cannot receive data from server

ClientCantSend

Default: Cannot send data to server

ConnectFailed

Default: Could not connect to RPC+ service on the following server

DupSocketFailed

Default: Could not duplicate socket

EncryptionKeyMismatch

Default: Encryption key from client did not match server

InactiveLimit

Default: Server Inactivity Time Out limit exceeded - process will be terminated

InternalError

Default: An internal inconsistency was detected

InvalidArg

Default: The following argument is invalid

InvalidBufferArgs

Default: The RPCPLUS.INI file entry for the following program specifies too many arguments

InvalidLicenseFormat

Default: The RPC+ license code has an invalid format

InvalidLogin

Default: Server rejected connection because user name and password were invalid

InvalidPackage

Default: The following was received in the handshake instead of a valid package

InvalidResponse

Default: The following data was received instead of a valid response

InvalidServerPassword

Default: The following was received instead of the valid server password

InvalidSync

Default: Invalid synchronization code

InvalidVersion

Default: The following was received in the handshake instead of a valid version

LicensingError

Default: Server rejected connection because it encountered an error in the licensing routines

LoginFailed

Default: Login failed for user

MemoryAllocation

Default: Cannot allocate memory

MemoryLock

Default: Cannot lock memory

MissingIniEntry

Default: The RPC+ initialization file is missing the following required entry

NoResponse

Default: No response received from server

NoResponseToSend

Default: No response to send

NoServerSpecified

Default: The following program does not specify a server and no DefaultServer entry is present

NotEnoughArgs

Default: Not enough arguments supplied for function

RebootRequired

Default: Reboot required to update files

SemCountWrong

Default: Semaphore count wrong

SemCreateFailed

Default: Semaphore create failed

SemCtlFailed

Default: semctl failed for the following

SemDecFailed

Default: Semaphore decrement failed

SemGetFailed

Default: semget failed

SemValueMismatch

Default: Semaphore values do not match

ServerCantReceive

Default: Cannot receive data from client

ServerCantSend

Default: Cannot send data to client

ServerLost

Default: Connection to the server was lost

ServerProgramFailed

Default: This program could not be executed on the server

SetSockOptFailed

Default: The sockets function setsockopt failed for the following option

StartupFailed

Default: Startup command failed

StatFailed

Default: Could not stat the following file

Status

Default: Status

Title

Default: RPC+ Error

TooManyConnections

Default: Server rejected connection because it has the maximum number of active connections

TooManyUsers

Default: Server rejected connection because it has the maximum number of active users

UnlicensedServer

Default: Server rejected connection because the server is not licensed

UserCountExceeded

Default: An attempt was made to exceed the licensed user count

VersionMismatch

Default: Server rejected connection because it is not using the same version of RPC+

[ProgramName]

The [ProgramName] section is used to configure specific parameters for a single remote program. The section name is the name of the remote program, as listed in the [RemotePrograms] section.

As such, this section can occur as many times as there are programs listed in the [RemotePrograms] section.

BufferArgCount

Default: 0

This entry defines the number of arguments passed to a program that is using argument buffering.

Sample: BufferArgCount=3

BufferArgs

This entry indicates which arguments should be buffered. Each argument that should be buffered is listed, separated by a comma. Values are 1 relative, meaning the first argument is 1.

Sample: BufferArgs=1,5,7

BufferKeyArg

Default: 1

This entry specifies which argument contains the key value used to store and retrieve the buffered arguments. Values are 1 relative, meaning the first argument is 1.

Sample: BufferKeyArg=2

BufferKeySize

Default: 0

This entry specifies the size of the key value used to store and retrieve the buffered arguments.

Sample: BufferKeySize=10

BufferKeyStart

Default: 1

This entry specifies the location of the first byte of key data in the argument specified to contain the key value used to store and retrieve the buffered arguments. Values are 1 relative, meaning the first byte is 1.

Sample: BufferKeyStart=5

[RemotePrograms]

Specifies the names of programs that will be executed remotely. The first item in each entry is the name of the program to be executed remotely. The name must be terminated with an "=". Following the "=" is an optional server name or IP address specifying where the program should be executed. If no server is indicated, the program will be assigned to the server specified in the DefaultServer entry.

If a server name is used, rather than an IP address, the name is resolved to an IP address by the TCP/IP stack using the hosts file.

Programs listed here may also be further described in a section named with the program name. See the [MyProgram] section documentation for details.

Sample: [RemotePrograms]

```
PROGRAM1.COB=
PROGRAM2=
PROGRAM3=127.0.0.1
PROGRAM4=myserver
```

[ServerName]

The [ServerName] section is used to configure specific parameters for a single server.

Port

The Port entry is used to indicate that RPC+ connections being made to this server should use this port number, rather than the default specified in the [ClientConfig] sections port entry.

Sample: Port=5001

[ServerConfig]

AllowCopyFrom

Default=TRUE

Setting the AllowCopyFrom entry to FALSE will prevent the client from being able to copy any files from the server using the RemoteCopy function.

Sample: AllowCopyFrom=FALSE

AllowCopyTo

Default=TRUE

Setting the AllowCopyTo entry to FALSE will prevent the client from being able to copy any files to the server using the RemoteCopy function.

Sample: AllowCopyTo=FALSE

AllowRemoteCmd

Default=TRUE

Setting the AllowRemoteCmd entry to FALSE will prevent the client from being able to execute commands on the server using the RemoteCmd or RemoteShellExecute functions.

Sample: AllowRemoteCmd=FALSE

CheckPasswords

Default=TRUE

This entry can be used to suppress password validation on Unix when RPC+ is setting the user and group ids for the server process. This option is provided because the routines used on Unix for validating user names and passwords are non-standard, and may not be available or properly implemented on all Unix systems.

Sample: CheckPasswords=FALSE

ClientArgUsage

Restrictions: Only available when using RPCPlusThinClient with a Windows server.

Default Value: On Windows – COMMAND
On Unix – PARAM

Indicates how parameters passed by the thin client startup program should be interpreted. If set to COMMAND the parameters are used as the command line on the server to start the server's COBOL process, thereby overriding the contents of the StartupCommand entry. If set to PARAM, the parameters are simply stored inside the server runtime and can be retrieved at any time using the GetClientArgs function.

On Unix this parameter is treated as if it is ALWAYS set to PARAM. The Unix architecture of utilizing the inetd process to invoke the rpcstart script does not allow for the command line to be

dynamically overriden. On Unix the client command line arguments are always available for retrieval using the GetClientArgs function.

Samples:

ClientArgUsage=PARAM

ClientArgUsage=COMMAND

CloseStdOut

Restrictions: Unix Only

Default: TRUE

Setting this option controls whether or not stdout is closed on Unix. stdout is closed by default to prevent unwanted DISPLAY statements from corrupting the client/server data stream. If stdout is not closed, any information written by the application to stdout will be displayed on the client as unexpected data.

Sample: CloseStdOut=TRUE

CloseStdIn

Restrictions: Unix Only

Default: FALSE

Setting this option controls whether or not stdin is closed on Unix.

Sample: CloseStdOut=TRUE

CmdArgs

Restrictions: Windows Only

Specifies alphanumeric data that should be appended to the command line used to start the server process. This data will be placed on the command line after the socket number and working directory (if specified). This is intended for use with thin client implementations on the server where rpcinit is not the initial program executed. It is supported for all languages, but is particularly important when using RM/COBOL because arguments on an RM/COBOL command line must be placed inside the A=" " notation which is generated by the RPC+ server.

Sample: CmdArgs=arg1 arg2 arg3

CobolType

Restrictions: Windows only

CobolType is used to specify what COBOL is being used so that differences between compilers can be adjusted for automatically. Possible values are ACUCOBOL, RMCOBOL, and MFCOBOL.

Sample: CobolType=MFCOBOL

CommLog

This entry requires a file name. If a file name is specified, all communications associated with server behavior will be recorded in the file. This is useful in some debugging situations.

Sample: CommLog=server.dat

DataCompression

Default: TRUE

Determines whether or not the server will allow data compression in its communications with clients. For any type of data compression to be utilized this option must be set to TRUE for both the client and server. In this way, either side of the conversation can suppress compression. This option being set to TRUE will trigger a first level of basic data compression. More advanced compression can be added by setting the DeltaCompression and PersistentDelta options to TRUE

as well. However, setting DataCompression to FALSE will eliminate all compression regardless of how other options are set.

Sample: DataCompression=FALSE

DataLog

Specifies a file name to which compression statistics should be written. This gives an opportunity to view the actual amount of data being transferred with and without compression. This will only be written to if DataCompression=TRUE

Sample: DataLog=datalog.txt

DeltaCompression

Restrictions: Requires DataCompression=TRUE

Default: FALSE

Determines whether or not a compression algorithm that compares the data passed to the server to the data it will return should be used. If set to TRUE, the server can return just the data that has been modified. This requires that DataCompression and DeltaCompression are both set to TRUE on the server and client. Setting this option to TRUE will generally result in higher compression then DataCompression alone. It cannot result in less compression.

Sample: DeltaCompression=TRUE

DetailedMessages

Default: TRUE

If this entry is set to TRUE, error messages will include additional information besides just the error description. If set to FALSE, only the error description will be displayed.

Sample: DetailedMessages=FALSE

InactiveLimit

Specifies the maximum amount of time a server will wait for an already connected client to make another remote program call. If this time limit is exceeded, the client will be automatically disconnected. When the client makes their next remote call a new connection will be established. This allows the server to limit memory usage by unused server processes. The value is in seconds.

Note that this disconnection of a client by a server will result in all server programs being reset to their initial state when the next connection is established. This option should not be used by applications that depend on the persistence of the state of server programs.

Sample: InactiveLimit=60

License

This entry specifies the RPC+ license number, required in most situations.

Sample: XXXXXX-XXXXXX-XXXXXX

LogActivity

Default: FALSE

Restrictions: Requires setting LogFileName.

If LogActivity is set to TRUE, a record of any errors encountered will be created and written to the file specified in the LogFileName entry.

Sample: LogActivity=TRUE

LogBuffer

Default: FALSE

Restrictions: Requires setting DataLog

If LogBuffer is set to TRUE, a record of buffered argument usage will be written to the file named in the DataLog entry.

Sample: LogBuffer=TRUE

LogDetail

Default: FALSE

Setting this option to TRUE will result in statistics being written to the DataLog file for each remote program call received. Otherwise, only summary information for the number of calls specified in the LogTotalInterval will be written.

Sample: LogDetail=TRUE

LogFileName

This entry names the file to which any communication or RPC+ related errors are written.

Sample: LogFileName=rpcplus.log

LogTotal

Default: FALSE

Setting this option to TRUE will result in summary statistics being written to the DataLog file after the number of calls specified in LogTotalInterval has been reached.

Sample: LogTotal=TRUE

LogTotalInterval

Restrictions: Requires LogTotal=TRUE

Setting this option indicates the number of calls that should be combined to provide summary statistics in the DataLog.

Sample: LogTotalInterval=10

MessageOnError

Default: TRUE

Specifies whether or not RPC+ should generate an error message when it encounters an error.

Sample: MessageOnError=FALSE

Password

Indicates a password that the server will require clients to send to connect to the server. This parameter allows a server administrator to change the password and restrict access only to users with the new password. Case is not significant in the password. Length is limited to 30 characters.

Sample: Password=MyPassword

PersistentDelta

Restrictions: Requires DataCompression=TRUE and DeltaCompression=TRUE

Default: TRUE

Determines whether or not a compression algorithm that compares the data passed to the server in previous calls to the data passed in this call should be used. If set to TRUE, the server can optimize data transmission from one call to the next. This requires that DataCompression, DeltaCompression, and PersistentDelta are all set to TRUE on the server and client. Setting this option to TRUE will generally result in higher compression then DataCompression and DeltaCompression. It cannot result in less compression.

Sample: PersistentDelta=TRUE

Port

Specifies the port number that will be used by RPCPlusServer.exe, the RPCPlus service, or in some cases by a Unix application using RPC+, to listen for remote program calls. This entry is required under Windows, and for Unix applications started from a command line. It is not required for Unix applications started by inetd.

Any client who will connect to the server with RPC+ must have this same number specified in their RPCPlus.ini file. The request from the client must use the same port number as the server, or the client cannot connect to the server.

On Unix this value is ignored during normal execution. This is because inetd, a standard listens for the client's requests and inetd gets the port number to listen on from the /etc/services file, not from rpcplus.ini.

However, if you start an RPC+ server program from the command line on Unix, rather than letting inetd start it, RPC+ will look for this entry in the rpcplus.ini file to determine what port number to listen on. This is quite often done during debugging.

Sample: Port=5000

ProgramCase

Restrictions: Unix only

Default: 0

Indicates the case translation that should be applied to the names of remote programs when they are called. This is provided so that Windows programs can make program calls without having to match the case of the programs on the Unix server. The Unix server can be in complete control of naming conventions in this way. Possible values are:

Value	Description
0	No translation
1	Force upper case
2	Force lower case

Sample: ProgramCase=2

ShowConfiguration

Default: FALSE

Setting ShowConfiguration to TRUE will cause the configuration information loaded by RPC+ to be written to the file specified in the LogFileName entry. This allows the user to confirm that the desired configuration options are actually being loaded and utilized.

Sample: ShowConfiguration=TRUE

ShowErrorFunction

Default: TRUE

The ShowErrorFunction entry determines whether or not the name of the function in which an error occurred is included in the display/written error message. The function name can be useful for debugging purposes, but may be distracting to end users of an application utilizing RPC+.

Sample: ShowErrorFunction=TRUE

StartupCommand

Specifies the command that will be used to start the application when a connection request is received. This is utilized only on Windows. It is used by both RPCPlusServer.exe, and the RPC+ service. On Unix this entry is ignored because the rpcstart script contains the command line.

It is generally a good idea to include full path names on all commands and files in the StartupCommand to make it insensitive to the working directory. Command line options can be included in the command line.

Sample: StartupCommand=c:\mydir\runcmd.exe c:\mydir\rpcinit.cob

Umask

Under unix the umask affects the permissions given to files created by a process. The umask can be set in the script used to start the server program. However, if you are using the login capabilities to change the user id a process runs under, you may also wish to set a different umask for different users. This can be done with an umask entry in the [UserName] section. However, in case a user name is used for which no umask entry is found in the [UserName] section, you can place a default entry here.

Sample: Umask=022

UseEncryption

Default: TRUE

This entry controls whether or not encryption is used on the communications between client and server. If set to TRUE, a 64-bit encryption technique utilizing dynamically assigned server generated keys is used.

Sample: UseEncryption=FALSE

UseKeepAlive

Default: TRUE

Setting this option to FALSE will prevent the TCP/IP KEEPALIVE socket option from being used. KEEPALIVE periodically tests the socket connection during periods of inactivity. This is the only way that some connection terminations can be detected.

For this option to work, the TCP/IP parameters KeepAliveTime and KeepAliveInterval must be set in the system registry or Unix kernel.

Sample: UseKeepAlive=FALSE

UseLogin

Default: FALSE

Setting this value to TRUE will cause RPCPlusServer.exe or RPCPlusService.exe to use login features to validate the user name, password, and domain passed on the RPCPlusThinClient.exe command line.

Sample: UseLogin=TRUE

WorkingDir

Restrictions: Does not work with RM/COBOL.

Specifies the working directory from which server programs should be executed. If no value is entered, the working directory will be the working directory of the process that starts the server process. In Unix, this would be the home directory of the user specified in the /etc/inetd.conf file. In Windows, this would be the working directory associated with the icon associated with the RPC+ server daemon.

The working directory has implications for locating both called subprograms and data files and should be carefully established.

Sample: WorkingDir=\mydir

[UserName]

This section is used to specify values that are specific to individual users. They are only used when the UseLogin entry in the [ServerConfig] section is set to TRUE.

Umask

Under unix the umask affects the permissions given to files created by a process. The umask can be set in the script used to start the server program. However, if you are using the login capabilities to change the user id a process runs under, you may also wish to set a different umask for different users. This can be done with an umask entry in the [UserName] section. If you want to supply a default value to use when a user name is used that does not have an explicit entry in this section, you can place a default entry in the [ServerConfig] section.

Sample: Umask=022

[UserParameters]

This section is used to supply values which are substituted for parameters on the command line when invoking RPCPlusThinClient. This allows a "generic" icon to be created for launching an application with actual values to be defined in the rpcplus.ini file. The parameter names must be enclosed in {} in the command line to be eligible for replacement. If a parameter is specified in the command line that is not present in this section, the parameter is deleted. Replaceable parameters and non-replaceable parameters can both be used on the command line together. The entries in this section are of the format:

PARAM1=VALUE1

PARAM3=VALUE3

If RPCPlusThinClient is invoked with the following command line:

RPCPlusThinClient {PARAM1} STATICVALUE {PARAM2} {PARAM3}

The command line as retrieved by the server program will contain:

RPCPlusThinClient VALUE1 STATICVALUE VALUE3

[Versions]

While all the other entries described in this chapter are entries in the RPCPlus.ini file, the entries in this section are made in a separate file. This file is the one named in the VersionList entry of the [AutoVersionControl] section. The entries take the form of filename=version, where filename is the name of the file and version is any character string you want to use to describe the version. The format of the version is not important. A check is made to see if the version strings match exactly between client and server, and if they do not match, the file will be transferred.

Sample:

MyFile1=v1.00

MyFile2=Version 3.45

Chapter 5: Functions

Acucobol

GetClientAddr

This function can by executed on the server to retrieve the IP address of the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTADDR" USING CLIENT-IP.`

where

CLIENT-IP is an alphanumeric data item of at least 15 characters.

GetClientArgs

This function can by executed on the server in a thin client architecture to retrieve any parameters that were part of the command line when the thin client executable was started. It can only be called by the server portion of a thin client application, not the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTARGS" USING ARG-DATA.`

where

ARGDATA is an alphanumeric data item of at least 255 characters.

Restrictions: This function may only be used in a thin client architecture where RPCPlusThinClient.exe was invoked on the client. The function may only be called by programs on the server.

RemoteCmd

The RemoteCmd function provides the ability to execute a command on the remote system. There are two formats for the RemoteCmd function. One format uses individual parameters and is only available when using a relinked Acucobol runtime. The other format uses a single parameter block and is available when using an unlinked or relinked runtime.

Format 1 – Individual parameters

```
Sample: CALL "REMOTECMD" USING RMT-STATUS RMT-SERVER RMT-
DIRECTORY RMT-COMMAND RMT-WAIT-FLAG
```

Where the parameters are defined as follows:

RMT-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

RMT-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

RMT-DIRECTORY is an alphanumeric literal or data item that contains the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

RMT-COMMAND is an alphanumeric literal or data item that contains the command to be executed, complete with parameters if desired.

RMT-WAIT-FLAG is a one byte alphanumeric literal or data item that contains indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Format 2 – Parameter block

Sample: `CALL "REMOTECMD" USING REMOTECMD-WS.`

where

REMOTECMD-WS is defined as follows:

```
01 REMOTECMD-WS.

03 RMT-STATUS            PIC 9(4).

03 RMT-SERVER            PIC X(80).

03 RMT-DIRECTORY         PIC X(128).

03 RMT-COMMAND           PIC X(128).

03 RMT-WAIT-FLAG         PIC X.
```

and

RMT-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

RMT-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

RMT-DIRECTORY is the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

RMT-COMMAND is the command to be executed, complete with parameters if desired.

RMT-WAIT-FLAG indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RemoteCopy

The RemoteCopy function provides the ability to copy files between the local and remote systems. There are two formats for the RemoteCopy function. One format uses individual parameters and is only available when using a relinked Acucobol runtime. The other format uses a single parameter block and is available when using an unlinked or relinked runtime.

Format 1 – Individual parameters

Sample: `CALL "REMOTECOPY" USING RC-STATUS RC-SERVER RC-LOCAL-NAME RC-REMOTE-NAME RC-DIRECTION.`

Where the parameters are defined as follows:

RC-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

RC-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the copying should be performed with. In a thin client architecture to copy a file to or from the client, this parameter should be set to spaces.

RC-LOCAL-NAME is an alphanumeric literal or data item that contains the name of the file on the local machine, including a path if desired.

RC-REMOTE-NAME is an alphanumeric literal or data item that contains the name of the file on the remote machine, including a path if desired.

RC-DIRECTION is a one byte alphanumeric literal or data item that indicates the direction of the copy operation. A value of "I" or "I" requests copying the file IN to the local machine. A value of "O" or "o" requests copying the file OUT to the remote machine.

RC-CRLF is a one byte alphanumeric literal or data item that contains indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of "Y" or "y" indicates that CRLF characters should be modified. A value of "N" or "n" indicates they should not be altered.

Format 2 – Parameter block

Sample: `CALL "REMOTECOPY" USING REMOTECOPY-WS.`

where

REMOTECOPY-WS is defined as follows:

```
01 REMOTECOPY-WS.
03 RC-STATUS            PIC 9(4).
03 RC-SERVER            PIC X(80).
03 RC-LOCAL-NAME        PIC X(128).
03 RC-REMOTE-NAME       PIC X(128).
03 RC-DIRECTION         PIC X.
03 RC-CRLF              PIC X.
```

and

RC-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error

3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

RC-SERVER is the name or IP address of the server the copying should be performed with. In a thin client architecture, to copy a file to or from the client, this parameter should be set to spaces.

RC-LOCAL-NAME is the name of the file on the local machine, including a path if desired.

RC-REMOTE-NAME is the name of the file on the remote machine, including a path if desired.

RC-DIRECTION indicates the direction of the copy operation. A value of "I" or "I" requests copying the file IN to the local machine. A value of "O" or "o" requests copying the file OUT to the remote machine.

RC-CRLF indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of "Y" or "y" indicates that CRLF characters should be modified. A value of "N" or "n" indicates they should not be altered.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RemoteProgram

The RemoteProgram function provides the functionality of the standard COBOL CALL statement, but allows it to operate between two systems. The RemoteProgram function provides the ability to pass parameters to, execute, and return parameters from, a program on a remote server. There are three formats of the RemoteProgram function available with Acucobol. Two of these are only available when using relinked Acucobol runtimes.

Format 1 – Transparent

Sample: CALL "MYPROGRAM" USING PARAM1 PARAM2 …

Where MYPROGRAM has been listed in the [RemotePrograms] section of the rpcplus.ini file. Including the name of the program in this section tells RPC+ to intercept the CALL statement and automatically route it to the remote system. The syntax is the same as the standard COBOL CALL statement.

Format 2 – Simplified

Sample: `CALL "REMOTEPROGRAM" USING PROGRAM-NAME PARAM1 PARAM2 …`

Where the parameters are defined as follows:

PROGRAM-NAME is an alphanumeric literal or data item specifying the name of the program to invoke on the remote system.

PARAM1 PARAM2… are literals or data items that should be passed to the remote program. No parameters are required.

Format 3 – Standard

When using the RPC+ REMOTEPROGRAM function, the copy file, RPCPLUS.WS, should be included in your program. It includes the declaration of a parameter block that contains all the variables needed by this function.

To see how the REMOTEPROGRAM function works let's look at the code required to call the program SAMPLE3 using the REMOTEPROGRAM function:

```
MOVE "SAMPLE3" TO RPC-PROGRAM.
MOVE 0 TO RPC-ARG-COUNT.
CALL "REMOTEPROGRAM" USING RPC-CONTROL.
```

All the RPC- variables are declared in RPCPLUS.WS. RPC-PROGRAM-NAME indicates the name of the remote program to be called. RPC-ARGUMENT-COUNT indicates the number of data items to be passed, in this case, zero.

Usually you will want to pass data to the server program. Let's examine the code required to call the program SAMPLE4 and pass it the following three data items:

```
01 DATA-1        PIC X(10).

01 DATA-2        PIC 9(5)

01 DATA-3.
   03 ITEM-1   PIC X(10).
   03 ITEM-2   PIC 9(5).

MOVE "SAMPLE4" TO RPC-PROGRAM.
MOVE 3 TO RPC-ARG-COUNT.
MOVE 10 TO RPC-ARG-SIZE (1).
MOVE 5 TO RPC-ARG-SIZE (2).
MOVE 15 TO RPC-ARG-SIZE (3).
CALL "REMOTEPROGRAM" USING RPC-CONTROL DATA-1 DATA-2
   DATA-3.
```

This time RPC-ARG-COUNT is set to 3, the number of arguments to be passed. Each entry in the array, RPC-ARG-SIZE, is loaded with the size of the corresponding argument. The size of the first argument is loaded in RPC-ARG-SIZE (1), the second in RPC-ARG-SIZE (2).

It is strongly recommended that when using the "REMOTEPROGRAM" function the copy file RPCPLUS.WS be used to declare the RPC- variables. Future versions of RPC+ may expand this parameter block and using the copy file will make migration to new versions easier.

The number of parameters to be passed is limited to 20 individual items, and the total size of these items cannot exceed 60,000 bytes.

RemoteProgramNoWait

The RemoteProgramNoWait function works just like the RemoteProgram function, except the caller does not wait for a response from the remote system. This allows the caller to proceed with other processing without waiting for a response. This does, however, prevent the caller from receiving any modified data from the remote system.

RemoteShellExecute

The RemoteShellExecute function provides the ability to execute the Windows ShellExecute function on a remote Windows system. There are two formats for the RemoteShellExecute function. One format uses individual parameters and is only available when using a relinked Acucobol runtime. The other format uses a single parameter block and is available when using an unlinked or relinked runtime.

Format 1 – Individual Parameters

Sample: CALL “REMOTESHELLEXECUTE” USING SE-STATUS SE-SERVER SE-VERB SE-FILE SE-PARAMS SE-DIR SE-STATE SW-WAIT.

Where the parameters are defined as follows:

RMT-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

SE-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

SE-VERB is an alphanumeric literal or data item that contains the action to be performed. Common verbs are “OPEN”, “PRINT”, and “EDIT”. The verb must correlate with the file that is specified, and the type of document the file is.

SE-FILE is an alphanumeric literal or data item that contains the name of the file the verb should be executed on.

SE-PARAMS is an alphanumeric literal or data item that specifies the command line parameters to be passed to SE-FILE, if SE-FILE indicates an executable.

SE-DIR is an alphanumeric literal or data item that specifies the default or working directory

SE-STATE is a four digit unsigned numeric field that specifies the window state. These are the Windows constants SW_HIDE, SW_SHOW, etc.

SE-WAIT is a one byte alphanumeric literal or data item that indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the document to be closed. A value of “Y” or “y” indicates that RPC+ should wait for the document to be closed. Any other value indicates that control should be returned immediately.

Format 2 – Parameter block

Sample: `CALL "REMOTESHELLEXECUTE" USING REMOTESHELLEXECUTE-WS.`

where

REMOTESHELLEXECUTE-WS is defined as follows:

```
01 REMOTESHELLEXECUTE-WS.
03 SE-STATUS         PIC 9(4).
03 SE-SERVER         PIC X(80).
03 SE-VERB           PIC X(128).
03 SE-FILE           PIC X(128).
03 SE-PARAMS         PIC X(128).
03 SE-DIR            PIC X(128).
03 SE-STATE          PIC 9(4).
03 SE-WAIT           PIC X.
```

SE-STATUS is a value set by the function to indicate success or failure Success is indicated by a value of 0. Other possible values are:

Value	Description
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

SE-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

SE-VERB indicates the action to be performed. Common verbs are “OPEN”, “PRINT”, and “EDIT”. The verb must correlate with the file that is specified, and the type of document the file is.

SE-FILE is the file the verb should be executed on.

SE-PARAMS specifies the command line parameters to be passed to SE-FILE, if SE-FILE indicates an executable.

SE-DIR specifies the default or working directory

SE-STATE specifies the window state. These are the Windows constants SW_HIDE, SW_SHOW, etc.

SE-WAIT indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the document to be closed. A value of "Y" or "y" indicates that RPC+ should wait for the document to be closed. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

SetProgramServer

The SetProgramServer function provides the ability to specify at runtime what server a program should be executed on. It is called as follows:

```
CALL "SETPROGRAMSERVER" USING PROGRAM-NAME SERVER.
```

Where the parameters are defined as follows:

PROGRAM-NAME is an alphanumeric string or data item specifying the name of the program that should be reassigned. When using an unlinked Acucobol runtime it is necessary to terminate this field with LOW-VALUES.

SERVER is an alphanumeric string or data item that contains the name or IP address of the server the program should be assigned to. When using an unlinked Acucobol runtime it is necessary to terminate this field with LOW-VALUES.

The program referenced in this function must be declared in the [RemotePrograms] section of the rpcplus.ini file.

ShutdownRPC

This function is used to terminate all connections and free resources associated with remote programs. It should be called before an application exits.

The SHUTDOWNRPC function is executed as follows:

```
CALL "SHUTDOWNRPC".
```

StartServer

This function is used on the server in thin client architecture. It must be called to initialize RPC+. This function should pass the command line as an argument. If the argument is a single uppercase D, RPC+ will attempt to initialize in direct mode with terminal support. This will cause the terminal to freeze while RPC+ waits for a connection.

Sample: `CALL "STARTSERVER" USING CMD-LINE.`

where

CMD-LINE is an alphanumeric data item that contains the contents of the command line tail. This field should be at least 80 characters in length.

VersionCheck

The VersionCheck function provides the ability to test a local file against a remote file and see if the remote file needs to be updated to match the local file. There are two formats for the VersionCheck function. One format uses individual parameters and is only available when using a relinked Acucobol runtime. The other format uses a single parameter block and is available when using an unlinked or relinked runtime.

Format 1 – Individual Parameters

```
Sample: CALL "VersionCheck" using VC-STATUS VC-SERVER VC-LOCAL-
FILE VC-FILE-STATUS VC-FULL-LOCAL-FILE-NAME VC-FULL-REMOTE-FILE-
NAME
```

Where the parameters are defined as follows:

VC-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
3	Server refused request
7	Cannot open the local file

VC-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

VC-LOCAL-FILE is an alphanumeric literal or data item that contains the name of the file on the local machine that should be checked against the remote machine.

VC-FILE-STATUS is a one byte alphanumeric data item that is set to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

VC-FULL-LOCAL-FILE-NAME is an alphanumeric data item that returns the fully qualified pathname of the file on the local machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

VC-FULL-REMOTE-FILE-NAME is an alphanumeric data item that returns the fully qualified pathname of the file on the remote machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Format 2 – Parameter block

Sample: `CALL "VersionCheck" using VC-WS.`

where

VC-WS is defined as follows:

```
01 VC-WS.
   03 VC-STATUS                PIC 9(4).
   03 VC-SERVER                PIC X(80).
   03 VC-LOCAL-FILE            PIC X(128).
   03 VC-FILE-STATUS           PIC X.
   03 VC-FULL-LOCAL-NAME       PIC X(128).
   03 VC-FULL-REMOTE-NAME      PIC X.
```

VC-STATUS contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
7	Cannot open the local file

VC-SERVER the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

VC-LOCALFILE contains the name of the file on the local machine that should be checked against the remote machine.

VC-FILESTATUS is set to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

VC-FULLLOCALFILENAME is the fully qualified pathname of the file on the local machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

VC-FULLREMOTEFILENAME is the fully qualified pathname of the file on the remote machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

MicroFocus COBOL

GetClientAddr

This function can by executed on the server to retrieve the IP address of the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTADDR" USING CLIENT-IP.`

where

CLIENT-IP is an alphanumeric data item of at least 15 characters.

GetClientArgs

This function can by executed on the server in a thin client architecture to retrieve any parameters that were part of the command line when the thin client executable was started. It can only be called by the server portion of a thin client application, not the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTARGS" USING ARG-DATA.`

where

ARGDATA is an alphanumeric data item of at least 255 characters.

Restrictions: This function may only be used in a thin client architecture where RPCPlusThinClient.exe was invoked on the client. The function may only be called by programs on the server.

RemoteCmd

The RemoteCmd function provides the ability to execute a command on the remote system.

Sample: `CALL "REMOTECMD" USING REMOTECMD-WS.`

where

REMOTECMD-WS is defined as follows:

```
01 REMOTECMD-WS.
03 RMT-STATUS           PIC 9(4).
03 RMT-SERVER           PIC X(80).
03 RMT-DIRECTORY        PIC X(128).
03 RMT-COMMAND          PIC X(128).
03 RMT-WAIT-FLAG        PIC X.
```

and

RMT-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description

0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

RMT-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

RMT-DIRECTORY is the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

RMT-COMMAND is the command to be executed, complete with parameters if desired.

RMT-WAIT-FLAG indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RemoteCopy

The RemoteCopy function provides the ability to copy files between the local and remote systems.

Sample: `CALL "REMOTECOPY" USING REMOTECOPY-WS.`

where

REMOTECOPY-WS is defined as follows:

```
01 REMOTECOPY-WS.
   03 RC-STATUS                PIC 9(4).
   03 RC-SERVER                PIC X(80).
   03 RC-LOCAL-NAME            PIC X(128).
   03 RC-REMOTE-NAME           PIC X(128).
   03 RC-DIRECTION             PIC X.
   03 RC-CRLF                  PIC X.
```

and

RC-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

RC-SERVER is the name or IP address of the server the copying should be performed with. In a thin client architecture, to copy a file to or from the client, this parameter should be set to spaces.

RC-LOCAL-NAME is the name of the file on the local machine, including a path if desired.

RC-REMOTE-NAME is the name of the file on the remote machine, including a path if desired.

RC-DIRECTION indicates the direction of the copy operation. A value of "I" or "I" requests copying the file IN to the local machine. A value of "O" or "o" requests copying the file OUT to the remote machine.

RC-CRLF indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of "Y" or "y" indicates that CRLF characters should be modified. A value of "N" or "n" indicates they should not be altered.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RemoteProgram

The RemoteProgram function provides the ability to pass parameters to, execute, and return parameters from, a program on a remote server. When using the RPC+ REMOTEPROGRAM function, the copy file, RPCPLUS.WS, should be included in your program. It includes the declaration of a parameter block that contains all the variables needed by this function.

To see how the REMOTEPROGRAM function works let's look at the code required to call the program SAMPLE3 using the REMOTEPROGRAM function:

```
MOVE "SAMPLE3" TO RPC-PROGRAM.
MOVE 0 TO RPC-ARG-COUNT.
CALL "REMOTEPROGRAM" USING RPC-CONTROL.
```

All the RPC- variables are declared in RPCPLUS.WS. RPC-PROGRAM-NAME indicates the name of the remote program to be called. RPC-ARGUMENT-COUNT indicates the number of data items to be passed, in this case, zero.

Usually you will want to pass data to the server program. Let's examine the code required to call the program SAMPLE4 and pass it the following three data items:

```
01 DATA-1      PIC X(10).
01 DATA-2      PIC 9(5)
01 DATA-3.
   03 ITEM-1   PIC X(10).
   03 ITEM-2   PIC 9(5).
MOVE "SAMPLE4" TO RPC-PROGRAM.
MOVE 3 TO RPC-ARG-COUNT.
MOVE 10 TO RPC-ARG-SIZE (1).
MOVE 5 TO RPC-ARG-SIZE (2).
MOVE 15 TO RPC-ARG-SIZE (3).
CALL "REMOTEPROGRAM" USING RPC-CONTROL DATA-1 DATA-2
   DATA-3.
```

This time RPC-ARG-COUNT is set to 3, the number of arguments to be passed. Each entry in the array, RPC-ARG-SIZE, is loaded with the size of the corresponding argument. The size of the first argument is loaded in RPC-ARG-SIZE (1), the second in RPC-ARG-SIZE (2).

It is strongly recommended that when using the "REMOTEPROGRAM" function the copy file RPCPLUS.WS be used to declare the RPC- variables. Future versions of RPC+ may expand this parameter block and using the copy file will make migration to new versions easier.

The number of parameters to be passed is limited to 20 individual items, and the total size of these items cannot exceed 60,000 bytes.

RemoteProgramNoWait

The RemoteProgramNoWait function works just like the RemoteProgram function, except the caller does not wait for a response from the remote system. This allows the caller to proceed with other processing without waiting for a response. This does, however, prevent the caller from receiving any modified data from the remote system.

RemoteShellExecute

The RemoteShellExecute function provides the ability to execute the Windows ShellExecute function on a remote Windows system.

Sample: `CALL "REMOTESHELLEXECUTE" USING REMOTESHELLEXECUTE-WS.`

where

REMOTESHELLEXECUTE-WS is defined as follows:

```
01 REMOTESHELLEXECUTE-WS.
   03 SE-STATUS         PIC 9(4).
   03 SE-SERVER         PIC X(80).
   03 SE-VERB           PIC X(128).
   03 SE-FILE           PIC X(128).
   03 SE-PARAMS         PIC X(128).
   03 SE-DIR            PIC X(128).
   03  E-STATE          PIC 9(4).
   03 SE-WAIT           PICX.
```

SE-STATUS is a value set by the function to indicate success or failure. Success is indicated by a value of 0. Other possible values are:

Value	Description
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

SE-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

SE-VERB indicates the action to be performed. Common verbs are “OPEN”, “PRINT”, and “EDIT”. The verb must correlate with the file that is specified, and the type of document the file is.

SE-FILE is the file the verb should be executed on.

SE-PARAMS specifies the command line parameters to be passed to SE-FILE, if SE-FILE indicates an executable.

SE-DIR specifies the default or working directory

SE-STATE specifies the window state. These are the Windows constants SW_HIDE, SW_SHOW, etc.

SE-WAIT indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the document to be closed. A value of “Y” or “y” indicates that

RPC+ should wait for the document to be closed. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

SetProgramServer

The SetProgramServer function provides the ability to specify at runtime what server a program should be executed on. It is called as follows:

```
CALL "SETPROGRAMSERVER" USING PROGRAM-NAME SERVER.
```

Where the parameters are defined as follows:

PROGRAM-NAME is a null terminated alphanumeric data item specifying the name of the program that should be reassigned.

SERVER is a null terminated alphanumeric data item that contains the name or IP address of the server the program should be assigned to.

The program referenced in this function must be declared in the [RemotePrograms] section of the cobolrpc.ini file.

ShutdownRPC

This function is used to terminate all connections and free resources associated with remote programs. It should be called before an application exits.

The SHUTDOWNRPC function is executed as follows:

```
CALL "SHUTDOWNRPC".
```

StartServer

This function is used on the server in thin client architecture. It must be called to initialize RPC+. This function should pass the command line as an argument. If the argument is a single uppercase D, RPC+ will attempt to initialize in direct mode with terminal support. This will cause the terminal to freeze while RPC+ waits for a connection.

Sample: `CALL "STARTSERVER" USING CMD-LINE.`

where

CMD-LINE is an alphanumeric data item that contains the contents of the command line tail. This field should be at least 80 characters in length.

VersionCheck

The VersionCheck function provides the ability to test a local file against a remote file and see if the remote file needs to be updated to match the local file. The function is used as follows:

```
CALL "VersionCheck" using VC-WS.
```

where

VC-WS is defined as follows:

```
01 VC-WS.
   03 VC-STATUS              PIC 9(4).
   03 VC-SERVER              PIC X(80).
   03 VC-LOCAL-FILE          PIC X(128).
   03 VC-FILE-STATUS         PIC X.
   03 VC-FULL-LOCAL-NAME     PIC X(128).
   03 VC-FULL-REMOTE-NAME    PIC X.
```

VC-STATUS contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
7	Cannot open the local file

VC-SERVER the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

VC-LOCALFILE contains the name of the file on the local machine that should be checked against the remote machine.

VC-FILESTATUS is set to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

VC-FULLLOCALFILENAME is the fully qualified pathname of the file on the local machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

VC-FULLREMOTEFILENAME is the fully qualified pathname of the file on the remote machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RM/COBOL

RM/COBOL uses a specialized format to describes parameters in a CALL statement. This format provides extensive information describing the parameters passed in the CALL. This is a significant advantage over the way CALLs are implemented in other languages. Because of the parameter information built into the CALL, RPC+ functions can determine the number, size, and type of the parameters passed in calls to RPC+ functions. This provides you a simpler way of invoking these functions, and prevents you from having to supply information on the number, size, and type of parameters. Because of this, the format of the functions described in this section is somewhat simpler than that described in the sections applying to other languages.

GetClientAddr

This function can by executed on the server to retrieve the IP address of the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTADDR" USING CLIENT-IP.`

where

CLIENT-IP is an alphanumeric data item of at least 15 characters.

GetClientArgs

This function can by executed on the server in a thin client architecture to retrieve any parameters that were part of the command line when the thin client executable was started. It can only be called by the server portion of a thin client application, not the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: `CALL "GETCLIENTARGS" USING ARG-DATA.`

where

ARGDATA is an alphanumeric data item of at least 255 characters.

Restrictions: This function may only be used in a thin client architecture where RPCPlusThinClient.exe was invoked on the client. The function may only be called by programs on the server.

RemoteCmd

The RemoteCmd function provides the ability to execute a command on the remote system. There are two formats for the RemoteCmd function. One format uses individual parameters and the other format uses a single parameter block.

Format 1 – Individual parameters

```
Sample: CALL "REMOTECMD" USING RMT-STATUS RMT-SERVER RMT-
DIRECTORY RMT-COMMAND RMT-WAIT-FLAG
```

Where the parameters are defined as follows:

RMT-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

RMT-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

RMT-DIRECTORY is an alphanumeric literal or data item that contains the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

RMT-COMMAND is an alphanumeric literal or data item that contains the command to be executed, complete with parameters if desired.

RMT-WAIT-FLAG is a one byte alphanumeric literal or data item that contains indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Format 2 – Parameter block

Sample: `CALL "REMOTECMD" USING REMOTECMD-WS.`

where

REMOTECMD-WS is defined as follows:

```
01 REMOTECMD-WS.
03 RMT-STATUS            PIC 9(4).
03 RMT-SERVER            PIC X(80).
03 RMT-DIRECTORY         PIC X(128).
03 RMT-COMMAND           PIC X(128).
03 RMT-WAIT-FLAG         PIC X.
```

and

RMT-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success

1	Socket error
2	Synchronization error
3	Server refused request
5	System error

RMT-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

RMT-DIRECTORY is the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

RMT-COMMAND is the command to be executed, complete with parameters if desired.

RMT-WAIT-FLAG indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architect

RemoteCopy

The RemoteCopy function provides the ability to copy files between the local and remote systems. There are two formats for the RemoteCopy function. One format uses individual parameters and the other format uses a single parameter block.

Format 1 – Individual parameters

Sample: `CALL "REMOTECOPY" USING RC-STATUS RC-SERVER RC-LOCAL-NAME RC-REMOTE-NAME RC-DIRECTION.`

Where the parameters are defined as follows:

RC-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

RC-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the copying should be performed with. In a thin client architecture to copy a file to or from the client, this parameter should be set to spaces.

RC-LOCAL-NAME is an alphanumeric literal or data item that contains the name of the file on the local machine, including a path if desired.

RC-REMOTE-NAME is an alphanumeric literal or data item that contains the name of the file on the remote machine, including a path if desired.

RC-DIRECTION is a one byte alphanumeric literal or data item that indicates the direction of the copy operation. A value of "I" or "I" requests copying the file IN to the local machine. A value of "O" or "o" requests copying the file OUT to the remote machine.

RC-CRLF is a one byte alphanumeric literal or data item that contains indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of "Y" or "y" indicates that CRLF characters should be modified. A value of "N" or "n" indicates they should not be altered.

Format 2 – Parameter block

Sample: `CALL "REMOTECOPY" USING REMOTECOPY-WS.`

where

REMOTECOPY-WS is defined as follows:

```
01 REMOTECOPY-WS.
03 RC-STATUS            PIC 9(4).
03 RC-SERVER            PIC X(80).
03 RC-LOCAL-NAME        PIC X(128).
03 RC-REMOTE-NAME       PIC X(128).
03 RC-DIRECTION         PIC X.
03 RC-CRLF              PIC X.
```

and

RC-STATUS is a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

RC-SERVER is the name or IP address of the server the copying should be performed with. In a thin client architecture, to copy a file to or from the client, this parameter should be set to spaces.

RC-LOCAL-NAME is the name of the file on the local machine, including a path if desired.

RC-REMOTE-NAME is the name of the file on the remote machine, including a path if desired.

RC-DIRECTION indicates the direction of the copy operation. A value of "I" or "I" requests copying the file IN to the local machine. A value of "O" or "o" requests copying the file OUT to the remote machine.

RC-CRLF indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of "Y" or "y" indicates that CRLF characters should be modified. A value of "N" or "n" indicates they should not be altered.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

RemoteProgram

The RemoteProgram function provides the ability to pass parameters to, execute, and return parameters from, a program on a remote server. There are two formats of the RemoteProgram function available with RM/COBOL.

Format 1 – Simplified

Sample: `CALL "REMOTEPROGRAM" USING PROGRAM-NAME PARAM1 PARAM2 …`

Where the parameters are defined as follows:

PROGRAM-NAME is an alphanumeric literal or data item specifying the name of the program to invoke on the remote system.

PARAM1 PARAM2… are literals or data items that should be passed to the remote program. No parameters are required.

Format 2 – Standard

When using the RPC+ REMOTEPROGRAM function, the copy file, RPCPLUS.WS, should be included in your program. It includes the declaration of a parameter block that contains all the variables needed by this function.

To see how the REMOTEPROGRAM function works let's look at the code required to call the program SAMPLE3 using the REMOTEPROGRAM function:

```
MOVE "SAMPLE3" TO RPC-PROGRAM.
MOVE 0 TO RPC-ARG-COUNT.
CALL "REMOTEPROGRAM" USING RPC-CONTROL.
```

All the RPC- variables are declared in RPCPLUS.WS. RPC-PROGRAM-NAME indicates the name of the remote program to be called. RPC-ARGUMENT-COUNT indicates the number of data items to be passed, in this case, zero.

Usually you will want to pass data to the server program. Let's examine the code required to call the program SAMPLE4 and pass it the following three data items:

```
01 DATA-1      PIC X(10).

01 DATA-2      PIC 9(5)

01 DATA-3.
   03 ITEM-1   PIC X(10).
   03 ITEM-2   PIC 9(5).

MOVE "SAMPLE4" TO RPC-PROGRAM.
MOVE 3 TO RPC-ARG-COUNT.
MOVE 10 TO RPC-ARG-SIZE (1).
MOVE 5 TO RPC-ARG-SIZE (2).
MOVE 15 TO RPC-ARG-SIZE (3).
CALL "REMOTEPROGRAM" USING RPC-CONTROL DATA-1 DATA-2
   DATA-3.
```

This time RPC-ARG-COUNT is set to 3, the number of arguments to be passed. Each entry in the array, RPC-ARG-SIZE, is loaded with the size of the corresponding argument. The size of the first argument is loaded in RPC-ARG-SIZE (1), the second in RPC-ARG-SIZE (2).

It is strongly recommended that when using the "REMOTEPROGRAM" function the copy file RPCPLUS.WS be used to declare the RPC- variables. Future versions of RPC+ may expand this parameter block and using the copy file will make migration to new versions easier.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

The number of parameters to be passed is limited to 20 individual items, and the total size of these items cannot exceed 60,000 bytes. However, if only a single item is passed, it may be up to 1,000,000 bytes in size.

RemoteProgramNoWait

The RemoteProgramNoWait function works just like the RemoteProgram function, except the caller does not wait for a response from the remote system. This allows the caller to proceed with other processing without waiting for a response. This does, however, prevent the caller from receiving any modified data from the remote system.

RemoteShellExecute

The RemoteShellExecute function provides the ability to execute the Windows ShellExecute function on a remote Windows system. There are two formats for the RemoteShellExecute function. One format uses individual parameters and the other format uses a single parameter block.

Format 1 – Individual Parameters

Sample: CALL "REMOTESHELLEXECUTE" USING SE-STATUS SE-SERVER SE-VERB SE-FILE SE-PARAMS SE-DIR SE-STATE SW-WAIT.

Where the parameters are defined as follows:

SE-STATUS is a numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

SE-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

SE-VERB is an alphanumeric literal or data item that contains the action to be performed. Common verbs are "OPEN", "PRINT", and "EDIT". The verb must correlate with the file that is specified, and the type of document the file is.

SE-FILE is an alphanumeric literal or data item that contains the name of the file the verb should be executed on.

SE-PARAMS is an alphanumeric literal or data item that specifies the command line parameters to be passed to SE-FILE, if SE-FILE indicates an executable.

SE-DIR is an alphanumeric literal or data item that specifies the default or working directory

SE-STATE is a four digit unsigned numeric field that specifies the window state. These are the Windows constants SW_HIDE, SW_SHOW, etc.

SE-WAIT is a one byte alphanumeric literal or data item that indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the document to be closed. A value of "Y" or "y" indicates that RPC+ should wait for the document to be closed. Any other value indicates that control should be returned immediately.

Format 2 – Parameter block

Sample: `CALL "REMOTESHELLEXECUTE" USING REMOTESHELLEXECUTE-WS.`

where

REMOTESHELLEXECUTE-WS is defined as follows:

```
01 REMOTESHELLEXECUTE-WS.
03 SE-STATUS         PIC 9(4).
03 SE-SERVER         PIC X(80).
03 SE-VERB           PIC X(128).
```

```
03 SE-FILE          PIC X(128).
03 SE-PARAMS        PIC X(128).
03 SE-DIR           PIC X(128).
03 SE-STATE         PIC 9(4).
03 SE-WAIT          PIC X.
```

SE-STATUS is a value set by the function to indicate success or failure Success is indicated by a value of 0. Other possible values are:

Value	Description
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

SE-SERVER is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

SE-VERB indicates the action to be performed. Common verbs are “OPEN”, “PRINT”, and “EDIT”. The verb must correlate with the file that is specified, and the type of document the file is.

SE-FILE is the file the verb should be executed on.

SE-PARAMS specifies the command line parameters to be passed to SE-FILE, if SE-FILE indicates an executable.

SE-DIR specifies the default or working directory

SE-STATE specifies the window state. These are the Windows constants SW_HIDE, SW_SHOW, etc.

SE-WAIT indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the document to be closed. A value of “Y” or “y” indicates that RPC+ should wait for the document to be closed. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

SetProgramServer

The SetProgramServer function provides the ability to specify at runtime what server a program should be executed on.

Sample: `CALL "SETPROGRAMSERVER" USING PROGRAM-NAME SERVER.`

Where the parameters are defined as follows:

PROGRAM-NAME is an alphanumeric literal or data item specifying the name of the program that should be reassigned.

SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the program should be assigned to.

The program referenced in this function must be declared in the [RemotePrograms] section of the cobolrpc.ini file.

ShutdownRPC

This function is used to terminate all connections and free resources associated with remote programs. It should be called before an application exits.

The SHUTDOWNRPC function is executed as follows:

```
CALL "SHUTDOWNRPC".
```

StartServer

This function is used on the server in thin client architecture. It must be called to initialize RPC+. This function should pass the command line as an argument. If the argument is a single uppercase D, RPC+ will attempt to initialize in direct mode with terminal support. This will cause the terminal to freeze while RPC+ waits for a connection.

Sample: `CALL "STARTSERVER" USING CMD-LINE.`

where

CMD-LINE is an alphanumeric data item that contains the contents of the command line tail. This field should be at least 80 characters in length.

VersionCheck

The VersionCheck function provides the ability to test a local file against a remote file and see if the remote file needs to be updated to match the local file. There are two formats for the VersionCheck function. One format uses individual parameters and the other format uses a single parameter block.

Format 1 – Individual Parameters

```
Sample: CALL "VersionCheck" using VC-STATUS VC-SERVER VC-LOCAL-
FILE VC-FILE-STATUS VC-FULL-LOCAL-FILE-NAME VC-FULL-REMOTE-FILE-
NAME
```

Where the parameters are defined as follows:

VC-STATUS is a 4 digit unsigned display numeric field that contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
3	Server refused request
7	Cannot open the local file

VC-SERVER is an alphanumeric literal or data item that contains the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

VC-LOCAL-FILE is an alphanumeric literal or data item that contains the name of the file on the local machine that should be checked against the remote machine.

VC-FILE-STATUS is a one byte alphanumeric data item that is set to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

VC-FULL-LOCAL-FILE-NAME is an alphanumeric data item that returns the fully qualified pathname of the file on the local machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

VC-FULL-REMOTE-FILE-NAME is an alphanumeric data item that returns the fully qualified pathname of the file on the remote machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Format 2 – Parameter block

Sample: `CALL "VersionCheck" using VC-WS.`

where

VC-WS is defined as follows:

```
01 VC-WS.
```

```
03 VC-STATUS                PIC 9(4).
03 VC-SERVER                PIC X(80).
03 VC-LOCAL-FILE            PIC X(128).
03 VC-FILE-STATUS           PIC X.
03 VC-FULL-LOCAL-NAME       PIC X(128).
03 VC-FULL-REMOTE-NAME      PIC X.
```

VC-STATUS contains a value set by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
7	Cannot open the local file

VC-SERVER the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

VC-LOCALFILE contains the name of the file on the local machine that should be checked against the remote machine.

VC-FILESTATUS is set to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

VC-FULLLOCALFILENAME is the fully qualified pathname of the file on the local machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

VC-FULLREMOTEFILENAME is the fully qualified pathname of the file on the remote machine. Returning this value from the VersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

Visual Basic

VBGetClientAddr

The VBGetClientAddr function provides the ability to retrieve the IP address of the client. This function may be called by any program in the run unit on the server and may be called more than once.

Sample: Status = VBGetClientAddr(Address)

where the parameters are defined as follows:

Dim Status

Dim Address as String * 15

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Failure

Address returns the IP address of the client.

VBGetClientArgs

The VBGetClientArgs function provides the ability to retrieve the command line arguments passed to RPCPlusThinClient.exe. This function can by executed on the server in a thin client architecture. It can only be called by the server portion of a thin client application, not the client.

This function may be called by any program in the run unit on the server and may be called more than once.

Sample: Status = VBGetClientArgs(Args)

where the parameters are defined as follows:

Dim Status

Dim Args as String * 255

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Failure

Args returns the arguments passed on the command line to RPCPlusThinClient.

VBRemoteCommand

The VBRemoteCommand function provides the ability to execute a command on the remote system.

Sample: Status = VBRemoteCommand(Server, Directory, Command, Waitflag)

where the parameters are defined as follows:

Dim Status

Dim Server as String * x

Dim Directory as String * x

Dim Command as String * x

Dim Waitflag as String * 1

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

Server is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

Directory is the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

Command is the command to be executed, complete with parameters if desired.

Waitflag indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of "Y" or "y" indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

VBRemoteCopy

The VBRemoteCopy function provides the ability to copy files between the local and remote systems.

Sample: Status = `RemoteCopy(Server, LocalName, RemoteName, Direction, CrLf);`

where the parameters are defined as follows:

Dim Status

Dim Server as String * x

Dim LocalName as String * x

Dim RemoteName as String * x

Dim Direction as String * 1

Dim CrLf as String * 1

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

Server is an alphanumeric literal or data item that contains the name or IP address of the server the copying should be performed with. In a thin client architecture to copy a file to or from the client, this parameter should be set to spaces.

LocalName is an alphanumeric literal or data item that contains the name of the file on the local machine, including a path if desired.

RemoteName is an alphanumeric literal or data item that contains the name of the file on the remote machine, including a path if desired.

Direction is a one byte alphanumeric literal or data item that indicates the direction of the copy operation. A value of “I” or “I” requests copying the file IN to the local machine. A value of “O” or “o” requests copying the file OUT to the remote machine.

CrLf is a one byte alphanumeric literal or data item that contains indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of “Y” or “y” indicates that CRLF characters should be modified. A value of “N” or “n” indicates they should not be altered.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

VBRemoteProgram

The VBRemoteProgram function provides the ability to pass parameters to, execute, and return parameters from, a program on a remote server.

Sample: Status = VBRemoteProgram(Program, Data, Datasize)

where the parameters are defined as follows:

Dim Status

Dim Program as String * x

Dim Data as String * x

Dim Datasize

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Could not connect to server
2	Could not execute requested program on server

Program is the name of the program to invoke on the remote system.

Data is the data that should be passed to the remote program.

Datasize is the size in bytes of the data to be passed.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

The data to be passed may be up to 1,000,000 bytes in size.

VBRemoteProgramNoWait

The VBRemoteProgramNoWait function works just like the VBRemoteProgram function, except the caller does not wait for a response from the remote system. This allows the caller to proceed with other processing without waiting for a response. This does, however, prevent the caller from receiving any modified data from the remote system.

VBSetProgramServer

The VBSetProgramServer function provides the ability to specify at runtime what server a program should be executed on.

Sample: Status = VBSetProgramServer(Program, Server)

where the parameters are defined as follows:

Dim Status

Dim Program as String * x

Dim Server as String * x

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Failure

Program is the name of the program that should be reassigned.

Server contains the name or IP address of the server the program should be assigned to.

The program referenced in this function must be declared in the [RemotePrograms] section of the cobolrpc.ini file.

VBShutdownRPC

The VBShutdownRPC function terminates all connections and frees resources associated with remote programs. It should be called before an application exits.

Sample: Call VBShutdownRPC

VBStartServer

The VBStartServer function is used on the server in thin client architecture. It must be called to initialize RPC+. This function should pass the command line as an argument. If the argument is a single uppercase D, RPC+ will attempt to initialize in direct mode with terminal support. This will cause the terminal to freeze while RPC+ waits for a connection.

Sample: Status = VBStartServer(CmdLine)

where the parameters are defined as follows:

Dim Status

Dim CmdLine as String * 80

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Failure

CmdLine contains the command line tail.

VBVersionCheck

The VBVersionCheck function provides the ability to test a local file against a remote file and see if the remote file needs to be updated to match the local file. The function is used as follows:

```
Sample: Status = VBVersionCheck(Server, FileName, FileStatus,
FullLocalFileName, FullRemoteFileName)
```

where the parameters are defined as follows:

Dim Status

Dim Server as String * x

Dim FileName as String * x

Dim FileStatus as String * 1

Dim FullLocalFileName as String * x

Dim FullRemoteFileName as String * x

and Status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
3	Server refused request
7	Cannot open the local file

Server contains the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

FileName contains the name of the file on the local machine that should be checked against the remote machine.

FileStatus is set by the function to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

FullLocalFileName returns the fully qualified pathname of the file on the local machine. Returning this value from the VBVersionCheck function makes it easy to use the file with theVBRemoteCopy function if the file needs to be copied.

FullRemoteFileName returns the fully qualified pathname of the file on the remote machine. Returning this value from the VBVersionCheck function makes it easy to use the file with the RemoteCopy function if the file needs to be copied.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

‘C’

CallRPCProgram

The CallRPCProgram function provides the ability to pass parameters to, execute, and return parameters from, a program on a remote server.

On Windows:

Sample: status = CallRPCProgram(Program, &hData, &argsize);

where the parameters are defined as follows:

int status;

char * Program;

HANDLE hData;

long argsize;

On Unix:

Sample: result = CallRPCProgram(Program, &hData, &argsize);

where the parameters are defined as follows:

int result;

char * Program

char * pData;

long argsize;

status is returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Could not connect to server
2	Could not execute requested program on server

Program is the name of the program to invoke on the remote system.

For Windows: hData is a memory handle allocated with GlobalAlloc containing the data that should be passed to the remote program.

For Unix: pData is a char * allocated with malloc containing the data that should be passed to the remote program.

argsize is the size in bytes of the data to be passed, and returns the size of the data returned.

The data to be passed may be up to 1,000,000 bytes in size. The value of hData/pData and argsize may be modified by the function.

ConnectToRPCServerEx

The ConnectToRPCServerEx function provides the ability to connect to a server and prepares for subsequent communication. This is typically used with thin client architectures, since the function does not interact with the CRemoteProgram function used for remote calls. Instead, it is used to prepare for receiving function calls back from the server.

Sample: hServer = ConnectToRPCServerEx(server, port, lpszCmdLine);

where the parameters are defined as follows:

For Windows:

HANDLE hServer;

For Unix:

int hServer;

For both systems:

char * server;

int port;

char * lpszCmdLine;

hServer returns a value that is used to identify the server in subsequent functions. If the function fails a value of 0 will be returned.

Port contains the port number to use to connect to the server.

lpszCmdLine is a pointer to the command line arguments.

These may be processed by the server, but an empty string can be passed.

CRPCShowError

The CRPCShowError provides the ability to display or log a message using RPC+'s internal error handling, just like the internal RPC+ messages.

Sample: CRPCShowError(msg);

where the parameters are defined as follows:

char * msg;

msg is a char * pointing to the message to be displayed.

CRemoteCommand

The CRemoteCommand function provides the ability to execute a command on the remote system.

Sample: status = CRemoteCommand(server, directory, command, waitflag)

where the parameters are defined as follows:

int status;

char * server;

char * directory;

char * command;

char waitflag;

and status is returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
5	System error

server is the name or IP address of the server the command should be executed on. In a thin client architecture, to execute the command on the client, this parameter should be set to spaces.

directory is the directory on the remote system from which the command should be executed. This provides the ability to establish a specific working directory for the command execution. It is not intended to be the path to the command file itself.

command is the command to be executed, complete with parameters if desired.

waitflag indicates whether control should be immediately returned to the calling program, or if RPC+ should wait for the completion of the command on the remote system. A value of 'Y' or 'y' indicates that RPC+ should wait for command completion. Any other value indicates that control should be returned immediately.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

CRemoteCopy

The CRemoteCopy function provides the ability to copy files between the local and remote systems.

Sample: status = CRemoteCopy(server, localname, remotename, direction, crlf);

where the parameters are defined as follows:

int status;

char * server;

char * localfile;

char * remotefile;

char direction;

char crlf_conv

and status is returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
1	Socket error
2	Synchronization error
3	Server refused request
4	Cannot create the local file
5	System error
6	Cannot write the new file
7	Cannot open the local file
8	Write operation failed on the remote system

server is an alphanumeric literal or data item that contains the name or IP address of the server the copying should be performed with. In a thin client architecture to copy a file to or from the client, this parameter should be set to spaces.

localname is an alphanumeric literal or data item that contains the name of the file on the local machine, including a path if desired.

remotename is an alphanumeric literal or data item that contains the name of the file on the remote machine, including a path if desired.

direction is a one byte alphanumeric literal or data item that indicates the direction of the copy operation. A value of “I” or “I” requests copying the file IN to the local machine. A value of “O” or “o” requests copying the file OUT to the remote machine.

crlf is a one byte alphanumeric literal or data item that contains indicates whether or not CRLF pairs should be altered based on the operating system conventions of the local and remote machine. This is helpful when copying text files. A value of “Y” or “y” indicates that CRLF characters should be modified. A value of “N” or “n” indicates they should not be altered.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

CVersionCheck

The CVersionCheck function provides the ability to test a local file against a remote file and see if the remote file needs to be updated to match the local file. The function is used as follows:

Sample: status = CVersionCheck(server, localfile, results, fulllocalname, remotefile)

where the parameters are defined as follows:

int status;

char * server;

char * localfile;

char * results;

char * fulllocalname;

char * remotefile;

and status is a value returned by the function to indicate success or failure. Possible values are:

Value	Description
0	Success
3	Server refused request
7	Cannot open the local file

server contains the name or IP address of the server the file should be checked against. In a thin client architecture, to check the file against the client, this parameter should be set to spaces.

localfile contains the name of the file on the local machine that should be checked against the remote machine.

results is set by the function to indicate whether the versions of the files on the local and remote system match. If the files match, this data item is set to a value of Y, otherwise it is set to N.

fulllocalname returns the fully qualified pathname of the file on the local machine. Returning this value from the VBVersionCheck function makes it easy to use the file with theVBRemoteCopy function if the file needs to be copied.

remotefile returns the fully qualified pathname of the file on the remote machine. Returning this value from the CVersionCheck function makes it easy to use the file with the CRemoteCopy function if the file needs to be copied.

Restrictions: This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

GetConfigFile

The GetConfigFile function returns the name of the configuration file used by RPC+. This can be used to directly retrieve configuration information such as server names and port numbers.

Sample: pConfigFile = GetConfigFile();

where the parameters are defined as follows:

char * pConfigFile;

pConfigFile returns a char * pointing to the name of the configuration file.

If INI file mapping is being used, pConfigFile will not contain a physical file name. Because of this, the GetPrivateProfile functions should be used to access the information in the file rather than open() or fopen() type access.

GetRequest

The GetRequest function is used to retrieve the details of a call request. This is commonly done by the client portion of a thin client architecture.

Sample: status = GetRequest(program, argdata, argdescs);

where the parameters are defined as follows:

long status;

char program[80];

char * argdata;

char argdescs[328];

program is the name of the program the caller requested.

argdata is the data passed by the caller.

argdescs is an array of offsets and sizes describing the data passed by the caller. The values are in ascii and occupy the following positions in argdescs:

bytes 0 - 7 = number of client parameters

bytes 8 - 15 = offset of parameter 1

bytes 16 - 23 = size of parameter 1

bytes 24 - 31 = offset of parameter 2 in client data

bytes 32 - 39 = size of parameter 2

up to a maximum of 20 parameter descriptions

ReturnDataToClient

The ReturnDataToClient function completes the processing of a remote caller's request and returns modified data to the caller.

Sample: result = ReturnDataToClient(argdata);

Where the parameters are defined as follows:

int result;

char * argdata;

Argdata is one of the arguments passed to the GetRequest function.

ShutdownRPC

The ShutdownRPC function terminates all connections and frees resources associated with remote programs. It should be called before an application exits.

Sample: ShutdownRPC();

StartServer

The StartServer prepares RPC+ for server operation, allowing remote calls to be received. It must be called once by a server before calls can be received.

Sample: result = StartServer(buf);

Where the parameters are defined as follows:

int result;

char * buf;

buf contains the command line parameters.

Java

The RPCPlus java class provides the functionality necessary for executing the RPC+ methods. A java application must declare an instance of the RPCPlus class, then use the class object to invoke functions and pass parameters.

CallRPCProgram

The CallRPCProgram function is the foundation of RPC+ when using Java. It provides the ability to CALL a program on another system, pass it parameters, and receive the modified parameters after the remote program terminates.

Usage:

```
Client.CallRPCProgram(status, "programname", mydata);
```

Where the parameters are defined as follows:

Client is an object of class RPCPlus.

Status is declared as:

```
int[] status = new int[1];
```

and will be set as a result of the remote call to one of the following values:

Value	Description
0	Success
1	Could not connect to server
2	Could not execute named program on server

programname is an alphanumeric literal specifying the name of the program to invoke on the remote system.

mydata is declared as:

```
byte[] mydata = new byte[xxx];
```

where xxx is between 1 and 60,000 bytes.

Restrictions:

This function may only be executed by the client in a fat client architecture, or the server in a thin client architecture.

ShutdownRPC

This function is used to terminate all connections and free resources associated with remote programs. It should be called before an application exits.

The ShutdownRPC function is executed as follows:

```
Client.ShutdownRPC();
```

Where Client is an object of class RPCPlus.

Chapter 6: Windows Server Administration

Introduction

Before RPC+ can execute remote program calls on a Windows server, the RPC+ Server program must be started. Once started, there are several administrative capabilities offered by the server. This section reviews the features of the Windows RPC+ Server.

Starting and Stopping the Server

To begin handling requests, the RPC+ server daemon must be started so it can listen for connection requests. The RPC+ Server can be started by double clicking its icon in the Program Manager group containing its icon.

When the server is started it immediately initializes and begins accepting remote program calls. This is reflected by the word "Started" displaying in the title bar of the window and by a message displayed in the activity log, if the log is turned on.

The server may be stopped by selecting the Stop command from the Options menu. The word "Started" in the title will be replaced by "Stopped" and a message will be displayed in the activity log, if enabled. The server will not accept any remote program calls while stopped. The server may also be stopped by clicking on the button with the picture of the upright hand.

After stopping, the server may be restarted by selecting "Start" from the Options menu, or by clicking the toolbar button picturing the runner.

The server may be exited by selecting Exit from the File menu.

Configuring the Server

The server can be configured by editing the rpcplus.ini file configuration options.

Using the Activity Log

The activity log keeps a record of RPC+ activity. Every time the server is started or stopped a message is added to the activity log. Every time a client connects to or disconnects from the server another message is added. All server errors are also recorded in the activity log.

The following commands are available for managing the log.

Command	Description
Save As	The activity log can be saved to disk by selecting Save As... from the File menu.
Print Setup	The print configuration can be changed by selecting Print Setup... from the File menu.
Print Preview	The format of the activity log can be previewed by selecting Print Preview... from the File menu.
Print	The activity log can be printed by selecting Print... from the File menu or clicking the toolbar button with the printer icon.
Clear	The contents of the activity log can be cleared by selecting Clear from the Edit menu.

Help

The RPC+ copyright information can be displayed by selecting the About RPC+ Server... option from the Help menu. This manual can be viewed on-line by selecting the Help with RPC+… option from the Help menu.

Chapter 7: Unix Server Administration

Introduction

The installation routine for RPC+ performs all necessary Unix configuration automatically. This section describes that configuration information so that you can change it manually should the need arise.

Before RPC+ can execute remote program calls, a process must be created that will listen for requests. This process, inetd, is a standard part of the Unix operating system. It provides this functionality for features like ftp and finger. It will also provide it for RPC+. We just need to add a line to its configuration file.

What inetd Does

inetd listens for connections on a number of ports. Each port is related to a service such as ftp or RPC+. When inetd senses a connection, it determines which service it is associated with and starts the process related to that service. The inetd.conf file lists the services inetd should listen for and tells it what process to start for each service.

Setting up inetd

inetd is configured by editing the contents of inetd.conf. This file is generally located in the /etc directory. The installation of RPC+ adds a line to this file. It looks like this:

```
rpcplus stream tcp nowait root /bin/sh /bin/sh
   /cobolrpc/rpcstart
```

Here's a description of each component:

The first item, rpcplus, is the name of the service inetd is supposed to listen for. This service must be described in /etc/services. We will get to that later. You should not need to change this.

The next three items, stream tcp nowait, described the type of network communication needed. Do not change these options.

The next item, root, indicates the user that the server process will be initiated for. You may want to have your remote program calls executed under a different user name. Just be certain that the user name used here has adequate permissions to find and execute remote programs.

The next entry, /bin/sh, is the name of the program inetd should initiate for the service. /bin/sh is specified because a shell script is used to start RPC+. This entry is repeated and should not be changed.

The last item, /cobolrpc/rpcstart, is a shell script that starts the Cobol runtime system. The rpcstart script can be edited to set additional environment variables required by the Cobol application.

Defining the server

Remember the service name that was included as the first element in the inetd.conf entry, rpcplus. When RPC+ was installed, the following line was added to the /etc/services file to define that service:

rpcplus 5000/tcp

This defines rpcplus as a service using tcp protocol on port 5000. Do not change the service name or protocol. You can, however, select a different port number. Just be certain that the port number you select is not used by any other service on the server and matches the port number used by the RPC+ clients.

Chapter 8: User Login/Validation Techniques

Thin Client with a Windows Server

Thin client usage, involving RPCPlusThinClient.exe provides the ability to automatically validate user names and passwords when working with a Windows server. Two things are required to implement this:

1. In the rpcplus.ini file on the server, the following entry must be included:

   [ServerConfig]

   UseLogin=TRUE

2. When RPCPlusThinClient is invoked, the user name and password must be supplied on the command line as follows:

   RPCPlusThinClient USER=MyUserName PASSWORD=MyPassword

The RPC+ server or service on the server will validate the user name and password on the server, and create the server process using the profile and access rights of the named user. If the user name or password is invalid, the connection is rejected and an error message is displayed.

3. A domain may also be specified by including it on the command line as follows:

   RPCPlusThinClient USER=MyUserName PASSWORD=MyPassword DOMAIN=MyDomain

This will cause the user name and password to be validated against the specified domain.

Chapter 9: Using a Windows Service

Overview

The RPCPlusServer program that listens for connection requests runs as a Windows application. This means that it must be started from the desktop, and someone must log onto the server to start it. This is not always the most convenient way to operate a server. Windows NT, 2000, and XP have the ability to support Windows Services. A windows service can be started automatically when the machine boots. This section describes how to implement one or more RPC+ services.

Installation

An RPC+ service is implemented in RPCPlusService.exe. To install an RPC+ service, simply execute the following command line:

RPCPlusService –install

This will install an RPC+ service with the default name, RPCPlus. If you want to install multiple services, or give the service a different name, simply place the service name after the –install option as follows:

RPCPlusService –install MyService

This will create a service called MyService.

Configuration

An RPC+ service gets all its configuration information from a configuration file, rpcplus.ini. The configuration information required is identical to that required when using RPCPlusServer.exe. However, RPCPlusServer.exe will always look in its working directory for the rpcplus.ini file. An RPC+ service will always look in the logical Windows directory for the rpcplus.ini file.

Another difference when using an RPC+ service is the working directory. When using RPCPlusServer.exe, if no working directory is specified in rpcplus.ini, the working directory for the server process will be the working directory for the RPCPlusServer process. With the service implementation, the working directory is not specified. Therefore, it is essential that you set the WorkingDir entry of the [ServerConfig] section of rpcplus.ini when using a service.

If you give the service your own name, or install multiple services (each of which must have a unique name), you will need multiple or differently named configuration files. The service version of RPC+ will always look for its configuration information in a file named by the service name, followed by the .ini extension. So, if you create a service called MyService, you would place the configuration information for that service in a file called MyService.ini. This file should of course be placed in the logical Windows directory.

Removal

To remove an RPC+ service, just use the following command:

RPCPlusService –remove

If you have given the service a name other than the default, add the service name on the command line following the –remove option as follows:

RPCPlusService –remove MyService

Starting/Stopping

Installing the service is not enough to get started. Once the service is installed, it must be started. Windows services are started, stopped, and otherwise administered through the Services icon in the Control Panel. This icon is generally located under Administrative Tools.

Note that with some languages/compilers it may be necessary to set the "Allow service to interact with desktop" option

Troubleshooting

Sometimes services can be difficult to debug because they run "behind the scenes". To simplify debugging, the service can be executed as an application. This will cause the process and any debugging windows to be visible on the desktop. To run the service in this manner use the following command:

```
RPCPlusService –debug
```

A console window will be displayed, and any windows associated with the application will also be displayed and can be interacted with.