REMOTE PROCESS CAPABILITY

Activation Information

To have RPC enabled, please follow the standard instructions outlined in the Installation Manual for your specific platform. When you are asked to enter the Serial Number and Activation Key, please use the following information.

Objectives:

The Remote Process Capability is intended to allow for true distributed processing of ProvideX applications across a network of TCP/IP connected machines. It allows applications and their files to be distributed in such a way that the processing logic can be execute on the same machine as the data files thus minimizing the time required to run the application -- increasing throughput and overall system performance.

Background:

Current network implementations limit their ability to perform in high volume situations because of the requirement to transfer the data from the database to the individual client machine. Normal implementations consist of a data or file server in which the application will pass requests to the server for each record they require to process a function. This means that a simple function such as inventory could involve multiple transfers -- one for the product master to read product status, one to update inventory, one to update a transaction log file of some sort, and possibly one to update GL and other related information. Ideally the inventory allocation transaction would consist of a single network request -- allocate inventory.

It is this scenario that the Remote Process Capability attempts to address through the use of Remote Procedure Calls while at the same time providing a 'classic' data or file server facility..

What is a Remote Procedure Call:

Fundamentally a Remote Procedure Call is a CALL to a program that will run on a different processor on the network. Syntactically it is the same as a standard CALL statement except that the name of the program and the parameters that are specified on the call will be placed into a TCP/IP data packet, shipped to a Remote Program Server, which in turn will load and run the specified program passing it the parameters. When the program EXITs, a response packet will be created containing the parameters as altered by the remote program and sent back to the CALLing program. The net result is the seamless integration of remote processing into ProvideX applications.

Using the original example of inventory allocation, using a RPC will result in a single network exchange in lieu of three or more. An inventory allocation ProvideX subprogram could be written that would be CALLed remotely to perform all the logic pertaining to transaction.

Server Names:

In order to simplify the process of configuration, ProvideX allows for the definition of Logical Server names. These logical server names can be from 1 to 12 characters and are used to identify which server will handle which CALL request.

Issuing a RPC:

In order to initiate a Remote Procedure Call the calling program simply issues a CALL statement with the name of the program being prefixed by "[RPC:*logical_server_name*]" (case insensitive). The presence of an [RPC] clause will be all that is required to initiate the remote CALL.

Since ProvideX already provides for a Prefix File, existing CALLs can be dynamically altered to RPCs. For example a PREFIX FILE entry for the file "INVALOC" could be created with its contents being "[RPC:Inventory]INVALLOC" thus a simple CALL "INVALLOC" would result in a RPC request.

Establishing a Remote Program Server:

Each remote program server will need to be identified to the CALLing application. This will be done through the directive:

PROCESS SERVER "name" ON "tcp_address;server"

Where:

"name" is the logical name associated with the program server
"tcp_address;server" is the TCP address and services identifier for the server

Example:

PROCESS SERVER "Inventory" ON "[tcp]192.1.1.100;15000"

If desired the "tcp_address;server" can be set to "LOCAL" which will result in any RPC call to this server simply becoming a local CALL directive. Alternatively the tcp_address can specify a pipe on Unix or a DLL on windows.

To terminate the Remote server connection:

PROCESS SERVER "name" CLOSE

Running subprograms on a Remote Program Server:

Subprograms designed to run on a Remote Server should make no assumptions about the current environment. While the server can be initiated in a specific disk directory, and as such could have it own START_UP procedure, the subprogram should attempt to make sure that the files it requires can be directly addressed.

In addition, no attempt should be made to display anything on the user terminal. An RPC subprogram is running as a background task on the server and any terminal output will be displayed on the server, not the CALLing terminal.

Remote File Access:

In addition to the ability to CALL programs on a remote server, the OPEN command has been extended to allow files to be opened remotely as well. To open a remote simply prefix the file name with the '[rpc:xxxxx]' designation to indicate the server that is to handle the file requests.

Background Processing:

You can also START background programs using the remote server facilities by simply adding the '[rpc:xxxxx]' designation to the start of the command-line/Program name to be started.

Running the Remote Server:

To start the remote server, you simply need to start up a background process that CALLs the program "*rpsrvr" with the port/service number. For example:

```
CALL "*rpsrvr", 15000
```

This will establish an Remote Process Server as port/service 15000 on you system. It is important that you carefully administer the port addresses that you assign to the servers to avoid duplication.

Using a Piped Remote server (UNIX)(Target - 1st Q / 98):

Instead of utilizing a TCP port, you can use a PIPE to establish a connection to a Remote Server. In this case instead of a TCP address case the PROCESS SERVER directive should contain the path name of the pipe server.

Example:

```
PROCESS SERVER "Inventory" ON "|/u/sys001/inventory"
```

Using a DLL Remote server (WINDOWS))(Target - 1st Q / 98):

Instead of utilizing a TCP port, you can use a DLL to establish a connection to a Remote Server. In this case instead of a TCP address case the PROCESS SERVER directive should contain the path name of the DLL and the Entry point within the DLL.

Example:

```
PROCESS SERVER "Inventory" ON "[dll]c:\app\server.dll;Entry"
```

Remote File Access Data Packet format

All data packets for File access contain the following header:

Major_code Single character major function (Always "F" for File I/O) Minor_code Single character minor function Size Two byte binary packet size Crc Two byte CRC-16 Data... Variable length packet data

The format of the data elements in the packets are:

CHR a one byte character INT a two byte binary value LONG a 4 byte binary value STR a string of data prefixed by a two byte binary length If string is omitted then the size = -1

If a return value is specified it should contain a ACK 06 in the first character position followed by a two byte return length then the data. If an Error occurs the first character should be NAK 15 followed by a two byte binary error code.

The following commands are supported:

OPEN			Minor code "O"
	Option	CHAR	Open option (*=Load, I=Input, P=Purge, L=Lock, Blank=None)
	Isz	INT	ISZ value if specified
	Path	STR	Pathname from the OPEN directive
	Opt=	STR	Value specified in OPT= parameter
	(Return)	INT	Logical file number to be used in all subsequent requests
CLOSE			Minor code "C"
	Fileno	INT	Logical File number to close
WRITE			Minor Code "W"
	Fileno	INT	Logical File Number
	Sub1	CHAR	Record access option (K=Key, I=Ind, R=Rno)
	Sub2	CHAR	Error Option (D=DOM specified)
	Key	STR	Key if specified
	Ind	LONG	Record Ind if specified or Rno value
	Data	STR	Data to write
READ			Minor Code "R"
	Fileno	INT	Logical File Number
	Sub1	CHAR	Record access option (K=Key, I=Ind, R=Rno)
	Sub2	CHAR	Blank (Not Used)
	Key	STR	Key if specified
	Ind	LONG	Record Ind if specified or Rno value
	Kno	INT	Key number
	(Return)	STR	Record Contents
REMOVE			Minor Code "-"
	Fileno	INT	Logical File Number
	Sub1	CHAR	Record access option (K=Key, I=Ind, R=Rno)
	Sub2	CHAR	Blank (Not Used)
	Key	STR	Key if specified
	Ind	LONG	Record Ind if specified or Rno value
KEY() IND() RNO()			Minor Code 'K'
	Fileno	INT	Logical File Number
	Sub1	CHAR	Record access option (K=Key, I=Ind, R=Rno)
	Sub2	CHAR	' '=KEY, 'P'=KEP, 'C'=KEC, 'N'=KEN, 'F'=KEF, 'L'=KEL,
			'R'=RNO, 'I'=IND
	Key	STR	Key if specified
	Ind	LONG	Record Ind if specified or Rno value
	Kno	INT	Key number
	(Return)	STR LONG	Returns KEY value as STR or IND/RNO as a LONG