Rocket Model 204

Parallel Query Option/204

User's Guide

Version 7 Release 5.0

September 2014
204–75–PQO-01

Notices

Edition

Publication date: September 2014
Book number: 204–75–PQO-01
Product version: User's Guide

Copyright

© Rocket Software, Inc. or its affiliates 1989—2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc. or its affiliates, are furnished under license, and may be used and copied only in accordance with the terms of such license.

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Corporate Information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America	+1.800.755.4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

About this Guide

This guide describes Parallel Query Option/204, a Model 204 distributed processing facility. Previously, Parallel Query Option/204 was called Distributed/204.

Audience

This guide is for all PQO administrators, managers, and programmers. Familiarity with Model 204 installation and operation, including User Language (SOUL) and application subsystems, is assumed.

A note about User Language and SOUL

Model 204 version 7.5 provides a significantly enhanced, object-oriented, version of User Language called SOUL. All existing User Language programs will continue to work under SOUL, so User Language can be considered to be a subset of SOUL, though the name "User Language" is now deprecated. In this manual, the name "User Language" has been replaced with "SOUL."

Rocket Model 204 documentation

To access the Rocket Model 204 documentation, see the Rocket Documentation Library (http://docs.rocketsoftware.com/), or go directly to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/).

Additional documentation

Depending on your level of experience and familiarity with WebSphere MQ and Model 204, you might need additional documentation. For WebSphere MQ documentation, contact your IBM representative.

Rocket recommends the following IBM manuals:

- *WebSphere MQ Application Programming Guide* (SC34-6064)
- *WebSphere MQ Application Programming Reference* (SC34-6062)
- *WebSphere MQ for z/OS Messages and Codes* V5.3.1 (SC34-6056)

To obtain additional information on WebSphere MQ or to download the WebSphere MQ manuals, access the IBM Web site. Their Web address is:

`http://www-4.ibm.com/software/ts/mqseries/`

Documentation conventions

This guide uses the following standard notation conventions in statement syntax and examples:

Convention	Description
`TABLE`	Uppercase represents a keyword that you must enter exactly as shown.
`TABLE` *`tablename`*	In text, italics are used for variables and for emphasis. In examples, italics denote a variable value that you must supply. In this example, you must supply a value for *tablename*.
`READ [SCREEN]`	Square brackets ([ ]) enclose an optional argument or portion of an argument. In this case, specify READ or READ SCREEN.
`UNIQUE | PRIMARY KEY`	A vertical bar (|) separates alternative options. In this example, specify either UNIQUE or PRIMARY KEY.
`TRUST | NOTRUST`	Underlining indicates the default. In this example, NOTRUST is the default.
`IS {NOT | LIKE}`	Braces ({ }) indicate that one of the enclosed alternatives is required. In this example, you must specify either IS NOT or IS LIKE.
`item ...`	An ellipsis (. . .) indicates that you can repeat the preceding item.
`item ,...`	An ellipsis preceded by a comma indicates that a comma is required to separate repeated items.
`All other symbols`	In syntax, all other symbols (such as parentheses) are literal syntactic elements and must appear as shown.
`nested-key` `::=` *`column_name`*	A double colon followed by an equal sign indicates an equivalence. In this case, *nested-key* is equivalent to *column_name*.
`Enter your account:` `sales11`	In examples that include both system-supplied and user-entered text, or system prompts and user commands, boldface indicates what you enter. In this example, the system prompts for an account and the user enters **sales11**.
File > Save As	A right angle bracket (>) identifies the sequence of actions that you perform to select a command from a pull-down menu. In this example, select the Save As command from the File menu.
E`DIT`	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

1

Introduction to Parallel Query Option/204

Overview

Parallel Query Option/204 (PQO) is a Model 204 distributed processing facility that allows the sharing of data between two or more copies of Model 204.

Parallel Query Option/204 provides SOUL applications access to remote files. A single SOUL transaction can read and/or update multiple remote files.

Parallel Query Option/204 features

Parallel Query Option/204 (PQO) allows a SOUL request to access data that is owned by other copies of Model 204. You can use the same SOUL statement syntax for remote files as for local files. You can horizontally partition your data. You can access data physically stored on multiple nodes in the network as if the data were contained in a single database on a single node.

In addition, PQO processing provides the following capabilities:

- Update access to remote data, limited to one node per transaction
- Read access to remote data in transactions that can read local data
- Access to remote data from ad hoc requests
- Access to remote data from application subsystems for precompiled and non-precompiled requests
- Error handling and recovery
- Synonyms for data files
- Symbolic references to network location

- File groups comprised of members from multiple copies of Model 204
- Scattered groups (file groups that include remote files)
- Application subsystem support for client and server applications
- Trust definitions for application subsystems, to control remote user access by using definitions on the service node

PQO *does not allow:*

- Local access to or execution of SOUL procedures stored in a remote file
- Remote execution of updating data definition language (DDL), Model 204 commands that define or modify files or fields

Communicating between versions of Parallel Query Option

PQO V6R3.0 and later cannot communicate with PQO V6R1.0 or earlier. If you attempt this, Model 204 issues the following message:

```
M204.2327: PQO VERSION INCOMPATIBILITY WITH REMOTE NODE:
location
```

Location transparency

Location transparency refers to the ability to access objects without knowledge of their location. PQO offers two location transparency options for local or remote files. You can define your own names for files and for the Onlines where the files reside. Your names are called *symbolic names* in this guide. Using them allows you to simplify your naming conventions and saves you from having to update all your file references if PQO partners change names or shift data.

- See “Specifying a remote file” on page 55 about using symbolic names for file locations.
- See “Using a file synonym” on page 57 for information about using symbolic file names that implicitly refer to a file name and location.

System requirements

Each system that participates in a PQO network must provide the necessary support structure. The following components are required for PQO network support:

- z/OS, z/VM, or z/VSE operating system
- TCP/IP or SNA Communications Server (formerly VTAM) for z/OS, z/VSE, and SNA Communications Server and Inter-User Communication Vehicle (IUCV) for z/VM
- Model 204 base product, Version 6 Release 1.0 or later

- The LU 6.2 communications feature of Model 204's Horizon facility, Version 6 Release 1.0 or later

PQO is incorporated directly into the core of Model 204 by default. Using PQO, application subsystem definitions and requests can refer to remote files or scattered groups. You have the opportunity to try PQO without additional expense. The number of threads is limited to two. If you want additional threads, please notify Technical Support.

Network communications

To support a PQO network, you must set up an underlying communications layer. This layer is governed by TCP/IP or the Systems Network Architecture (SNA) logical unit (LU) session type 6.2 conversation rules. PQO uses the Model 204 LU 6.2 interface provided with the Horizon facility. Chapter 2 describes how to define this layer.

In the PQO network, a node is typically synonymous with a Model 204 Online. Communication is between two or more nodes: one client and at least one server. The client issues the request to open a remote file or group. The server responds to the client's request and provides access to a file residing on the server. If files on other nodes are involved in the request, those other nodes become servers and provide access to the requested files.

Note: In the PQO network, a node can function as a client only, as a server only, or as both a client and server.

An LU 6.2 conversation between client and server is started when a remote file is opened. Figure 1-1 shows the basic components of PQO network processing. System A is defined as a client and System B is defined as a server in the network. A procedure that is stored on System A is executed. The procedure references a file that resides on System B.

When the OPEN FILE command is executed, the links connecting the two nodes are activated and an LU 6.2 conversation is initiated.

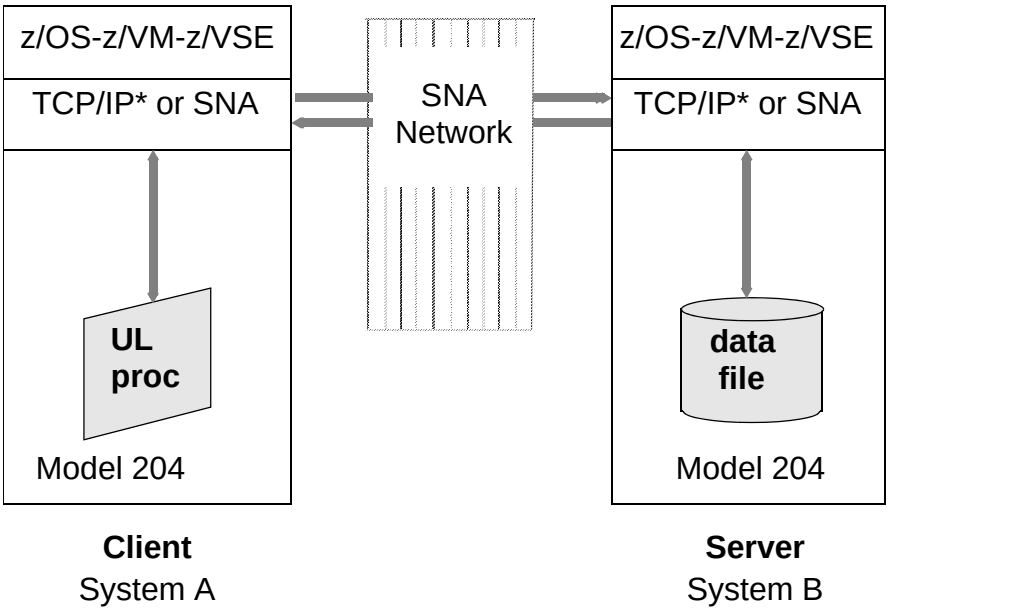


Figure 1-1. PQO processing

Basic setup and operation

Perform these basic steps to set up and use a PQO network.

1. Set up the network:
   - Prepare the PQO network definitions and CCAIN parameters for each copy of Model 204 in the PQO network.
   - Prepare the SNA Communications Server definition or allocate a TCP/IP port number for each copy of Model 204 in the PQO network.

   See Chapter 2 for details.

2. For each network location:
   - Activate the SNA Communications Server definition, unless using TCP/IP, and bring up the Model 204 Online to support PQO communications.
   - Log in to Model 204.
   - Open a PQO communications line. Each node must open its PQO communications link (OPEN LINK).

3. A client system initiates communications in the network. At the application level, from within a SOUL procedure:

– Issue a command to open a remote file or files (OPEN FILE or OPEN GROUP). This starts an LU 6.2 conversation between the client and each server system that contains one of the files. It also starts a service thread (IODEV=51) on each server that contains one of the files.

  A second OPEN command for another file on a remote system involved in the first OPEN uses that existing conversation and service thread.

– Begin a request that refers to the remote file(s).

– The request is compiled and execution begins.

– After a FIND statement is executed, a FOR EACH RECORD loop is entered. A buffer full of found records is shipped from the server to the client. The records are stored in the client's CCATEMP file.

  Each iteration of the FOR EACH RECORD loop gets the next record from CCATEMP until all buffered records are exhausted. A call is made to the service thread for another buffer of records.

– If the FOR EACH RECORD loop contains an update statement, the records buffered at the client are updated, and a call is made to the server to apply the same update to the records there.

– The request ends; other requests may follow.

– The client closes the remote file or group (CLOSE command). When the last remote file is closed on a server, its service thread for that client is logged out automatically, and that LU 6.2 conversation ends.

2

Defining a Parallel Query Option/204 Network

Overview

A Parallel Query Option/204 network must be defined before any PQO application can run successfully.

The Model 204 system manager defines the PQO network configuration and protocols to Model 204. The network systems programmer sets up the SNA Communications Server definitions, if you are using an SNA Communications Server for the network.

Finally, the Model 204 system manager sets Model 204 Online environment parameters that are required for PQO.

Once the network is properly defined, its operation is transparent to the SOUL application programmers.

Network configuration

The PQO network connects two or more copies of Model 204. The Model 204 copies can be on the same physical machine or not, and they can be in the same operating system or not. The copies communicate through TCP/IP or SNA Communications Server over the communications network. Both network types use the LU 6.2 interface services of the Model 204 Horizon facility.

Each copy of Model 204 opens links that are either SNA logical units or TCP/IP ports, and provides network access and services to its PQO

partners. Each node can accommodate two types of functions, or *processes*, in Horizon terminology:

This process...	Performs only this action...
Client	Queries partner databases
Server	Replies to database queries

These are companion processes, paired in the network. A client on one node initiates a conversation requesting access to a remote file. The corresponding server on another node responds to the request.

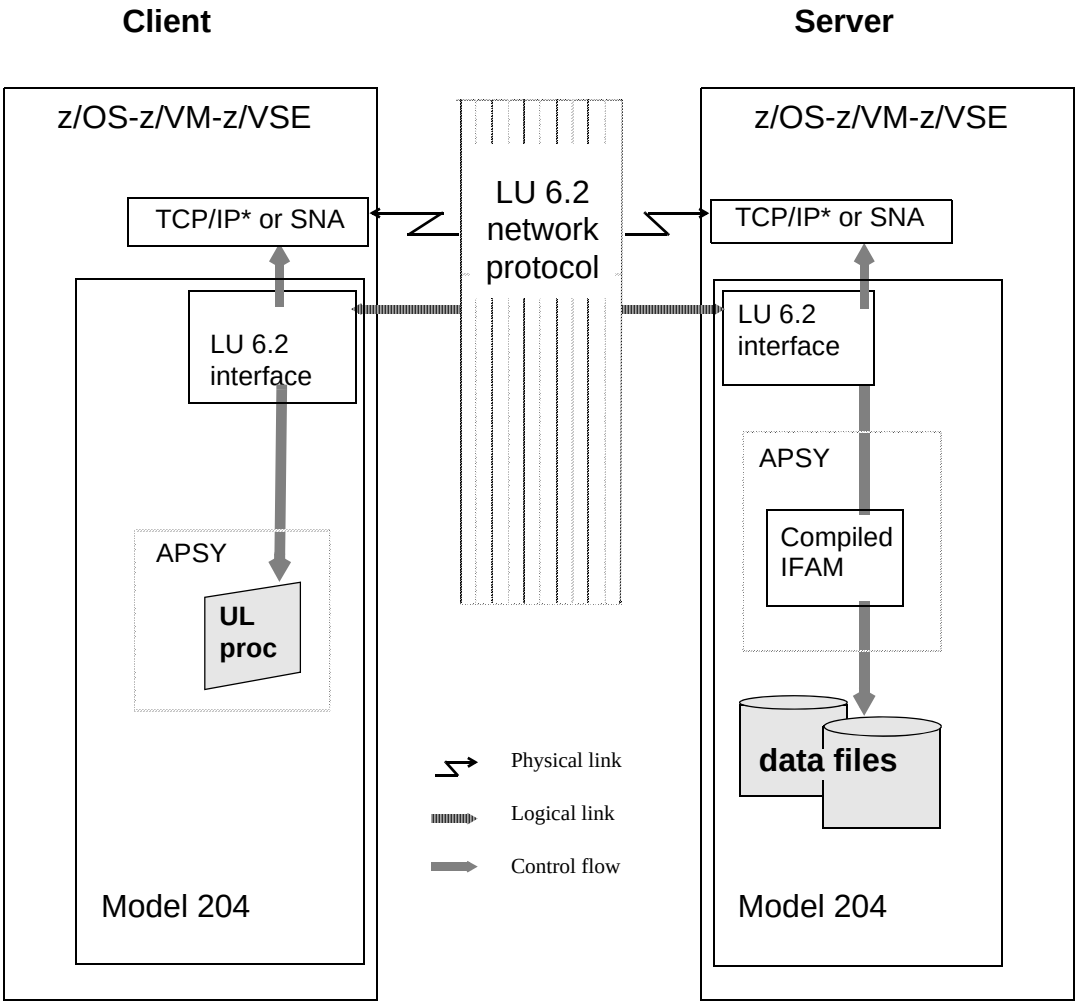
In this guide, a node is often referred to in terms of the type of network process that is active—a *client node* is the node on which the client program being referred to is defined.

Figure 2-1 shows the physical configuration of a PQO network with one client and one server. The direction of the control flow arrow shows the path of a request for remote data in the PQO network where communication is initiated by the client. The path is reversed when the data is returned.

Network characteristics:

- Two or more nodes: one is client; one or more are servers
- Two-way path, when links are activated
- Communication initiated by client
- Optional use of APSY subsystems

Figure 2-1. Basic network configuration



TCP/IP network service is currently available for only z/OS and z/VSE operating systems

Figure 2-1 illustrates a client/server relationship between two Onlines. An Online may be configured as a client of any server Online and at the same time as a server for any other client Online.

Interrelated network elements

Certain parameters in the Model 204 and the network definitions must be set in relation to one another, as described in the sections that follow. Because the definitions are mutually dependent, you might have syntactically valid definitions that are not compatible. Such an error is not detected until you first try to open a remote file.

Table 2-1 on page 10 summarizes these interrelated elements for TCP/IP service.

Table 2-1. Interrelated components of a PQO network for TCP/IP service

Model 204			
DEFINE LINK	**DEFINE PROCESS-GROUP**	**DEFINE PROCESS**	**Model 204 Online**
Each link requires a SERVPORT specifying the port number for this connection. LOCALID must specify the dotted decimal address of the TCP/IP on this operating system	REMOTEID applid must match LOCALID name on remote LINK definition.		
CONNECTIONS sets maximum number of concurrent sessions, plus two, for a minimum of 50.			
	For a server, use MASK to limit the range of addresses that can connect to this port. For a client, use PORT to specify the port of the server, which must match SERVPORT of the server's DEFINE LINK.		
Link definition name identifies connection	LINK parameter name matches DEFINE LINK name		
	Processgroup definition name identifies group	DESTINATION or FROM name matches defined processgroup name	
	LOGIN=TRUST specifies password not required		SYSOPT includes X'10' to require login (server system)
	INLIMIT sets maximum number of concurrent inbound conversations (server)		NOTERM sets maximum number of threads for inbound conversations (server)

Table 2-2 on page 11 summarizes these interrelated elements for SNA Communications Server service. Only those items that have a corresponding element in a related definition are shown in Table 2-2.

Table 2-2. Interrelated components of a PQO network for SNA Communications Server

SNA		**Model 204**			
Mode table	**APPL statement**	**DEFINE LINK**	**DEFINE PROCESS-GROUP**	**DEFINE PROCESS**	**Model 204 Online**
	One APPL for each link Label identifies LU application node	Each link requires an APPL LOCALID name must match APPL label	REMOTEID applid must match LOCALID name on remote LU		
	PARSESS=YES enables support for concurrent sessions	SESSIONS sets maximum number of concurrent sessions			
	PRTCT specifies password	PSWD enables password protection			
		Link definition name identifies connection	LINK parameter name matches DEFINE LINK name		
			Processgroup definition name identifies group	DESTINATION or FROM name matches defined processgroup name	
			LOGIN=TRUST specifies password not required		SYSOPT includes X‘10’ to require login (server system)
			INLIMIT sets maximum number of concurrent inbound conversations (server)		NOTERM sets maximum number of threads for inbound conversations (server)
LU 6.2 mode table for Model 204	MODETAB specifies LU 6.2 table DLOGMOD specifies entry in table (optional)		MODENAME specifies entry in mode table (optional)		

PQO DEFINE commands

For z/OS and z/VSE systems, TCP/IP may be used instead of SNA Communication Server. The discussion in this chapter will point out where and how you must define either type of network. You can also consult the Rocket Model 204 documentaiton wiki (http://m204wiki.rocketsoftware.com/index.php/DEFINE_LINK_command:_Horizon_for_TCP/IP).

For each copy of Model 204 in the PQO network, you must define three types of entities for each client or server application: a link, a processgroup, and a process. These entities and their interdependencies form the network's logical layer.

In addition, some of the network entity definition parameters must be set relative to the SNA Communications Server network definition, which is described on page 23.

The PQO network definitions are closely parallel to those of Horizon, which are described in the *Model 204 Horizon: Intersystem Processing Guide*. This section presents the PQO DEFINE command syntax and describes the parameter specifications that are modified or otherwise different from the standard Horizon definitions. Examples of sample PQO network definitions are provided in the section "Two-node network example" on page 26 and in Appendix A.

Using the DEFINE LINK command

The DEFINE LINK command requirements have differing resource parameters for TCP/IP or SNA, so the syntax for each varies a bit. In general TCP/IP and SNA supply the following link definition:

- Specifies a name that identifies the Model 204 node to the SNA network
- Defines the physical communications transmission method (TRANSPORT) and the logical conversation rules (PROTOCOL)
- Specifies resource parameters, which differ for TCP/IP and SNA. For example, for TCP/IP, CONNECTIONS replaces SESSIONS for SNA. Use the DEFINE LINK command syntax suitable for your site.

Although a single link can be shared by both PQO and Horizon conversations, for performance reasons, Technical Support recommends against sharing links.

A connection request for a link can succeed only if an OPEN LINK command has opened the link. For more information about OPEN LINK, see Chapter 3.

DEFINE LINK command for TCP/IP

The TCP/IP-specific PQO syntax of the DEFINE LINK command is:

Syntax

```
DEFINE  LINK name [LIKE previousname] WITH          -
        SCOPE=SYSTEM                                -
        PROTOCOL=IP                                 -
        TRANSPORT={TCPSE | TCPVSE}                  -
        CONNECTIONS=nnnn                            -
        INBUFSIZE=nn                                -
        LOCALID={address | ANY}                     -
        SERVPORT=nnnnn
```

Parameters PQO-specific comments and recommendations for TCP/IP service follow:

- *PROTOCOL=IP* is required and specifies the set of rules (SNA LU 6.2 on a TCP/IP link) used to control communications.
- *TRANSPORT* must be set to TCPSE (for z/OS) or TCPVSE (for z/VSE).
- *CONNECTIONS* is required and specifies the maximum number of concurrent connections that can be activated for the link, plus two. A connection is activated when a conversation is initiated.

  For PQO with TCP/IP service, a starting value of 50 is recommended with adjustment upward as necessary to accommodate actual network traffic. See “Usage notes on IODEV=51” on page 23 for information about the relationship between the number of connections, the number of server threads, and the number of remote users who can access remote files on a particular node.
- *INBUFSIZE* is required and specifies the size of the buffer allocated for the TCP/IP data transfer for each conversation. Technical Support recommends that you set this to at least 4096, for efficiency.
- *LOCALID* is required and specifies the TCP/IP address that identifies this system.
- SERVPORT must be set to a port number greater than 4096 and unique for this link. It must match the PORT parameter of the DEFINE PROCESSGROUP command of the client system.

DEFINE LINK command for an SNA network

The PQO syntax of the DEFINE LINK command for SNA network is

Syntax

```
DEFINE  LINK name [LIKE previousname] WITH          -
        SCOPE=SYSTEM                                -
        TRANSPORT=VTAM                              -
        PROTOCOL=LU62                               -
        SESSIONS=nn                                 -
        INBUFSIZE=nn                                -
        LOCALID=applid                              -
        [PSWD | NOPSWD]
```

Required for z/VM only:

```
GCSID=vmid
```

Parameters PQO-specific comments and recommendations for SNA service follow:

- *PROTOCOL=LU62* is required and specifies the set of rules (SNA LU 6.2) used to control communications.
- *SESSIONS* is required and specifies the maximum number of concurrent sessions, or connections between logical units, that can be activated for the link. A session is activated when a conversation is initiated.

  For PQO, a starting value of 3 is recommended with adjustment as necessary to accommodate actual network traffic. See “Usage notes on IODEV=51” on page 23 for information about the relationship between the number of sessions, the number of server threads, and the number of remote users who can access remote files on a particular node.
- *INBUFSIZE* is required and specifies the size of the buffer allocated for the receive-any operation for the initial SNA Communications Server Request Unit (RU) of each inbound conversation.

  Set INBUFSIZE to the same value as the SNA Communications Server logmode parameter RUSIZES (whose recommended setting for PQO is 2K). For more information about RUSIZES, see the *Model 204 Horizon Intersystem Processing Guide*.
- *LOCALID* is required and specifies the LU name that identifies this Model 204 application node to the SNA network. The LOCALID must match the name in the SNA Communications Server APPL network definition. See “Coding the APPL statements for the SNA Communications Server network” on page 24 for a description of the APPL name definition.

Using the DEFINE PROCESSGROUP command

The DEFINE PROCESSGROUP command:

- Groups processes according to certain attributes, such as the maximum number of concurrent conversations, to facilitate resource allocation by applications
- Connects processes associated with each named group to a specific link
- Specifies a remote node with which the associated processes can communicate
- Provides a pool of sessions to be used by multiple processes

A Model 204 site can use different processgroup definitions to segregate groups of processes, for example, to isolate groups that participate in PQO network processing from Horizon network groups.

A single processgroup definition can be used by both PQO and Horizon conversations.

DEFINE PROCESSGROUP command for TCP/IP service

The PQO syntax of the DEFINE PROCESSGROUP command for TCP/IP service is:

Syntax

```
DEFINE PROCESSGROUP name [LIKE previousname] WITH -
       SCOPE=SYSTEM                               -
       LINK=linkname                              -
       REMOTEID=address                           -
       INLIMIT={nnnn | 0}
       PORT=nnnnn
       GUESTUSE={ACCEPT | REJECT}
       LOGIN={TRUST | NOTRUST}
       MASK={dotted-decimal | 255.255.255.255}
```

Parameters PQO-specific comments and recommendations for TCP/IP service follow:

- *name* is used on DEFINE PROCESS commands to refer to this processgroup. The length of the name must be eight characters or less. ALL is a reserved word and cannot be used.
- *LINK* is required and associates the group with a locally defined link that implies the transport type and conversation protocol. A *linkname* must match the name used in a local DEFINE LINK command.
- *REMOTEID* is required and identifies the remote TCP/IP system with which the group communicates. It must match the LOCALID parameter specified in the DEFINE LINK command at the remote node.
- *INLIMIT=nnnn* and *NOINLIMIT* are mutually exclusive options that control the allocation of conversations with the server.

  INLIMIT sets the maximum number of client requests for conversations with a server. To ensure that there are enough service threads to handle these requests, specify an appropriate value for the IODEV line parameter NOTERM (see “Specifying IODEVs for PQO service threads” on page 22).

  NOINLIMIT or a nonzero INLIMIT value is required for the server in PQO.

  Specifying NOINLIMIT is equivalent to specifying 32767; that is, as many as 32,767 concurrent connections are allowed.
- *OUTLIMIT=nnn* and *NOOUTLIMIT* are mutually exclusive options that control the allocation of conversations with the client. NOOUTLIMIT or a nonzero OUTLIMIT value is required for the client in PQO.

- *PORT* is required for the client system and specifies the remote server port (SERVPORT) with which the client will establish a conversation. PORT is not required in the DEFINE PROCESSGROUP command of the service Online, unless that Online is also a client to another server.
- *LOGIN* indicates whether the local server process requires a password from the remote client process at OPEN FILE time. Since there is no facility for handling a password in PQO, TRUST is the required value for LOGIN.

  The CCAD2S process logs in the incoming client user without a password and assigns privileges from either the CCASTAT password table or an external security package. The client user ID must be specified in CCASTAT or in a security package on the server.

  Both client and server must specify LOGIN=TRUST.
- *MASK* is used to limit the range of remote network addresses that may connect to this system. MASK is used with REMOTEID to identify a range of remote addresses that may connect to this server system.

  For example, a REMOTEID of 12.120.0.0 and a MASK of 255.255.0.0 will prevent connections from any network address that does not begin with 12.120. However, the last two bytes of the remote dotted decimal address in this example can be any value, because the MASK has not selected this part of the address for comparison.

DEFINE PROCESSGROUP command for SNA service

The PQO syntax of the DEFINE PROCESSGROUP command is:

Syntax

```
DEFINE PROCESSGROUP name [LIKE previousname] WITH -
       SCOPE=SYSTEM                               -
       LINK=linkname                              -
       REMOTEID=applid                            -
       LOGIN=TRUST
```

Optional parameters:

```
INLIMIT=nnnn | NOINLIMIT
OUTLIMIT=nnnn | NOOUTLIMIT
RETAIN=nnnn | RETAINALL
MODENAME=modeentry
GUESTUSE={ACCEPT | REJECT}
```

Parameters PQO-specific comments and recommendations for SNA service follow:

- *LINK* is required and associates the group with a locally defined link that implies the transport type and conversation protocol. *linkname* must match the name used in a local DEFINE LINK command.

- *REMOTEID* is required and identifies the remote node with which the group communicates. The *applid* value, the SNA Communications Server APPL definition name of the remote node, must match the LOCALID parameter specified in the DEFINE LINK command at the remote node.
- *LOGIN* indicates whether the local server process requires a password from the remote client process at OPEN FILE time. Since there is no facility for handling a password in PQO, TRUST is the required value for LOGIN.

  The CCAD2S process logs in the incoming client user without a password and assigns privileges from either the CCASTAT password table or an external security package. The client user ID must be specified in CCASTAT or in a security package on the server.

  Both client and server must specify LOGIN=TRUST.
- *INLIMIT=nnnn* and *NOINLIMIT* are mutually exclusive options that control the allocation of conversations with the server.

  INLIMIT sets the maximum number of client requests for conversations with a server. To ensure that there are enough service threads to handle these requests, specify an appropriate value for the IODEV line parameter NOTERM (see page 22).

  NOINLIMIT or a nonzero INLIMIT value is required for the server in PQO.
- *OUTLIMIT=nnnn* and *NOOUTLIMIT* are mutually exclusive options that control the allocation of conversations with the client.

  NOOUTLIMIT or a nonzero OUTLIMIT value is required for the client in PQO.
- *MODENAME* is optional and specifies a particular entry in the SNA Communications Server mode table to be used when a SNA Communications Server session is established for an outbound conversation. For information about the SNA Communications Server mode table, see the MODETAB parameter description on page 25.
- GUESTUSER depends on whether and/or which you have a third-party external security package installed for Model 204.
  - GUESTUSER=ACCEPT is required for PQO when RACF or TOP SECRET is installed.
  - GUESTUSER=REJECT, the default, is required for PQO when CA-ACF2 is installed.

Using the DEFINE PROCESS command

There are two distinct types of PQO processes: client and server. The client process, CCAD2C, is the network entity that initiates a conversation. The server process, CCAD2S, accepts a remote communications request.

The DEFINE PROCESS command syntax is the same for TCP/IP and SNA.

The DEFINE PROCESS command associates the process name with one or multiple processgroups, each of which associates it with a link.

DEFINE PROCESS command for either TCP/IP or SNA service

The PQO syntax of the DEFINE PROCESS command is:

Syntax

```
DEFINE PROCESS {CCAD2C | CCAD2S}
  [LIKE previousname] WITH SCOPE=SYSTEM
  [TIMEOUT=nnnn]
```

Required for the CCAD2C client process:

```
DESTINATION=(pg1,sym1 [,pg2,sym2] •••)
```

Required for the CCAD2S server process:

```
FROM= {processgroup | (pg1 [,pg2] •••)}
```

Client process DESTINATION parameter

The DESTINATION parameter is required for the CCAD2C client process. DESTINATION associates the process with a locally defined processgroup or set of processgroups.

DESTINATION is specified as a list of processgroups with associated symbolic names.

```
DESTINATION=(pg1,sym1 [,pg2,sym2] •••)
```

where:

- *pg1* and *pg2* specify the names of the local processgroups with which the client process is associated.
- *sym1* and *sym2* are symbolic names, each from 1-8 characters in length. Each symbolic name is associated with the corresponding processgroup name in the pair.

  A symbolic name defined here is used with AT in a remote file specification to refer to a processgroup that is associated with a particular remote location. “Specifying a remote file” on page 55 describes how symbolic location names are used to reference a remote file.

Each processgroup must be paired with a symbolic name, and vice versa. If not, the DEFINE PROCESS command is rejected or the opening of remote files fail. See Appendix A for examples of DESTINATION parameter settings.

Server process FROM parameter

The FROM parameter is required for the CCAD2S server process and associates it with a locally defined processgroup or set of processgroups. The

FROM parameter can be specified as either a single processgroup or as a list of processgroups:

```
FROM={processgroup | (pg1 [,pg2]•••)}
```

where:

- *processgroup* specifies the name of a local processgroup with which the server process is associated.
- *pg1* and *pg2* specify the names of the local processgroups with which the server process is associated.

See Appendix A for examples of FROM parameter settings.

Client and server TIMEOUT parameter

TIMEOUT sets the time in seconds a process waits for a network transmission to arrive from a partner. If the TIMEOUT limit is exceeded, the connection is abnormally terminated. A PQO server process is likely often to be in protracted waiting states (wait type 27) while clients process the data the server has just sent.

Therefore, if you set the TIMEOUT value in a CCAD2S definition, make sure it is sufficiently high to avoid frequent unintended timeouts. In agreement with the client system manager, you can accept the default, TIMEOUT=0 (no server timeout) for the CCAD2S definition and rely on the client to provide terminal timeout checking with the Model 204 CCAIN TIMEOUT.

Runtime and user environment definition

In addition to specifying the network definitions, it is necessary to set up the Model 204 Online environment to support the PQO network. This section describes how to set the Model 204 environment parameters in the CCAIN input stream for PQO.

The parameters listed in Table 2-3 and described in this section are required in the client Online, the server Online, or both to support the PQO network.

Table 2-3. CCAIN parameters required for PQO

Parameter	Required on client?	Required on server?
LOCATION	Yes	Yes
NRMTFILE	Yes	No
NRMTLOCS	Yes	No
NSUBTKS	Yes	Yes
IODEV	No	Yes

In addition to these parameters, you need to make sure that your settings are correct for Model 204 storage requirements and for the following Model 204 parameters:

```
FIXSIZE
LFTBL, LQTBL, LRTBL, LSTBL, LVTBL
MINBUF
SPCORE
```

Setting the LOCATION parameter

The LOCATION parameter is a Model 204 User 0 parameter required for *both* clients and servers. It identifies the Model 204 copy as a remote application site within the PQO network.

The syntax of the LOCATION parameter is:

Syntax `LOCATION=`*`name`*

where:

name is required and specifies the locally known name of the PQO network node. The name is a unique value, from 1-8 characters in length.

The LOCATION parameter is used internally, is displayed to identify client users when a service thread issues a LOGWHO command, and is used with BUMP SUBSYSTEM, MONITOR SUBSYSTEM, and STOP SUBSYSTEM commands to identify a service subsystem by its associated client.

This LOCATION parameter value does *not* have to match the value specified for any DEFINE FILE command LOCATION parameter or for any remote file specification (*filename* AT *location*).

During the Online run, the current LOCATION parameter setting can be displayed on the screen using the VIEW command.

LOCATION is nonresettable. See Appendix A for examples of LOCATION parameter settings.

Setting the NRMTFILE parameter

The NRMTFILE parameter is a Model 204 runtime parameter that is required for PQO clients. NRMTFILE determines the number of remote Model 204 files that can be open concurrently to users on the client system.

The syntax of the NRMTFILE parameter is:

Syntax `NRMTFILE=`*`nnnnn`*

where:

nnnnn is a numeric value that specifies the maximum number of remote file save areas allocated by Model 204 for the client. For PQO, the value must be at least one (for the client) in order to access remote files. The maximum value is 16383. The default NRMTFILE value is zero.

A nonzero setting of NRMTFILE is required in the User 0 parameter line of all Onlines that access remote files (all PQO clients). It is not required in an Online that is exclusively a server.

During an Online run, the current NRMTFILE parameter setting can be displayed on the screen using the VIEW command. See Chapter 3 for a description of network monitoring commands.

NRMTFILE is nonresettable. See Appendix A for examples of NRMTFILE parameter settings.

NRMTFILE functions like the NFILES parameter, except that NFILES allocates file save areas for local files. See the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/NFILES_parameter) for a description of the NFILES parameter.

Setting the NRMTLOCS parameter

The NRMTLOCS parameter is a Model 204 runtime parameter that is required for PQO clients. NRMTLOCS specifies the number of remote nodes that can be accessed by client users.

The syntax of the NRMTLOCS parameter is:

Syntax `NRMTLOCS=`*`nnnnn`*

where:

nnnnn is a number from 0 to 2040. The default is 0.

A nonzero setting of NRMTLOCS is required in the User 0 parameter line of all Onlines that access remote nodes (all PQO clients). It is not required in an Online that is exclusively a server.

NRMTLOCS is nonresettable. During an Online run, you can display the current NRMTLOCS setting using the VIEW command.

Setting the NSUBTKS parameter

The NSUBTKS parameter is required for LU 6.2 pseudo-subtasks. The syntax of the NSUBTKS parameter is:

Syntax `NSUBTKS=`*`nnnnn`*

where:

nnnnn is a number value that specifies the maximum number of pseudo-subtasks that can be used in a Model 204 run. For PQO, this number must be increased by two per link if running under z/OS, or by four per link if running under z/VM.

For performance reasons, Horizon and PQO conversations should not share links. However, if the NSUBTKS parameter was previously increased for Horizon links and the PQO network is sharing link definitions with Horizon, it is not necessary to increase NSUBTKS for PQO.

The NSUBTKS parameter is located in the User 0 parameter line in the client and server Onlines. See the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/NSUBTKS_parameter*)* for a description of the NSUBTKS parameter.

See the Model 204 documentation wiki for a description of pseudo-subtasks:

http://m204wiki.rocketsoftware.com/index.php/Controlling_System_Operations_(SDN)

Specifying IODEVs for PQO service threads

In PQO, the server application processes the file locally and sends the data back to the client. The server's Online must provide the PQO service threads, which are defined by a particular class of IODEV statements, to process client requests.

Each IODEV=51 line in a server Online CCAIN stream defines a PQO service thread. You must determine the number of IODEV=51 lines in each server Online that optimally service your client demand.

The syntax of the IODEV parameter line is:

Syntax `IODEV=51,NOTERM=`*`nn`*

where:

- *IODEV=51* is the required user parameter in the CCAIN for the PQO server Online. Each IODEV statement defines a Model 204 user. The total number of IODEV=51 lines determines the maximum number of PQO server applications that can run concurrently.
- *NOTERM* is a parameter that is required on the first user parameter line that specifies IODEV=51. The *nn* value specified indicates the total number of threads allocated for PQO inbound conversations in the server's Online.

  Setting NOTERM close to the total of INLIMIT parameter values on the DEFINE PROCESSGROUP commands for the server system avoids a shortage of PQO inbound threads so that all incoming conversation requests can be processed. See page 17 for a description of the INLIMIT parameter.

See Appendix A for examples of IODEV statements for PQO.

Usage notes on IODEV=51

- IODEV=51 threads are required for a Model 204 Online only if it supports PQO server applications. A client is supported by its local user threads and requires no additional IODEV statements unless it also functions as a server.
- The number of IODEV=51 statements limits the server threads only. In a Model 204 Online supporting both client and server applications, the number of IODEV=51 statements does not affect the number of concurrent conversations that a client can initiate on other remote systems.
- A server Online allocates one IODEV=51 thread for each client user that opens a server file or files. That thread services all the user's requests of that Online. When a client closes its last open file on a server node, the service thread is logged out and made available for another client user.
- When a client is using an IODEV=51 thread, the USERID parameter of the service thread is set to the USERID value of the client Horizon thread. USERID must be specified in CCASTAT or external authorizers such as ACF2, but it is not validated by a password. Client users are not prompted for passwords.
- A Model 204 Online has a pool of IODEV=51 threads to share among all nodes requesting access to files owned by that Online. The number of remote users who can access files on a given node is limited by the number of sessions that can be bound to the service node (the size of the pool) and the number of service threads (NOTERM) that are allocated in the server Online IODEV statement.

  There is no way to explicitly allocate a particular number of service threads to one or more specific client nodes. This capability is implicitly provided through the processgroup entity (see "Using the DEFINE PROCESSGROUP command" on page 14).

SNA Communications Server network definition

PQO communicates over an SNA network through SNA Communications Server by using the LU 6.2 interface of the Model 204 Horizon facility. For z/OS and z/VSE systems, PQO can alternatively communicate using TCP/IP, although this still uses Horizon. See below for an example.

SNA Communications Server communications support must be in place to use PQO. This section describes how to set up the SNA Communications Server network definition with parameters that correspond correctly to the Model 204 network definitions.

The SNA Communications Server setup is similar to that required for Horizon without PQO. See the *Model 204 Horizon: Intersystem Processing Guide* for information about SNA Communications Server network support for Horizon. Figure 2-1 on page 9 shows the basic PQO communications pathway.

Components of SNA Communications Server definition

The host copies of Model 204 used in PQO host-to-host communications are SNA Communications Server application nodes. In a PQO session, both partners (client and server) are SNA Communications Server application LUs.

The SNA Communications Server network definition for PQO includes:

- APPL (application) statement (for each PQO link)
- Mode table (for LU 6.2 session parameters for the network)

Figure 2-2 shows the basic components of the SNA Communications Server definition for PQO. The PQO network shares the existing LU 6.2 table definition (shown as mode table LU62 in the diagram) with the Horizon facility.

Figure 2-2. Components of the SNA Communications Server definition for PQO

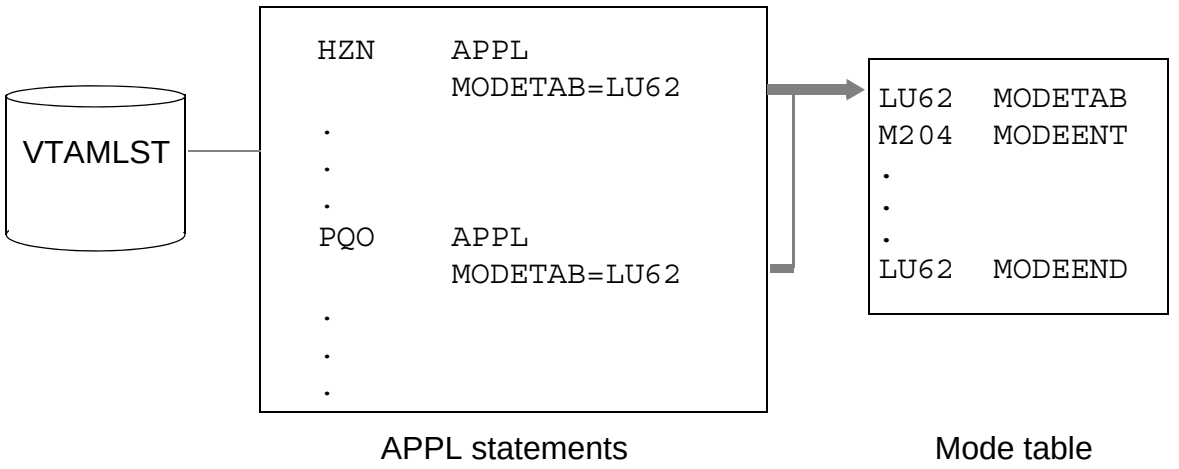


Figure 2-2 on page 24 illustrates a SNA Communications Server definition for two application nodes, one for non-PQO Horizon communications (HZN) and one for Parallel Query Option/204 (PQO). The LU 6.2 mode table shown (LU62) contains a single entry (M204) for Model 204 network session parameters.

Coding the APPL statements for the SNA Communications Server network

In an SNA Communications Server network a separate APPL statement is required for each PQO link entity defined with a Model 204 DEFINE LINK command.

There are three parameters that you must specify in the APPL statement to define SNA Communications Server support for PQO link entities. In addition, there are three optional parameters that you can specify. Other parameters exist for the APPL statement but, because they default to values appropriate for PQO communications, they are not mentioned here.

The required and optional parameters of the APPL statement used for PQO are shown in the following syntax. These parameter settings are the same as those recommended in the *Model 204 Horizon: Intersystem Processing Guide* for typical Horizon connections. Only the MODETAB parameter for PQO requires additional comment.

Syntax

```
name    APPL    AUTH=ACQ,
                PARSESS=YES,
                MODETAB=tablename,
Optional:
                 DLOGMOD=entryname,
                 VPACING=5,
                 PRTCT=password
```

MODETAB is a required parameter. It ties the APPL definition to the mode definition that specifies the particular set of SNA protocols required for the LU 6.2 session between two PQO LUs.

Since PQO uses Horizon protocols, no new table is created for the LU 6.2 session parameters. The MODETAB name for PQO must match the name of the mode table used by the Horizon facility. See the LU 6.2 mode table example on page 26.

Setting session parameters

The set of LU 6.2 session parameters (the mode definition) required for PQO is stored in a mode table for each LU. The server LU's APPL definition points to the mode table, as is done in Horizon.

In PQO communications, the client LU acts as the primary partner in the conversation. The client LU initiates a conversation and acquires a session with the secondary partner, the server LU. The LU 6.2 session parameters are obtained by the client LU from the entry defined for the server, and are then sent to the partner server LU in the BIND message that establishes the session. The client and server LUs share the common set of LU 6.2 rules required to govern the session.

Each PQO link entity supports multiple sessions. A single link entity that supports both client and server processes can act as the primary role partner, or client, on one session and as the secondary role partner, or server, on another session.

The mode table showing the basic set of LU 6.2 session parameters recommended for both Horizon and PQO is shown in the following example.

See the MODETAB parameter remarks on page 25 for a description of the APPL parameter used to reference this table. Refer to the *Model 204 Horizon: Intersystem Processing Guide* for parameter descriptions and for information about modifying the mode table parameters such as RUSIZES and SSNDPAC.

Mode table example

```
table name  MODETAB
entry name  MODEENT  LOGMODE=entryname
                     TYPE=X'00',
                     FMPROF=X'13',
                     TSTPROF=X'07',
                     PRIPROT=X'B0',
                     SECPROT=X'B0',
                     COMPROT=X'50B1',
                     RUSIZES=X'8888',
                     SSNDPAC=X'05',
                     PSERVIC=X'060200000000000000102000'
.
.                (additional table entries, if specified)
.
table name  MODEEND
```

Two-node network example

This section provides sample network definitions for PQO including:

- Model 204 DEFINE commands
- Runtime and user environment parameters
- SNA Communications Server APPL statements, as needed

The sample PQO network supports a Model 204 reporting application for an insurance company requiring communication between two offices in the Boston area. The downtown office produces a policy report using data files from the suburban office.

Each office system operates independently running under its own version of Model 204 in an z/OS environment. The downtown office (Boston1) is the client in the network, requesting information, and the suburban office (Boston2) is the server, providing access to files.

Comparing network types

There are examples below for SNA Communications Server and for TCP/IP. For TCP/IP systems the User 0 parameters and the DEFINE PROCESS parameters are the same, The DEFINE LINK and DEFINE PROCESSGROUP parameters are different as shown, and the APPL statements are not required.

Client input stream excerpt for SNA Communications Server network

```
//CCAIN DD  *
*       User 0 parameters
.
.
.
   NRMTFILE=4,
   NRMTLOCS=1,
   NSUBTKS=4,
   LOCATION=BOSTON1
.
.
.
*     Single link, single process network definition

DEFINE LINK LINK1 WITH SCOPE=SYSTEM LOCALID=DISUSR01   -
   TRANSPORT=VTAM PROTOCOL=LU62 INBUFSIZE=512          -
   SESSIONS=4

DEFINE PROCESSGROUP PGRP1 WITH SCOPE=SYSTEM            -
   LINK=LINK1 OUTLIMIT=48 INLIMIT=0                    -
   REMOTEID=125.20.34.9 LOGIN=TRUST
   PORT=7001    */the SERVPORT value in service Online/*

DEFINE PROCESS CCAD2C WITH SCOPE=SYSTEM                -
   DESTINATION=(PGRP1,OFFICE1)
.
.
.
/*
```

Modifications for client input stream for TCP/IP network

The following DEFINE LINK and PROCESSGROUP definitions are for a TCP/IP system. The other elements in the client input stream remain the same as for the previous SNA Communications Server network.

```
DEFINE LINK LINK1 WITH SCOPE=SYSTEM         -
LOCALID=125.20.34.16                        -
TRANSPORT=TCPSE PROTOCOL=IP SERVPORT=7000   -
INBUFSIZE=4096                              -
CONNECTIONS=50

DEFINE PROCESSGROUP PGRP1 WITH SCOPE=SYSTEM -
LINK=LINK1 OUTLIMIT=48 INLIMIT=0            -
REMOTEID=125.20.34.9 LOGIN=TRUST            -
PORT=7001
```

Server input stream excerpt for SNA Communications Server network

```
//CCAIN DD  *
*       User 0 parameters
.
.
.
   NSUBTKS=6,
   LOCATION=BOSTON2

*     User parameter lines

IODEV=51,NOTERM=4
IODEV=51
IODEV=51
IODEV=51

*     Single link, single process network definition

DEFINE LINK LINK2 WITH SCOPE=SYSTEM LOCALID=DISUSR02   -
   TRANSPORT=VTAM  PROTOCOL=LU62  INBUFSIZE=512        -
   SESSIONS=4

DEFINE PROCESSGROUP PGRP2 WITH SCOPE=SYSTEM            -
   LINK=LINK2  OUTLIMIT=0  INLIMIT=4                   -
   REMOTEID=DISUSR01  LOGIN=TRUST

DEFINE PROCESS CCAD2S WITH SCOPE=SYSTEM                -
   FROM=PGRP2
.
.
.
/*
```

Modifications for server input stream for TCP/IP network

The following DEFINE LINK and PROCESSGROUP definitions are for a TCP/IP system. The other elements in the server input stream remain the same as the previous SNA Communications Server network.

```
DEFINE LINK LINK2 WITH SCOPE=SYSTEM LOCALID=125.20.34.9 -
TRANSPORT=TCPSE PROTOCOL=IP SERVPORT=7001               -
INBUFSIZE=4096                                          -
CONNECTIONS=50

DEFINE PROCESSGROUP PGRP1 WITH SCOPE=SYSTEM -
LINK=LINK1 OUTLIMIT=0 INLIMIT=48            -
REMOTEID=125.20.34.0 LOGIN=TRUST            -
MASK=255.255.255.0
```

APPLO statements required for only SNA Communications Server network

Client APPL statement

The APPL statement for LINK1, coded in BOSTON1's VTAMLST data set, is:

```
DISUSR01  APPL  AUTH=ACQ,
                PARSESS=YES,
                MODETAB=LU62,
                VPACING=5
```

Server APPL statement

The APPL statement for LINK2, coded in BOSTON2's VTAMLST data set, is:

```
DISUSR02  APPL  AUTH=ACQ,
                PARSESS=YES,
                MODETAB=LU62,
                VPACING=5
```

3

Managing the Parallel Query Option/204 Network

Overview

After you have defined the Parallel Query Option/204 network as described in Chapter 2, you need to be aware of certain tasks involved in managing the daily operation of the network. These tasks are performed by the network administrator, system administrator, or system manager, and include:

- Controlling the network
- Monitoring network activity
- Using the Model 204 audit trail
- Interpreting communications errors

Controlling the network

You use the following types of Model 204 commands to control the PQO network:

- LU 6.2 network administration commands
- BUMP system control command

These commands are described in the sections that follow.

Using the LU 6.2 network administration commands

The basic set of network administration commands used to control the operation of the PQO network are also used for Horizon. The commands shown in Table 3-1 are useful for controlling utilization of resources in the network.

Table 3-1. Network administration commands

Table <x>-1.

Command	Description
[OPEN | CLOSE] LINK	Enables/disables the link specified in a DEFINE LINK command (allocates/releases internal resources). If a link is closed after a service subsystem has been successfully started, that subsystem remains available to the client (although the link must be reopened before the client can reaccess it).
[STOP | START] LINK	Halts/resumes operation of the link specified in a DEFINE LINK (deactivates/activates sessions).
[STOP | START] PROCESSGROUP	Halts/resumes operation of a processgroup specified in a DEFINE PROCESSGROUP command (enables processgroup redefinition). If a processgroup is stopped after a service subsystem has been successfully started, that subsystem remains available to the client (although the processgroup must be restarted before the client can reaccess it).
MODIFY PROCESSGROUP	Modifies an operational PROCESSGROUP definition (overrides current parameter settings).

The Model 204 network administration commands require User 0, system manager, or system administrator privileges. See the *Rocket Model 204 Horizon: Intersystem Processing Guide* for information about using the LU 6.2 network administration commands. That volume serves as the primary reference for the commands.

Using the OPEN LINK command

Use the OPEN LINK command to establish the connections for a PQO conversation. Both the link on the client and the corresponding link on the server must be opened before a client user can access remote server files. The syntax of the OPEN LINK command is the standard Horizon network OPEN LINK syntax. For information about using the OPEN LINK command, see the *Rocket Model 204 Horizon: Intersystem Processing Guide*.

Using the BUMP command

Use the BUMP command to log out a PQO user or to log out all users of a subsystem. For more information about the BUMP command, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/BUMP_command).

Bumping a user

The BUMP command operates differently depending on whether it is issued by a client or a server system manager.

If the *client* issues a BUMP command for a user with an open remote file, the client and service threads are freed in the following way:

Client
1. Soft restart processing for the user begins.
2. Files opened by the user are closed.
3. The user is logged out.
4. Soft restart processing for the user ends.

Server
1. The user's service thread is logged out.
2. Soft restart processing for the user begins.
3. Local files opened by the user are closed.
4. The user is logged out.
5. Soft restart processing for the user ends.

If the *server* issues a BUMP command for a remote user with a server file open, the service and client threads are freed in the following way:

Server
1. Soft restart processing for the user begins.
2. Local files opened by the user are closed.
3. The user is logged out.
4. Lost session message is issued.
5. Soft restart processing for the user ends.

Client
1. User query of the remote file receives lost session and communication error messages.
2. User can open remote file again.

Bumping all users of a subsystem

To bump all users of a specified subsystem, use the SUBSYSTEM argument of the BUMP command. The syntax is as follows:

Syntax `BUMP SUBSYSTEM subsysname`

Model 204 schedules all users in the subsystem to be bumped, except those in nonbumpable states such as BACKOUT.

You can also BUMP all local users of a remote service subsystem. Use the client LOCATION name with BUMP SUBSYSTEM. The syntax is as follows:

Syntax `BUMP SUBSYSTEM subsysname FROM location`

where:

location is the value of the client CCAIN LOCATION parameter (which is also the value of the CLNT field in the LOGWHO command output), or is the symbolic name specified by the process definition.

Monitoring network activity

The Model 204 commands used to check on the activity in the PQO network are shown in Table 3-2.

Table 3-2. Network monitoring commands

Command	Description
DISPLAY GROUP	Displays information about Model 204 groups
LOGWHO	Identifies the users who are currently logged in to the Model 204 system
MONITOR	Provides usage statistics for each service thread currently open in the network; also provides information about active local and remote subsystems
STATUS	Displays information about remotely opened Model 204 files
VIEW	Specifies the current setting for the named parameter

These commands can be used system-wide. These commands also provide information that identifies users, files, groups, or settings that are particular to the PQO network, as described in the sections that follow.

Using the DISPLAY GROUP command

The DISPLAY GROUP command can display the name and member files of any or all locally created scattered groups. It does *not* display definitions for groups that were created on other nodes.

The names of remote files displayed are their file synonyms or remote file specifications (*filename* AT *location*).

In addition, the display might include:

- *OPTIONAL* (in parentheses) following each optional member. *MANDATORY* is not displayed if the member is mandatory.

  For more information about optional and mandatory group members, see “Creating ad hoc scattered groups” on page 48.

- MAXFAIL parameter (CREATE GROUP command) value, if it has a value other than asterisk (*).

- UPDTFILE parameter value, including the update file’s location if its location was used in the CREATE GROUP definition.

Using the LOGWHO command

The LOGWHO command lists the users who are currently logged in to Model 204. For PQO, the LOGWHO command operates differently depending on whether it displays information for a client or a server system.

For each PQO user, LOGWHO shows the remote files currently open. Model 204 includes the following LOGWHO information for a PQO user:

Client	Displays the user ID and the name of the open remote file using the full remote file specification (*filename* AT *location*)
Server	Displays the client user number and user ID, and also displays name of subsystem into which service thread is logged, if any

For example, the LOGWHO command displays the line below for a PQO user on a client system. The user has two files open: a local file, DEMOINV, and a remote file, SALES1, at the BOSTON location:

```
USER 5 SYSADM  L3270621  DEMOINV  SALES1 AT BOSTON
```

The values in this example come from the following sources:

Value	Source
USER 5	User thread number displayed in Model 204 journal/audit trail user statistics (see the Rocket Model 204 documentation wiki: http://m204wiki.rocketsoftware.com/index.php/Tracking_system_activity_(CCAJRNL,_CCAAUDIT,_CCAJLOG))
SYSADM	User's Model 204 login user ID
L3270621	User's SNA Communications Server Communications Server (formerly VTAM) terminal ID or z/VM machine name
DEMOINV	Name of a local file open on the client
SALES1	File's name as specified in a Model 204 CREATE FILE command on the server
BOSTON	Node on which the remote file resides; a symbolic destination name specified in a client DEFINE PROCESS command for CCAD2C

In the following example, a LOGWHO command is issued for a PQO service thread. The display includes the following line for a client user. The line includes the service thread's originating client node, the user number of the client thread, and the name of the subsystem into which the client user is logged.

```
USER 34 MGRXYZ  XXXXXXXX  SALES1  CLNT: 00005 ON CHICAGO
APSY: DEMOSALES
```

The values in this example come from the following sources:

Value	Source
USER 34	User thread number of service thread as displayed in Model 204 journal/audit trail user statistics
MGRXYZ	Client user's Model 204 login user ID
XXXXXXXX	Value of local DEFINE PROCESSGROUP command REMOTEID parameter
SALES1	Name of a local file open on the server
CLNT: 00005	User number at the client thread that is currently using the service thread MGRXYZ (only if user is a client)
ON CHICAGO	Value of User 0 LOCATION parameter in the client's Model 204 CCAIN
APSY: DEMOSALES	Name of APSY subsystem into which client thread is logged (only if USER 34 is a service thread)

The LOGWHO command requires system administrator privileges. See the Rocket Model 204 documentation wiki

(http://m204wiki.rocketsoftware.com/index.php/LOGWHO_command) for a description of the LOGWHO command.

Using the MONITOR command

The MONITOR command can provide statistics on the current usage of a network entity: link, processgroup, or process. MONITOR can also provide information about active PQO subsystems. This section describes details specific to the PQO output lines.

The MONITOR commands of particular interest for PQO are summarized in Table 3-3.

Table 3-3. MONITOR commands

Command	Displays...
MONITOR LINK	Single summary line, and a detail line for each bound session
MONITOR PROCESSGROUP	Single summary line, and a detail line for each active conversation
MONITOR PROCESS	Detail line for each active conversation
MONITOR SUBSYSTEM	List of active subsystems and information about those subsystems

The MONITOR command for network entities operates the same for PQO as it does for the Horizon network. It requires User 0, system manager, or system administrator privileges. See the *Rocket Model 204 Horizon: Intersystem Processing Guide* for more information about MONITOR for network entities.

MONITOR SUBSYSTEM output reports whether subsystem members are optional or mandatory, and it has an option (FROM *location*) that allows PQO users to specify exactly the service subsystem monitored. Ordinary user privileges are required.

MONITOR LINK statistics

Monitoring link activity in the PQO network produces the following difference in the output lines compared to the Horizon statistics: The PROCESS term in the detail line for PQO specifies CCAD2S for the server process or CCAD2C for the client process.

The PROTO term in the summary line specifies LU 6.2 as the type of communications protocol for PQO, the same as is used for Horizon.

MONITOR PROCESSGROUP statistics

Monitoring processgroup activity in the PQO network produces differences in the detail output line compared to the Horizon statistics. The PROCESS term in the detail line for PQO specifies:

- CCAD2S for an inbound conversation
- CCAD2C for an outbound conversation

MONITOR PROCESS statistics

Monitoring process activity in the PQO network produces differences in the detail output line compared to the Horizon statistics. The CID (conversation ID) term for a PQO conversation ID specifies:

- CCAD2S (for a server), with the FLGS term set for I (inbound)
- Concatenation of the processgroup REMOTEID and LINK values (for a client), with the FLGS term set for O (outbound)

MONITOR SUBSYSTEM command

To display information about a PQO service subsystem, a server node user specifies the *client* subsystem name and LOCATION name with the MONITOR SUBSYSTEM command. The format is as follows:

Syntax `MONITOR SUBSYSTEM (optionlist) subsysname FROM location`

where:

- *optionlist* and *subsysname* are the standard Model 204 MONITOR SUBSYSTEM variable arguments.
- *location* is the value of the client CCAIN LOCATION parameter (which is also the value of the CLNT field in the LOGWHO command output), or is the symbolic name specified by the process definition.

You can also use pattern matching:

```
MONITOR SUBSYSTEM (optionlist) LIKE 'pattern' FROM 'loc-pattern'
```

loc-pattern is used to match a client LOCATION parameter. An asterisk (*), meaning all nonlocal locations, is a valid value for *loc-pattern*. If you use pattern matching, you must specify a pattern for both the subsystem name and location, and you can only match remote location names.

The MONITOR SUBSYSTEM display includes subsystem member locations (if remote) and whether members are optional or mandatory. A client subsystem display also includes items that are not relevant for the server, such as the storage used for resident requests and whether client subsystem files and groups are automatically or manually opened.

For example, this command produces the output that follows:

```
MONITOR SUBSYSTEM (ALL) SALES FROM BOSTON

SUBSYSTE NAME:  SALES FROM BOSTON
```

```
SUBSYSTEM STATUS:  ACTIVE
NUMBER OF USERS:     1
NUMBER OF PRECOMPILABLE PROCEDURES (SAVED):  1
FILE:  BUYERS
       STATUS=OPEN,NUSERS=1,DFRD UPDATE=N,MANDATORY
FILE:  REG1
       STATUS=OPEN,NUSERS=1,DFRD UPDATE=N,OPTIONAL
FILE:  REG2
       STATUS=CLOSED,NUSERS=1,DFRD UPDATE=N,OPTIONAL
FILE:  REG3
       STATUS=CLOSED,NUSERS=1,DFRD UPDATE=N,OPTIONAL
FILE:  TOTLS
```

Using the STATUS command

For PQO, the STATUS command displays recovery information about remote Model 204 files that were opened during a client Online run.

A STATUS command issued by a client user displays a line with the following format for a remotely opened file:

filename:(REMOTE FILE), LOCATION: *name1* ONLINE: *name2*

where:

- *filename* is the actual (nonsynonym) name of the remote file.
- *name1* is the symbolic name of the file's remote location as specified in the DESTINATION parameter of a local DEFINE PROCESS command.
- *name2* is the value of the CCAIN parameter LOCATION in the server Online.

For more information about location references, see "Client process DESTINATION parameter" on page 18 and "Setting the LOCATION parameter" on page 20. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/STATUS_command:_Listing_file_recovery_information) for a description of the STATUS command.

Using the VIEW command

Use the VIEW command display the current settings of Model 204 parameters. In PQO remote file contexts, the VIEW command is not reliable for all file parameters.

In remote file context, reliable VIEW command results are guaranteed only for the following parameters:

```
ASTRPPG
ATRPG
```

```
CURFILE
CURLOC
FICREATE
FILEORG
FITRANS
HASHKEY
LOCATION
OPENCTL
RECSCTY
SORTKEY
```

Similarly, for VIEW FPARMS or VIEW TABLES in remote file context, correct information is guaranteed only for the file parameters listed above.

The following examples are of remote file context VIEW command displays for individual file parameters:

- VIEW LOCATION displays the value of the CCAIN LOCATION parameter of the Model 204 copy to which the file you are in belongs.

  For example, VIEW LOCATION in remote file context displays a line with the following format:

  ```
  LOCATION name  D204 LOCATION
  ```

  name is the value specified in the LOCATION parameter in the remote Online's User 0 input stream. See "Setting the LOCATION parameter" on page 20 for a description of this parameter.

- VIEW CURFILE displays the name of the file you are currently in.

  For example, VIEW CURFILE in remote file context displays a line with the following format:

  ```
  CURFILE name CURRENT FILE
  ```

  name is the name given to the file when it was created. No file synonyms or remote file specifications are displayed.

- VIEW CURLOC displays the current file's location.

  For example, VIEW CURLOC in remote file context displays a line with the following format:

  ```
  CURLOC name CURFILE LOCATION IF REMOTE
  ```

  name is the symbolic name of the file's remote location as specified in the DESTINATION parameter of a local DEFINE PROCESS command.

  If the current file is local, VIEW CURLOC displays a line with the following format:

  ```
  CURLOC name (LOCAL)
  ```

Using the Model 204 audit trail

Technical Support recommends that PQO clients and servers run with a Model 204 audit trail activated. The audit trail can be printed directly by a Model 204 run without requiring a separate job step. It is invaluable for debugging application programs and for analyzing system performance.

For your own diagnostic purposes or when reporting debugging problems to Technical Support, set the Model 204 SYSOPT parameter on server nodes to include audit trail RK lines. The RK lines record calls to Host Language Interface functions, and PQO service threads make use of Model 204 HLI calls to access data.

When you need to conserve audit trail space, turn off the RK lines.

For more information about the SYSOPT parameter, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/SYSOPT_parameter).

Interpreting communications errors

When you use or attempt to use PQO communications, Model 204 generates messages that track the network activity. Model 204 writes these messages to the system's audit trail.

If an error occurs involving PQO communications, conversation processing is not started or is stopped, and Model 204 automatically displays status codes and sometimes an error message on the client user's terminal or server operator's console. Model 204 also writes the error message to the audit trail of the client or server on which the error occurred.

PQO errors that involve the underlying communications layer are handled differently from application level errors. For a description of error handling at compilation and evaluation time, see "Errors during compilation and evaluation" on page 64.

Communications error display format

Each PQO error condition involving the underlying communications layer is displayed as one or two error messages. One of these messages contains the status codes. The other line, if any, contains a Model 204 message that helps to describe the condition that caused the error.

An error message that includes the status codes has the following format:

```
M204.nnnn: COMM[UNICATION] ERROR  STATUS=nn, STATUSD=nn
```

The STATUS value refers to the status of the communications statement last executed. This value is a number value greater than or equal to zero. A value of two or greater indicates that a communications error occurred and identifies the error as belonging to a particular category.

STATUSD is set to the return code that details the specific error condition within a particular STATUS category.

STATUS and STATUSD refer to the SOUL $STATUS and $STATUSD function values described in the *Rocket Model 204 Horizon: Intersystem Processing Guide*.

Using the status return codes

The STATUS and STATUSD codes that correspond to PQO communications errors are displayed for informational purposes only and cannot be manipulated by the PQO user in a SOUL procedure.

This limitation on use of the status return codes applies only to communications errors.

Using the Model 204 error message

The Rocket Model 204 messages documentation provides a description of Model 204 errors associated with PQO error conditions. Wherever possible, refer to this messages documentation for an explanation of the associated error message.

If an error message is encountered for which no description is available, contact Technical Support for assistance.

4

Managing Files and Groups for Parallel Query Option/204

Overview

This chapter describes how to set up a symbolic file name definition for client users and how to set permission to access a remote file at a server location. These tasks are performed by the file manager.

The chapter also includes a description of how to define groups that include remote files and how to stop and start remote files and scattered groups.

For information about working with remote files and scattered groups in application subsystems, see Chapter 6.

Defining file synonyms

Parallel Query Option/204 supports the use of symbolic names for referencing a remote file. To use a symbolic file name, a *file synonym* definition (DEFINE FILE command) is required.

The DEFINE FILE command maps a file synonym defined by a local user against the actual name of a local or remote file, using the location where the file resides.

Once the DEFINE FILE command has been issued, local users can refer to the file using the synonym name, as described in “Using a file synonym” on page 57.

DEFINE FILE command

The syntax of the DEFINE FILE command is:

Syntax

```
DEFINE  FILE name [LIKE previousname] WITH
        SCOPE=SYSTEM
        FILENAME=actualfilename
        LOCATION={locationname | '='}
```

where:

Parameters

- *name* is required, and is the symbolic name (file synonym) assigned to the file. The name must be from 1-8 characters in length. If *name* is already in use as a file synonym for this copy of Model 204, DEFINE FILE *name* gives *name* a new definition.
- *LIKE* is optional and gives the file synonym currently being defined the attributes of the file referred to by *previousname*, where *previousname* is a symbolic name that was previously defined.

  If used, *LIKE previousname* assigns an additional file synonym, and the FILENAME and LOCATION parameters are not required.
- *SCOPE=SYSTEM* is required and indicates that this definition is available to all users of this copy of Model 204 for the entire length of the run.
- A *FILENAME* setting is required if *LIKE previousname* is not specified in the definition. *actualfilename* is the file's name as specified in a Model 204 CREATE FILE command on the node to which the file belongs.
- *LOCATION* is required if *LIKE previousname* is not specified in the definition. *locationname* refers to the location of the node on which the actual file resides.

  An equal sign (=) between single quotation marks indicates that the location is the client node: meaning, the file is local.

  The location name also might be required to match a value that is specified in the DESTINATION parameter of a local CCAD2C process definition: If the CCAD2C DESTINATION name is specified in *processgroup-symbolic name* format, *locationname* must match a specified symbolic name.

  For example, corresponding to a CCAD2C process definition that includes:

  ```
  DESTINATION=(PGRP1,BOSTON)
  ```

  the system manager defines a synonym for a remote file by specifying the following location name in a file definition:

  ```
  LOCATION=BOSTON
  ```

  For more information about the DESTINATION parameter, see page 18.

Using the DEFINE FILE command

The DEFINE FILE command requires User 0 or system manager privileges, and can be issued in any of the following ways:

- Inside the CCAIN input file in the User 0 stream of the client copy of Model 204
- At the user terminal, with the necessary privileges
- Inside a SOUL procedure, with the necessary privileges

You may use DEFINE FILE to define multiple synonyms for the same file, and you may reissue DEFINE FILE with the same name in the same Online or batch run. However, you cannot use multiple synonyms for the same file in a group or subsystem definition.

Redefining file synonyms for group members

Since Model 204 maps file synonyms when the first user opens a permanent or temporary group, changes to file synonyms have no effect on permanent groups previously opened in the run or temporary groups previously opened in the session. If you want to redefine a file synonym for a member of an already open group, you must:

1. Issue a new DEFINE FILE command.
2. Close the group.
3. Delete the group.
4. Issue a new CREATE GROUP command.
5. Open the group again.

Creating scattered groups

PQO supports file groups consisting of files residing on different nodes. A group that contains remote files is a *scattered group*. Model 204 ad hoc, temporary, and permanent groups can be scattered groups. With no additional requirements, ad hoc groups allow remote file synonyms and remote file specifications. With some additional parameter requirements, temporary and permanent groups allow remote file synonyms and remote file specifications.

Availability, a file status that concerns scattered groups, refers to whether a group member can be opened and, if not, whether the group can be opened. See “File and group availability” on page 49.

Scattered group members are *optional* or *mandatory*. If a mandatory member is unavailable, the group cannot open. You specify whether members are optional or mandatory in the group definition. See “Creating ad hoc scattered groups” on page 48.

A scattered group is a local entity—you cannot create a group for a remote node, and you cannot refer to a group defined on a remote node. As for all Model 204 groups, you define a scattered group with the CREATE GROUP command or the SOUL IN clause. The authority required to define a scattered group is no different from that required for a nonscattered group.

It is important to remember that before you create any Model 204 permanent group, scattered or not, you must create a CCAGRP file with the CREATEG command, as described in the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Storing_and_using_file_group_definitions_(CCAGRP)

For more information on the CREATEG command, see:

http://m204wiki.rocketsoftware.com/index.php/CREATEG_command

CREATE GROUP command

The CREATE GROUP command syntax and scattered group parameter options are described as follows:

Syntax

```
CREATE [PERM | TEMP] GROUP groupname
 FROM {filename [OPTIONAL | MANDATORY]} ,...
 [PARAMETER parameter[=value] [, parameter[=value]] ...]
 [PARAMETER parameter[=value] [, parameter[=value]] ...]
.
.
.
END
```

where:

Parameters

- If neither *PERM* nor *TEMP* is specified, the default *TEMP* is assumed.
- If you are specifying remote files, *filename* must include a remote location (AT *location*) or be a defined file synonym.

  You cannot specify multiple identifiers for the same file—you cannot include two synonyms for the same file, and you cannot include a synonym and a remote file specification for the same file. Duplicate identifiers for a local file are detected when Model 204 tries to open the group. A Model 204 message tells you the name of file duplicates that result from using file synonyms.
- *OPTIONAL* or *MANDATORY* indicates whether group operations can proceed in the absence of the member. The default is *MANDATORY*. Valid abbreviations are OPT and MAND. You cannot specify *OPTIONAL* for local files.

  If too many optional files are unavailable, group operations cannot proceed. See the MAXFAIL parameter, below.

You cannot open a group if a mandatory member is unavailable.

- *parameter* specifies Model 204 group parameters. Any number of PARAMETER clauses can follow, and each can contain as many parameters in any order as can fit on one input line.

  Valid parameters are described in the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/CREATE_command:_Permanent_group or http://m204wiki.rocketsoftware.com/index.php/CREATE_command:_Temporary_group). All but PRIVATE, PUBLIC, and SEMIPUB must be followed by a *value*.

  PARAMETER clause *PROCFILE* parameter (if present) must refer to a local file. You cannot access or use remotely stored procedures. If the group is scattered, the value of *PROCFILE* cannot be an asterisk (*).

  PARAMETER clause *MAXFAIL* parameter specifies the maximum number of optional members that can be unavailable. If more than *MAXFAIL* files are unavailable, the group fails to open or operations on the group are aborted. The value of *MAXFAIL* must be one of the following:

  – Integer between one and the number of optional group members.

  – Asterisk (*), which is equivalent to the number of optional members in the group. This setting means that the group can function without any of its optional members. This is the default.

  For ad hoc scattered groups, MAXFAIL is always set to the default.

  The PARAMETER clause *BLDGFT* parameter is ignored for groups that include remote members.

  The PARAMETER clause *UPDTFILE* parameter can refer to either local or remote files. If the update file you are naming is a remote file, you can include its remote file specification (AT *location*).

CREATE GROUP example

The following CREATE GROUP command creates a permanent scattered group with four local members and four remote members. Five members are mandatory, and the MAXFAIL setting (*) is equivalent to MAXFAIL=3 (the group has three optional members). The update file is a remote member.

```
CREATE PERM GROUP GRP1                                          -
 FROM INSURE89, OWNERS AT DALLAS(MAND), AUTOS AT DETROIT,       -
 DETTMP1 AT DETROIT(OPT), DETTMP2 AT DETROIT(OPT),              -
 DALTMP3 AT DALLAS(OPT), CLNTPROC(MAND), INSURE90,              -
 PARAMETER PROCFILE=CLNTPROC, UPDTFILE=AUTOS AT DETROIT,        -
 MAXFAIL=*                                                      -
END
```

Creating ad hoc scattered groups

You use the following IN clause syntax to create an ad hoc scattered group:

Syntax

```
IN file [(OPTIONAL | MANDATORY)],
  {file [(OPTIONAL | MANDATORY)]} ,...
```

- If you are specifying remote files, *file* must include a remote location (AT *location*) or be a defined file synonym.

  You cannot specify multiple identifiers for the same file—you cannot include two synonyms for the same file, and you cannot include a synonym and a remote file specification for the same file.

- *OPTIONAL* and *MANDATORY* are the same as in “CREATE GROUP command” on page 46.

In the ad hoc group created in the following example, CLIENTS is a local file, CARS is a remote file synonym, and SALES AT DETROIT is a remote file specification:

Example and discussion of an ad hoc scattered group

```
IN CLIENTS, CARS, SALES AT DETROIT (OPTIONAL)
```

If you want to refer to an ad hoc group you created earlier in a request, you must repeat the earlier IN clause. Model 204 considers two ad hoc groups as duplicates if all the following are true:

- Their IN clauses contain the same set of files.
- Their IN clauses list the files in the same order.
- Corresponding files in their IN clauses are both optional or both mandatory.

The following IN clause is issued later in the same request as the previous example. Model 204 considers this ad hoc scattered group a new group, because the SALES file below is mandatory (by default), not optional:

```
IN CLIENTS, CARS, SALES AT DETROIT
```

Because this group is not a duplicate, it requires extra server table space and extra network traffic.

File and group availability

Unlike typical Model 204 file groups, scattered groups do not require that all members of the group can be opened (*available*) at the time the group is opened and for all subsequent operations. In the CREATE GROUP command, you specify the members, if any, that are optional. With the MAXFAIL parameter of CREATE GROUP command, you specify how many of the optional files in the group can be unavailable before group processing is stopped.

The terms in Table 4-1 describe file or group availability with respect to a user. For more information about availability for files and groups that are members of application subsystems, see Chapter 6.

Table 4-1. File or group status terms

Term	Description
available	The file is an open member of an open group.
closed	The file or group is not currently open. If a CLOSE command is issued for a disabled file, the file's status is changed to *closed* to the user.
disabled	If communications with the user's service thread fails, all remote files on that node are *disabled* for the user. If the MAXFAIL parameter for an open group is exceeded, or if a mandatory member is unavailable, the group becomes *disabled* for the user.
open	When a remote file is opened in single file or group context, its status is *open*.
unavailable	If an optional file in an open group fails to open or becomes disabled, the file is *unavailable* to the group for the user who has the group open. All group context operations behave as if the unavailable member is not a part of the group, with the exception of ON units specifically designed to handle such situations.

Allowing access to remote files

Before users can access a remote file, the file must be defined as a transaction backout (TBO) file, and it might require a new OPENCTL parameter setting.

This section also discusses using the PQOSYS parameter to generate record security keys for remote users.

If the file is a member of a service application subsystem, the server system manager must determine what kind of access to the file to allow. This is discussed in Chapter 6.

Parallel Query Option and Large Object data

The Parallel Query Option does not support access to actual Large Object data—neither BLOBs nor CLOBs. For PRINT ALL INFORMATION statements of a remote record, the Table B portion of the Large Object field is displayed, but not the data itself. In this way, you can identify remote records that have Large Object fields, although you will need to access the file locally to work with the Large Object data.

Defining transaction backout files

PQO allows only transaction backout files to be remotely accessed. Any attempt to open a non-TBO file in remote context fails.

To enable the transaction backout mechanism for a file, turn off bit settings in the file parameters FOPT and FRCVOPT, as described in the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Transaction_back_out#Transaction_back_out_file_parameter_settings).

Setting the OPENCTL parameter

For files that are not application subsystem members, PQO file security is the same as Model 204 file security except for remote files that you do not want to protect with a password.

For such files, public or semipublic, you must use a special setting of the OPENCTL parameter. Otherwise, remote public files do not open and remote semipublic files do not open without a correct password.

Normal Model 204 file security settings apply to remote *private* files. The OPENCTL parameter is part of the file definition statement that specifies a password option at file open time. It is set at the server node initially at file creation. Its settings do not affect application subsystem files.

Use one of the OPENCTL settings in Table 4-2 for remote access of non-subsystem public and semipublic files on a server.

Table 4-2. OPENCTL settings

Setting	Meaning	Password handling
X‘08’	File may be accessed remotely (PQO) *without* valid password	If the file is public, no password is requested and the file is opened remotely with default privileges. If the file is semipublic, a password is requested. If the password is missing or invalid, the user is granted default file privileges. If the file is semipublic or private and a valid password is presented, the file is opened only if the X‘02’ bit is on.

Table 4-2. OPENCTL settings (Continued)

Setting	Meaning	Password handling
X‘04’	File can be accessed remotely (PQO) as permanent group member	No effect.
X‘02’	File can be accessed remotely (PQO) *with* a valid password	The user who specifies a valid password for the private or semipublic file is allowed to open the file remotely.

OPENCTL can be reset after file creation.

OPENCTL is ignored when a file defined in an application subsystem is opened within that subsystem. File privileges are specified in the subsystem definition.

The PQO remote access settings (X‘08’, X‘04’, X‘02’) have no affect on files that are only accessed locally.

See the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/OPENCTL_parameter or http://m204wiki.rocketsoftware.com/index.php/File_design#File_security_levels) for more information about the OPENCTL parameter.

Record security for remote users—the PQOSYS parameter

The new CCAIN SYSTEM parameter PQOSYS lets you direct Model 204 to generate a special record security key for remote users.

PQOSYS has two settings:

- X‘00’ (the default)

  Your system manager is responsible for assigning unique record security IDs to all users and subsystems with remote access to files with record security. If IDs are not unique, record security might fail to distinguish users from different nodes with the same user ID (for example, SMITH from BOSTON might access records stored by SMITH from CHICAGO).

- X‘80’

  Model 204 provides a special record security key for remote users.

  The STORE, FIND, and FOR RECORD NUMBER statements use a record security key that is a concatenation of the user’s record security ID (either their login user ID or their record security key from a subsystem definition) with a period (.) followed by the value of the CCAIN LOCATION parameter of the user’s node.

  Trailing blanks are stripped from both the record security ID and the LOCATION part of the record security key.

  For local users, the record security key is the user’s usual record security ID (without any concatenation).

Setting PQOSYS to X‘80’ requires extra disk space, because the value of each record security key stored by the system includes the period sign and the LOCATION parameter from the remote node. Setting PQOSYS to the default minimizes disk space use.

PQOSYS is set in the User 0 part of the CCAIN stream *on the node where the file is located*. It cannot be reset after the Model 204 run is initialized.

Explicitly stored record security keys

If you are explicitly storing an occurrence of the record security field (with SOUL, HLI, or a File Load program) for a user who accesses the file remotely, you need to append a period and the value of the LOCATION parameter of the other user’s node to the usual record security ID (the user’s login user ID or the record security key of the user’s application subsystem).

Record security fields with the BINARY or FLOAT attribute

If the record security field is BINARY or FLOAT and PQOSYS is X‘80’, Model 204 stores as character strings the record security values stored by remote users. If the file is also a NUMERIC VALIDATION file, remote users cannot store records in this file and only those remote users who are allowed to override record security can retrieve records from the file.

Record security keys stored by local users are stored in the BINARY or FLOAT form.

Stopping and starting a remote file or scattered group

You can stop (prevent new opens of) a remote file by issuing the STOP FILE command using the AT location clause or a file synonym. You can stop a permanent scattered group with STOP GROUP.

A stopped file or group cannot be opened until it is restarted.

In remote as in nonremote contexts, a user can continue to reference a file or group that was already open when a STOP FILE or STOP GROUP command is issued.

This section describes the effect of STOP and START commands in remote contexts.

- For complete information about STOP and START commands, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/STOP_command:_Making_a_file_or_group_unavailable and http://m204wiki.rocketsoftware.com/index.php/START_command:_Making_a_file_or_group_available).
- For information about STOP SUBSYSTEM and START SUBSYSTEM, see Chapter 6.

Using STOP FILE on a server node

If a STOP FILE command issued on the server refers to a local file that was opened remotely by a client, the file is stopped to all server and client users until started. However, if the file is a mandatory member of an active subsystem (service or nonservice), or a mandatory member of a group that is a mandatory member of an active subsystem, the STOP command fails.

Model 204 frees all precompiled requests and any saved compilations for the file that are on the local node or on the client node after all current users have closed the file. If the file is being used by an active subsystem, it is marked unavailable for future use.

Using STOP FILE on a client node

If STOP FILE issued on the client refers to a remote file, the file is stopped for users on the client node only. However, if the file is a mandatory member of an active subsystem or a mandatory member of a group that is a mandatory member of an active subsystem, the STOP command fails.

If a stopped file was used by a client subsystem that locks its files, STOP unlocks the file, and it can be opened by any user of this client Online after it is started.

Model 204 does not free any compilations on the client or server nodes nor any precompiled requests referring to the stopped file.

If a stopped remote file belongs to a subsystem, the file's file save area is not freed for reuse until after the subsystem is stopped. If the subsystem is not stopped, subsequent starting and opening of the file uses an additional file save area. Be sure to allocate enough file save areas (NRMTFILE is sufficiently large) to avoid being prevented from opening remote files.

Using STOP GROUP

Since a group can be accessed only by local users, a STOP GROUP command can be issued only for a locally defined group, which might be scattered. Also, a STOP GROUP command cannot be used for a temporary scattered (or nonscattered) group.

If a STOP GROUP command is issued for a local scattered group, the group is stopped for all users, and Model 204 frees any compilations on the client or server nodes that refer to the group after all current users have closed the group.

If a scattered group is a member of a subsystem, STOP GROUP does not free the file save areas of remote members of the group.

START FILE and START GROUP

START FILE and START GROUP commands reverse the effects of STOP FILE and STOP GROUP commands, respectively. After a START FILE or START GROUP command is issued, a remote file can be opened.

A START GROUP command cannot be used for a temporary scattered (or nonscattered) group.

5

Working with Remote Files and Scattered Groups

Overview

With Parallel Query Option/204, the user can access files that are located on remote Model 204 systems. Special guidelines apply when working with files in remote context.

This chapter describes the specific commands, statements, and functions that are available to the Model 204 application programmer, and the rules that apply when referencing files in remote context.

Unless otherwise specified, use these commands, statements, and functions in PQO contexts the same as in typical Model 204 contexts.

Specifying a remote file

You can explicitly reference a remote file in a command or statement using:

- Remote file specification
- File synonym

You can also implicitly reference a remote file by using the name of a scattered group of which the file is a member. This section describes explicit remote file specification.

Using a remote file specification

A command or statement that references a file by name and includes an AT clause is a *remote file specification*. The AT clause references the node to which the file belongs (where the file was created). This node can be local or remote.

Remote file specification is valid with all the commands and statements that reference local files except for the DEFINE DATASET command. Remote file specification has the following syntax:

Syntax `[command | statement] filename AT [location | = ]`

where:

- *filename* specifies the actual name of a Model 204 file on the node where the file resides.
- *location* is the symbolic name (as many as eight characters) that refers to the location of the node on which the file resides.
- An equal sign without quotation marks (=) specifies a client node, meaning the file is local.

AT *location* matches DEFINE PROCESS destination

The remote file specification location name must match a symbolic destination name specified in the DESTINATION parameter of a local DEFINE PROCESS command for CCAD2C.

For example, if the definition of a CCAD2C process includes:

```
DESTINATION=(PGRP1,BOSTON)
```

the following OPEN FILE command using the AT clause is valid:

```
OPEN VEHICLES AT BOSTON
```

where VEHICLES is the name of a file residing at the remote location that is identified by the symbolic name BOSTON. The DESTINATION parameter is discussed on page 18.

AT location not required to match CCAIN parameter LOCATION

The remote file specification location name need *not* match the value of the server node CCAIN parameter LOCATION. See “Setting the LOCATION parameter” on page 20.

Using a dummy string in the AT clause

The location name in the AT clause can be coded as a dummy string inside a SOUL procedure. At run time, the user can enter an equal sign (=) to specify

that the location of the named file is the node on which the command or statement is currently issued, the local node.

Using a dummy string allows the application programmer to code a single file reference in the procedure, and to access different versions of the same file (having the same name), either locally or remotely, at run time. See the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Procedures#Dummy_strings_in_procedures) for a description of dummy strings in SOUL procedures.

Using a file synonym

Once a DEFINE FILE command has been issued for a particular remote file, a user can refer to the file using the synonym name. The synonym is the symbolic name assigned to the file and is available to local users. See “Defining file synonyms” on page 43.

Even though a file synonym has been defined, you still can use the full remote file specification, as described in “Specifying a remote file” on page 55. When using a file synonym in a file command, the syntax of the command for a remote file is the same as for local files. For example, the following DEFINE FILE command specifies the synonym CARS for the file VEHICLES:

```
DEFINE  FILE CARS WITH
        SCOPE=SYSTEM
        FILENAME=VEHICLES
        LOCATION=BOSTON
```

You can specify an OPEN FILE command, using the file synonym, as:

```
OPEN CARS
```

Using a file synonym provides name and location transparency in SOUL applications. You can reference a file using only a locally assigned name. The system manager maintains the file definition in keeping with any changes in the network definition such as the location specification. Changes to the network definition do not require changes to the application code.

Using Model 204 file and group commands

A remote file or group specification, using either the AT clause or a file synonym, is syntactically valid in most of the same commands that are used to refer to local files or groups. The Model 204 commands you can use to refer to a remote file are shown in Table 5-1.

Table 5-1. Commands for remote files or scattered groups

Command	Type of user
CLOSE (for a file or group)	Any user
CREATE PERM GROUP	System manager

Table 5-1. Commands for remote files or scattered groups

Command	Type of user
CREATE TEMP GROUP	Any user
DEFAULT	Any user
DEFINE FILE	User 0 or system manager
DELETE PERM GROUP	System manager
DELETE TEMP GROUP	Any user
DISPLAY (for a field, group, or record)	Any user
MONITOR SUBSYSTEM	Any user
OPEN FILE	Any user
OPEN GROUP	Any user
START (for a file or group)	System administrator
STATUS	Any user
STOP (for a file or group)	System administrator
VIEW	Any user

These are the only file or group commands that support remote file specification.

Supported in PQO but not included in Table 5-1 are BUMP, LOGWHO, MONITOR, the network control and definition commands, and the subsystem control commands. See Appendix B for a list of unsupported commands. Most of the commands in Table 5-1 are described in Chapter 3 and Chapter 4. The following commands are described in this section:

```
OPEN FILE
OPEN GROUP
CLOSE (for a file or group)
DEFAULT
DELETE GROUP
```

Opening a remote file

You open a remote file by issuing the OPEN FILE command using either the AT location clause or a file synonym. You can also use an OPEN GROUP command if the file is a member of a scattered group. See “Opening a scattered group” on page 60 for a discussion.

Successful attempts to open

When you first successfully issue a command to open a remote file at a given node, a conversation is initiated with that server and a service thread is

activated to support the conversation. If a conversation is already established with that remote node and is available, the conversation continues and is used for the request to open the file.

If a conversation is available, you are prompted to enter a password if the file is private or semipublic. If the OPENCTL setting for the file allows remote access, any password gains access to a semipublic file. Private files require the correct password. Application subsystem files are protected by APSY security. See “Allowing access to remote files” on page 49 for a description of the OPENCTL parameter.

When a remote file is opened, Model 204 displays a confirmation message with the file’s name (nonsynonym) and location, stating whether updates are allowed.

Unsuccessful attempts to open

If a conversation cannot be allocated, Model 204 displays the error message:

```
M204.1984: COMMUNICATION ERROR ON REMOTE NODE PROCESSING
FILE filename AT location
```

If an OPEN FILE or OPENC FILE statement issued from within a SOUL request fails, the $STATUS return codes indicate the error condition. A $STATUS value of 2 is returned for an open failure. For more information about $STATUS return codes, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Files,_groups,_and_reference_context).

File context

You can open a file in single file context or in permanent group context. A file open as a member of a temporary group is considered open in single file context. You can open in a different context an already open file. Therefore, a remote file can be open in any or all of the following contexts at the same time:

- Single file
- Permanent group
- Multiple permanent group

File locking behavior

The locking behavior of the OPEN FILE command for a remote file is identical to that for a local file with regard to:

- Shared direct access storage devices (DASD)
- System level resources
- Application subsystems

For information about resource locking, see the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Resource_locking

Field definition is restricted

When the OPEN FILE command is successfully executed and a remote file is opened, the status of the file is *open*. The remote file's Table A is stored on the client system for as long as the file remains open.

While a remote file is open to *any* remote user, the fields cannot be redefined, deleted, or renamed on the server. Server Online users receive a message that the file is in use. New fields, however, can be defined.

Opening deferred update data sets

A file that is *not* in deferred update mode can be opened remotely.

A file that is in deferred update mode can be opened remotely only when the file is a member of a service application subsystem. If a remote deferred update data set is specified in the OPEN FILE command, the open is rejected.

Opening a scattered group

You open a remote file that is a member of a scattered group by issuing the OPEN GROUP command. OPEN GROUP for scattered temporary and permanent groups has the same syntax as for nonscattered groups. Password requirements and privileges for member files are also the same for scattered and nonscattered groups.

In scattered group context, OPEN GROUP causes Model 204 to allocate conversations with each of the nodes that contain members of the scattered group, if such conversations do not already exist.

If scattered groups are defined to allow update privileges, all member files are opened with the indicated privileges when the group is successfully opened. For each remote file opened in group context, Model 204 displays a confirmation message with the file's name (nonsynonym) and location, stating whether updates are allowed.

Unsuccessful attempts to open

If a mandatory member of a scattered group cannot be opened, the OPEN operation fails for the entire group.

If an optional member of the group cannot be opened, an error message is issued stating that the file is unavailable. The OPEN operation continues as if the file were not included in the group definition. Storage is still allocated in various internal data structures, however.

If the CREATE GROUP command MAXFAIL parameter value is exceeded, the group open fails and any successfully opened files are closed. An error message is issued stating that more than MAXFAIL optional files cannot be opened.

If all a scattered group's files are optional and unavailable, a group open fails, even if MAXFAIL is not exceeded. An error message is issued stating that the open failed because all group members are missing.

Using an OPEN GROUP statement

If an OPEN GROUP command is issued as a SOUL statement (from within a request), the result of the open operation is indicated with one of three $STATUS values:

$STATUS	Description
0	All members of the group were successfully opened.
1	One or more optional members could not be opened.
2	A mandatory member could not be opened, MAXFAIL members could not be opened, or all members are missing.

For more information about $STATUS, see the *Rocket Model 204 Horizon: Intersystem Processing Guide*.

Closing a remote file

You can close a remote file by issuing the CLOSE command using the AT location clause or a file synonym, using CLOSE ALL, or, if the remote file is the default file, using CLOSE by itself. If the file is a member of a permanent scattered group, you can close the files in the group with CLOSE GROUP.

When a CLOSE command for a remote file is successful, the file is closed on the server node on which the file resides. Model 204 displays a message confirming the close to the user on the client node. The message identifies the file name and location.

If the file being closed is your last open file on the server node, the conversation with the server is terminated.

A loss of communications with a remote node during or just prior to your attempt to close a remote file results in the following:

- File is closed on your local node, and you receive a communications error message. The service thread is restarted, which closes the file on the server.
- If the file is a member of a scattered group and you issue CLOSE GROUP, the file is closed on your local node, and Model 204 attempts to close as usual the rest of the files in the group.

- If the file was disabled for you prior to the CLOSE, the file is closed only on your local node.

Using DEFAULT and DELETE GROUP

The Model 204 DEFAULT command, which establishes the current default file or group, applies to remote file synonyms, remote file specifications, and scattered groups.

The Model 204 DELETE GROUP command, which deletes an existing permanent or temporary group, applies to scattered groups.

Referencing remote files in a SOUL request

SOUL compilation is initiated when Model 204 encounters a BEGIN command. A request that refers to a remote file is compiled on both client and server nodes. Model 204 initiates compilation on the remote server system when a reference to a remote file is first encountered.

During compilation, as remote references are encountered, the client system sends data manipulation language (DML) calls to the participating server node, or nodes, to be compiled.

SOUL example

The following example shows an excerpt from a SOUL procedure that is submitted locally and references a Model 204 file (named VEHICLES) residing at a remote location (whose symbolic name is defined as BOSTON).

Prior to accessing a remote file, the following conditions must be met:

- You must define the PQO network on both the client and server systems.
- Model 204 system manager (or User 0) must successfully issue the OPEN LINK command on both the client and server systems, so that the connection for the PQO conversation is enabled.

```
DEFINE FILE CARS WITH SCOPE=SYSTEM   -
    FILENAME=VEHICLES                -
    LOCATION=BOSTON
OPEN CARS
.
.
.
BEGIN
.
.
.
FD: IN CARS FD
    MAKE=FORD
    END FIND
```

```
.
.
.
FR: FR IN FD
    PAI
    END FOR
.
.
.
END
```

Input/output operations that involve network transmissions are entailed in using the OPEN, FIND, and FOR statements to reference a remotely located file. In the previous SOUL example:

- OPEN
  - The user request to open a remote file (CARS) initiates activity at the remote location (BOSTON) where the file resides.
  - The server sends a copy of the Model 204 file information (Table A) to the user's (client) node so that compilation can proceed.
- FIND (FD)
  - The client sends FIND criteria (FORD vehicle records) to the remote location for evaluation.
- FOR EACH RECORD (FR)
  - The client processes records using a FOR EACH RECORD loop.
  - The server sends records from the found set to the user's node, a page at a time.

"SOUL example" on page 62 shows remote file references using a file synonym. As an alternative, if no DEFINE FILE command is issued on the client system, you can reference the remote file using a remote file specification (AT clause).

For example, instead of OPEN CARS, you use:

```
OPEN VEHICLES AT BOSTON
```

Instead of IN CARS FD, you use:

```
FD: IN VEHICLES AT BOSTON FD
```

See "Defining file synonyms" on page 43 and "Specifying a remote file" on page 55 for additional information.

Request continuation is not supported

Remote file reference is *not* supported in a SOUL request continuation. The SOUL MORE and END MORE statements are not supported if the continued request refers to a remote file.

Errors during compilation and evaluation

Compilation and evaluation errors in PQO applications are discussed in the following sections. Both sections discuss how Model 204 handles non communications errors and communications errors when a remote file is referenced.

Compile time error handling

If a non communications error occurs during compilation when a client application references a file on a server node:

- Model 204 prints on the client node the current line in the client procedure and the error message that was generated on the server node.
- Compilation continues on the client node only, regardless of the number of errors encountered on server nodes.

If a communications problem occurs during compilation, in any remote file or group context:

- Compilation is terminated.
- All files at the referenced node are disabled for the client user.

Evaluation time error handling

Model 204 begins to execute a SOUL request even though that request contains a reference to a remote file or group that was already made unavailable to the request prior to evaluation. The action Model 204 takes depends on the context of the file or group referenced in the request:

In this context...	**Model 204 takes this action...**
Remote single file	ON MISSING FILE unit invoked
Scattered group	ON MISSING FILE unit invoked
File member of scattered group	ON MISSING MEMBER unit *not invoked*

If a non communications error occurs on an already opened remote file, the error message generated at the remote node is printed on the client node, and the request is canceled.

If you lose communications during evaluation with an already opened remote file, all files at that node are disabled for you, and Model 204 makes one of three responses:

- Takes no further action, if all of the following are true:
  - File is in group context
  - File is an optional group member
  - MAXFAIL parameter setting in CREATE GROUP is not exceeded
  - Node to which the file belongs has no uncommitted updates
- Invokes one of two ON units:
  - ON MISSING MEMBER
  - ON MISSING FILE
- Invokes the ON ERROR unit, if present, and cancels the request.

Using ON MISSING MEMBER and ON MISSING FILE units

The specific behavior of the ON MISSING MEMBER and the ON MISSING FILE units is described below. For a code example with these ON units, see “ON MISSING FILE unit example” on page 83.

Using the ON MISSING MEMBER unit

ON MISSING MEMBER handles errors involving the availability of scattered group members. It obeys the same scoping rules as all other ON units. Its format follows:

```
ON MISSING MEMBER
.
.
.
END ON
```

ON MISSING MEMBER is invoked whenever the following are all true:

- Operation in group context fails against a remote optional member of the group.
- Member was not previously unavailable to the request.
- MISSING FILE condition is not raised.

When ON MISSING MEMBER is invoked, these consequences follow for all the user’s member files at that server node:

- Group member is made unavailable to the group, both during and after request execution.
- Group member is considered disabled.

- Subsequent references to the group in the same request for files at that server do not invoke ON MISSING MEMBER.

If an operation in group context fails against a remote optional member of a group and no ON MISSING MEMBER unit is active, the member is made unavailable to the group and the operation completes as if the member were not part of the group definition.

When control is passed to an ON MISSING MEMBER unit, you can return control to the request with a RETRY, CONTINUE, BYPASS, or JUMP TO statement.

ON MISSING MEMBER units cannot be nested.

PQO supports use of a CLEAR statement for clearing of an ON unit for ON MISSING MEMBER.

Communications failures during ON MISSING MEMBER

If a communications failure occurs while communicating with a different node within an ON MISSING MEMBER unit, the ON ERROR unit is invoked or the request is canceled.

If a communications failure occurs while receiving a new portion of records from a remote node in a FOR EACH RECORD loop in group context, you can use one of these statements to tell PQO how to proceed:

- CONTINUE—Causes the FOR loop to continue processing records from the next available group member. All unavailable files are skipped, but the remaining available files are processed.
- BYPASS—Exits the FOR loop, bypassing the remaining group files against which the FOR loop would have run, and skips to the next instruction after END FOR.

Using the ON MISSING FILE unit

ON MISSING FILE handles errors involving the availability of remote files in single file context, and involving mandatory and MAXFAIL conditions of remote files in group context. It obeys the same scoping rules as all other ON units. Its format follows:

```
ON MISSING FILE
.
.
.
END ON
```

ON MISSING FILE is invoked for any of the following circumstances:

- Operation in single file context fails against a remote file.

- Operation has failed against a remote file when there was no specific file or group context. This includes the following SOUL statements:

  RELEASE ALL RECORDS

  BACKOUT

- An operation in group context fails against a remote mandatory member.
- An operation in group context fails against a remote optional group member and the number of missing members is greater than the MAXFAIL parameter (CREATE GROUP), or all members of the group are now missing.

When ON MISSING FILE is invoked, these consequences follow:

- File or group is made unavailable to the request.
- File or group is disabled for the user.
- All future references to the file or group in the same request cause the request to be canceled.

If an operation that normally calls for an ON MISSING FILE unit fails, and no MISSING FILE unit is active, the ON ERROR unit is invoked. If no ON ERROR unit is active, the request is canceled.

When control is passed to an ON MISSING FILE unit, you can return control to the request with a BYPASS or JUMP statement. RETRY is not allowed.

ON MISSING FILE units cannot be nested.

PQO supports use of the CLEAR statement for clearing of an ON unit for ON MISSING FILE.

Communications failures during ON MISSING FILE

If a communications error failure occurs while communicating with a different node within an ON MISSING FILE unit, the ON ERROR unit is invoked or the request is canceled.

The special handling for BYPASS and CONTINUE in a FOR EACH RECORD loop in group context described above for communication failures during ON MISSING MEMBER does not apply to ON MISSING FILE units, because the entire group is made unavailable.

DML statements in PQO

The SOUL statements and conditions in Table 5-2 comprise the PQO data manipulation language (DML).

Only these statements and conditions can be used in reference to a remote file or group.

Following the table are discussions of the use of these statements and conditions in PQO, maintaining the classification (retrieval or update) in Table 5-2.

Table 5-2. PQO DML statements and conditions

Retrieval statements	Retrieval conditions	Update statements
CLEAR LIST	FIND$	ADD
COMMIT RELEASE	FILE$	BACKOUT
COUNT OCCURRENCES OF	LIST$	CHANGE TO
COUNT RECORDS	LOCATION$	COMMIT
FIND RECORDS	POINT$	DELETE
FIND ALL RECORDS	SFGE$	DELETE EACH
FIND ALL VALUES		
FIND AND PRINT COUNT	SFL$	DELETE RECORD
FOR EACH OCCURRENCE		DELETE RECORDS
FOR EACH RECORD		FILE RECORDS
FOR EACH VALUE		
FOR *k* RECORDS		INSERT
FOR *k* VALUES		
FOR RECORD NUMBER		STORE RECORD
NOTE *fieldname*		UPDATE RECORD
OPEN		
OPENC		
PLACE RECORD ON LIST		
PLACE RECORDS ON LIST		
PRINT ALL INFORMATION		
RELEASE ALL RECORDS		
RELEASE RECORDS		
REMOVE RECORD FROM LIST		
REMOVE RECORDS FROM LIST		
SORT RECORD KEYS		
SORT RECORDS		
SORT VALUES		

Using IN clauses

A remote file specification (using either the AT clause or a file synonym) or a scattered group is syntactically valid in the IN FILE or IN GROUP clause of supported DML statements in remote context.

Also, you can use the following IN clause variations to refer to a remote file if you are in scattered group context:

```
IN GROUP groupname MEMBER
IN $CURFILE
IN $UPDATE
```

Using field names in expressions

Fields in remote context can be used in the same ways as fields in local context including the VALUE IN phrase, which refers to a label on a FOR EACH RECORD IN ORDER statement.

Handling record locking conflicts

If a client request cannot complete because of a record-locking conflict on the server system, the server automatically tries again to lock the record or set of records. The server tries again until it succeeds or until it has tried as many times as the value of the *client thread* ENQRETRY parameter. The value of the ENQRETRY parameter that is specified on the server thread has no effect on the number of retries.

If ENQRETRY attempts to lock a record or set of records do not succeed, the server notifies the client about the conflict. If an ON RECORD LOCKING or ON FIND CONFLICT unit is active, the unit is invoked. Otherwise, the client receives a message that the locking failed, followed by a prompt asking if the client wants to try again.

If the client enters N, the request is canceled. If the client enters Y, the server repeats the locking attempt cycle, making as many as ENQRETRY attempts before prompting again.

Using PQO retrieval statements

This section describes the individual SOUL retrieval statements and any usage restrictions that you can use with a remote file or group.

Using the FIND ALL RECORDS statement

All forms of the FIND ALL RECORDS statement are supported in remote file and scattered group context.

A FIND statement in remote context produces a record set on each of the server nodes to which the statement refers.

All record locks are maintained on the server nodes. If a record-locking conflict occurs, partial found sets are dequeued (on several nodes if necessary), and the normal FIND CONFLICT action is taken.

See the discussion of LOCATION$ on page 74 for information about restricting group context FIND statements.

Using the FOR EACH RECORD statement

All forms of the FOR EACH RECORD statement are supported in remote file and scattered group context except for the IN ORDER clause, which is not supported in scattered group context.

Except for the following cases, when a FOR statement is executed, it retrieves from each record only the fields that are referred to in the loop. Data is handled differently by the server in the following cases:

- If the FOR loop references a field name variable, such as %%*fieldname*, or if it contains a PAI statement, the entire record is transmitted from the server node.
- If a subscripted field reference is used in a FOR loop, all occurrences of the field are transmitted from the server node.

A CODED field is decoded before being transmitted from the server node.

OPTIMIZING FNV option

Use the OPTIMIZING FNV option with the FOR EACH RECORD statement to optimize retrievals with field name variables.

You can use the OPTIMIZING FNV option (OPTIMIZING can be abbreviated as OPT) to prevent FOR loop retrievals with field name variables from triggering the retrieval of all fields and all occurrences.

To use the OPTIMIZING FNV option with the FOR EACH RECORD statement, you must specify OPT FNV in one of the following places:

- Before the WHERE/WITH option and after all other options
- At the end of the FOR statement when the WHERE/WITH option is not present

When you use OPTIMIZING FNV, only the initial value of the field name variables at the start of the FOR loop are used to select the fields that are retrieved and shipped to the client node. Therefore, the field name variable *must* have the proper value at the time the FOR loop is first executed. Each time the FOR loop is entered (after the loop has completed), the field name variable values are reset.

If the field name variable is changed in the FOR loop to the name of another field that was not explicitly referenced, the other field is *not* retrieved and a default value of null is used for the field name variable.

Using the FOR RECORD NUMBER statement

All forms of the FOR RECORD NUMBER statement are supported in remote file and scattered group context, although FOR RECORD NUMBER and FOR RECORD NUMBER IN *label* might have different results if you are working with records that are not locked.

FOR RECORD NUMBER IN *label* refers to a preceding FOR EACH RECORD statement. In PQO, the records retrieved for the FOR EACH RECORD loop are copied and sent to the client when the FOR EACH RECORD statement is evaluated. If the records are not locked after the retrieval, by the time the FOR RECORD NUMBER IN *label* statement is executed, the record copy to which you are referring might no longer match the record at the server.

However, if you use FOR RECORD NUMBER without the IN label, Model 204 locks the record to which you are referring and sends a copy to the client during FOR RECORD NUMBER execution. Only the fields referred to by FOR RECORD NUMBER are sent.

OPTIMIZING FNV option

You can also use the OPTIMIZING FNV option with the FOR EACH RECORD NUMBER statement.

If you do so, the information provided in “Using the FOR EACH RECORD statement” on page 70 is valid, with this exception—you must specify the OPTIMIZING FNV option at the end of the FOR RECORD NUMBER statement.

Using the FOR EACH VALUE statement

All forms of the FOR EACH VALUE statement are supported in remote file and scattered group context.

The FOR EACH VALUE statement initiates a loop that is executed once for each unique value of the specified field. There are several variations of the FOR EACH VALUE statement, as described in the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Value_loops).

You can use the FOR EACH VALUE statement to:

- Sort the retrieved values, using the IN ORDER clause
- Specify a range of values to be retrieved, using FROM *value1* TO *value2* clauses
- Specify a pattern to be used to retrieve values, using the LIKE pattern clause
- Process a sample of values, using the *k* clause

Using RELEASE

The RELEASE ALL RECORDS and RELEASE RECORDS IN/ON statements are performed on the server system, releasing any remotely held found sets.

Using the SORT RECORDS statement

The SORT RECORDS statement is supported in remote file and scattered group context.

Note: The SORT RECORDS *k* option is ignored. If you use SORT k RECORDS, all the records referred to, not just the first *k* records, are sorted.

Sort operations for remote file data are executed on each affected server node and the results are merged. Sort keys can include fields that are not defined in all members of a scattered group.

Using LIST functions

List processing functions are supported in remote context. List operations for remote file data are performed on the server node on which the file resides. For scattered groups, a list is created on each node that contains a member of the group.

The following segment of SOUL code has comments to show the resulting conversation elements:

```
IN PARTS AT DETROIT BEGIN
CLEAR LIST XYZ            (Send CLEAR LIST function to DETROIT)
A: FD PTYPE='XYZ'              (Send FIND function to DETROIT)
B: FR A                        (Retrieve records)
   PLACE RECORD ON LIST XYZ (Send PLACE RECORD function to
                                DETROIT)
   .
   .
   .
```

Note: PLACE RECORDS IN A ON LIST XYZ has the same effect, but requires only one network operation.

Using the PRINT statement

The PRINT *ID statement *cannot* be used in remote file context. All other forms of the PRINT statement can be used to print field values for remote files, with the exception of Large Object Data. You can print the locator information stored in Table B, but not the actual data stored in Table E.

Printing preallocated fields with a PAI statement

PAI or PAI INTO normally displays preallocated fields first in its output. In PQO, however, some update operations store new preallocated field values at the end of the record. This placement at the end of the record is displayed by PAI.

Fields in a remote file update are transferred to the local node in the nonpreallocated format. Some updates to preallocated fields that take place within a FOR loop (for instance, ADD) store new values at the end of the record.

If preallocated fields are not updated within a FOR loop, the output of the PAI statement is the same as in the nondistributed case.

Using retrieval conditions

Except as described below for FILE$ and LOCATION$, retrieval conditions act the same in remote context as they do locally. In scattered groups, retrieval conditions can include fields that are not defined in all members of the group.

The following retrieval conditions are supported in remote context:

```
FILE$
FIND$
LIST$
LOCATION$
POINT$
SFGE$
SFL$
```

Using FILE$

The FILE$ condition is valid only in group context. It can accept remote file specifications and remote synonyms. For example, the following FIND is restricted to values in the STUDENTS file whose location is NYC:

```
IN GROUP STUDENTS FD FILE$ 'STUDENTS' AT NYC AND SEX = 'F'
```

Valid FILE$ argument formats for remote files are:

- Literal remote file specifications or file synonyms
- Literal remote file specifications or file synonyms following dummy strings
- Equal sign (=) to point to the node on which the request is running

Invalid FILE$ argument formats for remote files are:

- %Variables
- Remote file specifications enclosed entirely in quotes (for example, 'STUDENTS AT NYC' is not valid)

Using LOCATION$

You can use the LOCATION$ condition to restrict group context FIND statements to a particular node. Used like FILE$, LOCATION$ is valid only in group context.

The location referred to by LOCATION$ is the location name used in file synonyms and remote file specifications—the symbolic name specified in the DESTINATION parameter of a client DEFINE PROCESS command (described on page 17 and page 56).

Valid LOCATION$ argument formats are:

- Literal location names
- Literal location names following dummy strings
- Equal sign (=) to point to the node on which the request is running

LOCATION$ arguments cannot be %variables.

If an optional member of a scattered group is unavailable, no records are found for that file and processing continues.

Using PQO update statements

In PQO, you typically execute single-node updates; you update data on only one node per Model 204 transaction. This is the default.

If the current transaction has updated a node, either local or remote, that is different from the node referred to by another update statement within the transaction, Model 204 displays an error message stating that you violated the single-node update rule. The current transaction is backed out and the request is canceled. This restriction is checked at evaluation time.

An application program must end the current transaction (either commit it or back it out) before the application can update files on a different node (either remote or local).

This section discusses the SOUL update statements that you can use with a remote file or group and includes a discussion of updating records that are not locked.

Limitations to updating remote files

- There is no support in Parallel Query Option for the date/time stamp feature. If you attempt a remote open of a data/time stamp file (FOPT=X’10), the following message is issued:

  ```
  M204.1977: %F MAY NOT BE ACCESSED REMOTELY
  ```

- You cannot open a file remotely and update fields defined with the BLOB or CLOB file attribute.

PQOOPT and multiple-node updates

For the SOUL DELETE RECORDS and FILE RECORDS statements only, you can enable multiple-node updates with the Model 204 parameter PQOOPT.

For more information about multiple-node updates, see “Using DELETE RECORDS” on page 77 and “Using FILE RECORD” on page 77.

Updating unlocked record sets

In a typical PQO remote update, a client issues a FIND statement to retrieve a set of records from a service node file. A client FOR EACH RECORD statement triggers the transfer of a copy of the record set to the client node. The client application loops through and updates the records on the service node and the client node copy.

If the service node records are not locked (for example, you use FIND WITHOUT LOCKS) during the update processing, other users can make changes on the service node to the records in the copied set. Changes that happen after the set is copied and before the update is finished might introduce inconsistencies between the client copy and the actual server data. Such changes make obsolete the client copy of the record set.

For example, some field-level operations depend on the current state of a record: if there are three occurrences of a field, CHANGE fieldname(6) *adds* a fourth occurrence. If there are six or more occurrences, the same statement *changes* the sixth occurrence.

A client issues this CHANGE anticipating six or more occurrences. Before the CHANGE is processed, users on the server delete all but three occurrences. The CHANGE is processed and the service node file ends up with three occurrences unchanged and four occurrences altogether; the client copy has no deletions and a changed sixth occurrence.

If the client user prints the record information after the CHANGE but within the same FOR EACH RECORD loop, the display shows the client’s view of the record. The client does not detect that this view does not match the current server data.

If the print of the record information is after the CHANGE but within a new FOR EACH RECORD loop, the display is a refreshed view of the actual current server record. The client can detect at this point that the expected result of the CHANGE did not occur.

Though updates like CHANGE lock a record, they alone do not prevent inconsistencies between the client’s view of the record and the actual state of the record on the server. For unlocked record processing, you need to use FOR RECORD NUMBER.

Lock a record before updating it

Always lock a record before updating it. If an unlocked record set or list is being processed, use the FOR RECORD NUMBER statement to lock the record, as shown next:

```
%X = $CURREC
FOR RECORD NUMBER %X
```

FOR RECORD NUMBER locks, copies, and sends the record to the client when the FOR RECORD NUMBER statement is executed.

Using FOR RECORD NUMBER IN *label* is not sufficient, as explained in “Using the FOR RECORD NUMBER statement” on page 71.

Using ON FIELD CONSTRAINT CONFLICT $functions

If a field constraint violation occurs on a remote node and there is an active ON FIELD CONSTRAINT CONFLICT (ON FCC) unit, clients can issue the $functions that report about the constraint (for example, $UPDSTAT, $UPDOVAL).

For more information about ON FCC $functions, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Files,_groups,_and_reference_context#Using_ON_FIELD_CONSTRAINT_CONFLICT_.24functions).

Using ADD, CHANGE, INSERT, and DELETE statements

The following update statements are supported in remote file and in scattered group context with the limitations explained following the list:

```
ADD
CHANGE
DELETE
DELETE EACH
DELETE RECORD
INSERT
```

Using BACKOUT

The BACKOUT statement backs out a transaction against data on a remote node if the current transaction is remote. In addition to backing out the transaction on the remote node, BACKOUT processing includes backing out the effects of the transaction on the local copy of the remote data.

A remote transaction is backed out automatically in the same situations in which Model 204 backs out a local transaction.

Using COMMIT and COMMIT RELEASE

The COMMIT statement commits a transaction on a remote node if the current transaction is remote. A remote transaction is committed automatically in the same situations in which Model 204 automatically commits a local transaction.

The COMMIT RELEASE statement commits a transaction on a remote node and releases found sets on all nodes, local and remote.

Using DELETE RECORDS

The DELETE RECORDS statement is supported in remote file context and in scattered group context. The functionality of this statement depends on the setting of the PQOOPT parameter, which allows DELETE RECORDS to perform multiple-node updates.

If PQOOPT is not set for multi-node updates, DELETE RECORDS can be used in scattered group context only if either of the following conditions is true:

- All files in the group reside on the same remote node.
- Group found set to which either statement applies is restricted to a single node—the found set was created by:

```
IN GROUP groupname MEMBER %var FIND ...
```

where:

%var specifies a particular file and *%var* is not an asterisk (*).

Using FILE RECORD

The FILE RECORDS statement is supported in remote file context and in scattered group context. The functionality of this statement depends on the setting of the PQOOPT parameter, which allows FILE RECORDS processing to perform multiple-node updates.

If PQOOPT is not set for multi-node updates, a FILE RECORDS statement can be used in scattered group context only if all group members reside on the same node.

If PQOOPT is set for multiple-node updates and an optional member of a scattered group is unavailable, it is skipped. The records are not filed on that node and the old index entries for the targeted field name=value pair are not deleted, but the statement processes successfully.

Using INSERT

The INSERT statement is not supported for Large Object fields.

Using STORE RECORD

All forms of the STORE RECORD statement are supported in remote file and scattered group context.

Field=value pairs specified by statements that are between STORE RECORD and END STORE statements are sent to and stored on the remote node.

After a record is stored on a remote node, you can access it by requests using either FIND or FOR RECORD NUMBER statements.

Using $CURREC, you can extract the record number of the record you most recently stored.

The file in a scattered group in which STORE RECORD processing stores records might not be resolved until evaluation time, so the statement might have to be compiled on multiple nodes. For storage files pointed to by the following IN clauses, STORE RECORD is compiled on all nodes in the group; at evaluation time, it is executed on one node only:

- IN GROUP *groupname* MEMBER, in temporary or permanent scattered group context
- IN $CURFILE, in temporary or permanent scattered group context
- IN GROUP *groupname* (where the group update file is the implied storage file), in temporary scattered group context

Using UPDATE RECORD

The UPDATE RECORD statement improves performance when you are executing a set of field-level update operations against the current record. UPDATE RECORD is allowed in remote and non-remote contexts.

The UPDATE RECORD syntax is:

Syntax

```
UPDATE RECORD
field-level-operation-1
field-level-operation-2
.
.
.
field-level-operation-N
END UPDATE
```

where *field-level-operation* is one of the following:

- ADD *fieldname* = *value*
- DELETE *fieldname* [(*subscript*)] [= *value*]
- CHANGE *fieldname* [(*subscript*)] [= *value*] TO *newvalue*
- INSERT *fieldname* [(*subscript*)] = *value*

All field-level operations between the UPDATE RECORD and END UPDATE statements are packaged and sent to the remote node in one call. Without UPDATE RECORD processing, the same set of operations requires one call per operation.

If there are no field-level operations between the UPDATE RECORD and END UPDATE statements, Model 204 ignores the UPDATE RECORD statement.

Note: The DELETE EACH *fieldname* option is not allowed in an UPDATE RECORD statement.

Usage

Model 204 handles errors that occur during UPDATE RECORD processing the same as those that occur during STORE RECORD processing. For more information about STORE RECORD error handling, see the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Data_maintenance).

If a field constraint violation occurs, the entire UPDATE RECORD statement is backed out. The $UPDSTMT function returns the type of statement that caused the conflict: ADD, CHANGE, or INSERT.

If an ON unit is invoked during the UPDATE RECORD processing and the ON unit is exited with a BYPASS statement, the request execution continues with the statement that follows the END UPDATE statement.

Using $functions

Except for the following functions, all file-related $functions that applied to local files or groups now apply to remote files and scattered groups as well:

```
$LSTPROC
$RDPROC
```

`$VIEW` (works only for certain remote file parameters)

$VIEW is valid in remote file context with the following file parameters:

```
ASTRPPG         FITRANS
ATRPG           HASHKEY
CURLOC          LOCATION
CURFILE         OPENCTL
FICREATE        RECSCTY
FILEORG         SORTKEY
```

If $VIEW is issued in remote file context with any other file parameter, Model 204 processes the request but the result of $VIEW is unreliable.

If $LSTPROC or $RDPROC is coded in a SOUL statement that is in remote context, Model 204 displays an error message at evaluation time.

This section describes $functions that are new or whose use in PQO requires comment.

Using $CURFILE, $RLCFILE, $UPDATE, and $UPDFILE

The output of the $CURFILE, $RLCFILE, $UPDATE, and $UPDFILE functions issued in remote context is the remote file specification (*filename* AT *location*). Each of these functions returns the string:

filename AT *location*

where:

location is a name for the node where the remote file resides.

You can use $SUBSTR to extract the name or the location.

You can use $UPDLOC to determine the location of a file that is being updated. See “Using $UPDLOC” on page 81.

Using $FDEF and $LSTFLD

You can use the $FDEF and $LSTFLD functions in local or remote file context. They recognize the FILE and AT keywords in the *filename* argument string. You also can use a file synonym.

Using $ITSOPEN

The $ITSOPEN function indicates whether the current file or group is open. Its format is:

Syntax `$ITSOPEN (name)`

where:

name is a %variable or literal character string representing a file name, file synonym, or group name. This argument string can include the keywords AT, FILE, GROUP, PERM, or TEMP.

$ITSOPEN can be issued in any context. It returns *1* if the file is open; it returns *0* if the file is closed.

Using $ITSREMOTE

The $ITSREMOTE function indicates whether the current file or group is remote or scattered. Its format is:

Syntax `$ITSREMOTE (name)`

where:

name is a %variable or literal character string representing a file name, file synonym, or group name. This argument string can include the keywords AT, FILE, GROUP, PERM, or TEMP.

$ITSREMOTE can be issued in any context. It returns *1* if the context is remote file or scattered group; it returns *0* if the context is local.

Using $UPDLOC

Use the $UPDLOC function to determine the node location where an UPDATE UNIT is in progress. If the file being updated is local, $UPDLOC returns the word LOCAL. If the file being updated is remote, $UPDLOC returns an identifier for the node location (in the form AT *location*). The $UPDLOC function takes no arguments.

Using file and group availability $functions

The $functions in Table 5-3 on page 82 and Table 5-4 on page 82 provide information about file and group availability. The $functions in Table 5-3 report on scattered group members. The $functions in Table 5-4 are for files or groups but are valid only in an ON MISSING MEMBER or ON MISSING FILE unit.

An example in which these $functions are used is shown on page 83.

The *index* parameter in some of the $functions in Table 5-3 and Table 5-4 is assigned internally by Model 204 when those $functions are issued. Unavailable files are assigned index numbers in the order in which the files are listed in the CREATE GROUP command. For example, if unavailable fileA appears earlier in the CREATE GROUP list of files than unavailable fileB, fileA will have a lower index number than fileB. If fileA appears the earliest in the CREATE GROUP list, it always has the index value *1* when it is unavailable. And when fileA is available, another file will have *1*.

Scattered group $functions

You must use the $functions in Table 5-3 in group context or Model 204 cancels the request within which you issued the $function.

You can ensure the group context by specifying a group name (*groupname* in syntax in Table 5-3) as an argument of the $function. If you do not specify a group name, the context of the current statement is used. The *index* parameter is an integer from one to the value of $GRNMISS. If index is less than one or greater than $GRNMISS, Model 204 cancels the request.

Table 5-3. $functions for groups

$function syntax	Result
$GRNLEFT [(*groupname*)]	Returns the number of remaining available optional group members that can fail before the MAXFAIL parameter value (CREATE GROUP command) is exceeded. (MAXFAIL specifies the maximum number of optional members that can be unavailable.)
$GRNMISS [(*groupname*)]	Returns the number of optional group members that are unavailable to the request.
$GRMNAME (*index* [,*groupname*])	Returns the name of the missing (unavailable) group member represented by the *index* parameter.
$GRMLOC (*index* [,*groupname*])	Returns the location of the missing (unavailable) group member identified by the *index* parameter.

ON MISSING unit $functions

The $functions in Table 5-4 are valid only in an ON MISSING MEMBER or ON MISSING FILE unit. Any other use of these functions is treated as a compilation error. The *index* parameter is an integer from one to the value of $MISNUM. If index is less than one or greater than $MISNUM, Model 204 cancels the request.

Table 5-4. $functions for ON MISSING units

$function	Returns...
$MISGRUP	String containing the name of the group, if the error that invoked the ON unit occurred in group context. Otherwise, it returns the null string.
$MISNUM	Number of files in the group that are newly unavailable (that have become unavailable since the last time the ON MISSING unit was invoked).

Table 5-4. $functions for ON MISSING units (Continued)

$function	Returns...
$MISNAME (index)	Name of the unavailable file identified by the *index* parameter.
$MISLOC (index)	String containing the location of the unavailable file indicated by the *index* parameter.
$MISSTMT	String containing the type of SOUL statement that failed, for example, ‘DELETE ALL RECORDS’.

ON MISSING FILE unit example

This ON MISSING FILE unit example uses the $functions in Table 5-3 and Table 5-4. The example works for an ON MISSING MEMBER unit as well.

```
ON MISSING FILE
 %C IS STRING LEN 255
 %D IS STRING LEN 255
 %C = $MISGRUP
 IF %C = '' THEN
 * This is file context because $MISGRUP returned nulls *
   %C = 'MISSING FILE ' WITH $MISNAME(1) WITH ' AT ' WITH $MISLOC(1)
 PRINT %C
 ELSE
   %GROUPNAME = %C
   %C = 'MISSING GROUP ' WITH %C WITH ' FILES FOLLOW: '
   FOR %JUNK FROM 1 TO $MISNUM BY 1
     %C = %C WITH $MISNAME(%JUNK) WITH ' AT ' WITH $MISLOC(%JUNK)
   END FOR
   %C = %C WITH ' ON STATEMENT ' WITH $MISSTMT
   PRINT %C
   %D = 'COMPLETE LIST OF MISSING FILES FOR' WITH $MISGRUP WITH ': '
   FOR %INDEX FROM 1 TO $GRNMISS BY 1
     %D = %D WITH $GRMNAME(%INDEX,%GROUPNAME) WITH ' AT '
     %D = %D WITH $GRMLOC(%INDEX,%GROUPNAME)
   END FOR
   PRINT %D
 END IF
 BYPASS
END ON
```

Using the Host Language Interface

A Host Language program that issues an IFDIAL call (IFREAD or IFWRITE) can access remote data.

Any functionality that is supported for SOUL requests, as described in this chapter, is supported for IFDIAL calls. There are no additional limitations or functional differences between local and remote file access for IFDIAL calls.

Use of IFSTRT calls is not supported

A Host Language program that issues an IFSTRT call (IFFIND or IFGET) *cannot* access remote data.

An IFOPEN call issued with a remote file specification causes Model 204 to perform the following actions:

- Set the completion code to 260.
- Write an error message to the client system audit trail stating that IFOPEN remote access is not allowed.

6

Parallel Query Option/204 and Scattered APSY Subsystems

Overview

With Parallel Query Option/204, application subsystem definitions and requests can refer to remote files or scattered groups.

This chapter explains how to manage, run, and maintain client and service subsystems that run under the Application Subsystem Facility (APSY). Emphasis is on features and concepts that are specific to Parallel Query Option/204. Basic knowledge of Model 204 subsystem management is assumed.

Required system parameters

The following parameter settings are required to run APSY subsystems with distributed data:

- The SYSOPT=X'01' bit must be set on all nodes that the client subsystem will access.
- The LGTBL parameter on the service thread should be reset to at least 132 whenever a user is logged into a service subsystem. This parameter is used to set the size of the Global Variable Table (GTBL). GTBL contains name/value strings for global variables and is used by procedures included by APSY to support subsystem processing.

Client and service subsystems

PQO provides access to remote files under APSY by allowing the system manager to define client and service subsystems:

- ***Client subsystem*** is the subsystem a user is running in when requesting access to remote data.
- ***Service subsystem*** is the subsystem on a server node that a client user's service thread is logged in to.

A service subsystem definition is stored in the CCASYS file on each node that the client subsystem accesses. The name of a subsystem must be the same at each node. The location of the client node is included in the subsystem name to uniquely identify it to the server node.

Node availability

A server node can be *available* or *unavailable* to a client APSY subsystem.

An APSY node is available if the service subsystem has been successfully started. If the service subsystem has not been started, it does not have a subsystem definition structure accessible to the client and is, therefore, unavailable.

A node can be marked unavailable during start processing only if there are mandatory members on a server node and the service subsystem cannot be started. If this happens, start processing also fails on the client node.

Client subsystems attempting to access service subsystems that are not started receive an error message from the server node.

A previously available node can become unavailable when:

- Resumption of communication fails after recovering from a system failure
- User attempts to log in to the service subsystem by logging in to the client subsystem, the service subsystem definition is not found, and at least one mandatory member resides on that node
- User attempts to open a file on a node where the user was not previously logged in

The user is automatically logged in to all associated service subsystems when entering a subsystem that contains remote files. If the service subsystem is unavailable on a node, the user cannot be logged in.

For details on operational parameters that affect logon processing, see "Operational Parameters screen (service definition)" on page 100.

File and group availability

The *members* of an APSY subsystem are files and permanent groups. With PQO, members can be either automatic or manual:

- An *automatic member* is a subsystem group or file that is opened automatically when the subsystem is started or when a user enters the subsystem.
- A *manual member* is a group or file that must be opened explicitly by the OPEN or OPENC command.

Members can also be either mandatory or optional:

- A *mandatory member* must be open in order to access a subsystem. Mandatory members cannot be manual.
- An *optional member* is not required for subsystem access (start and login processing can succeed without it).

At any given time a member can be *open* or *closed* to a subsystem or to a user within a subsystem. The following sections explain the conditions under which the different kinds of members are accessible to APSY subsystems and their users.

Member availability to APSY subsystems

Automatic members of APSY subsystems are always opened by the START SUBSYSTEM command or by SUBSYSTEM LOGIN. At the end of START processing, each automatic member is open unless either the START or OPEN failed.

Manual members of APSY subsystems are in the closed state at the completion of START SUBSYSTEM processing and must be explicitly opened by the user. Manual members become open to the subsystem if an OPEN or OPENC operation succeeds. If OPEN or OPENC fails due to node unavailability or for user-specific reasons (for example, if the user's line goes down) the member remains closed to the subsystem.

If a node becomes unavailable to a subsystem, all automatic subsystem members and all open manual subsystem members residing on the unavailable node are marked disabled.

If a STOP FILE or STOP GROUP command is issued for a manual member on the client subsystem's node, the member is closed to the client subsystem when the last user closes it. If the member is located on the service subsystem node, the file is closed to the service subsystem when either the STOP is complete or the last user closes the file.

Member availability to subsystem users

When a user enters a subsystem, automatic subsystem members are opened.

If a user LOGIN or OPEN operation fails for an optional member, the member is left closed for the user but remains open to the subsystem. If a mandatory member cannot be opened, the user is denied access to the subsystem.

If a user LOGIN or OPEN operation fails for an already open member, the member is left disabled for the user but remains open to the subsystem.

If an automatic mandatory member is closed to the subsystem, new users are not allowed to enter the subsystem.

Manual members of APSY subsystems are closed for a user within a subsystem until the user issues an OPEN command or statement. In this case it does not matter whether the member is open or closed to the subsystem.

If compilation and/or loading of a request fails due to a communications failure, previously opened members on the failing node become disabled to the user.

A user can close optional members at any time by issuing the CLOSE command.

Enabling a disabled subsystem file

If a file is marked as disabled to the subsystem, you can use the ENABLE SUBSYSTEM FILE command to make the file available again. If necessary, correct any communications problem. Then enter this command:

```
ENABLE SUBSYSTEM subsysname [FILE | GROUP] name -
                AT location
```

where*:*

- *subsysname* is the name of the client subsystem
- *location* is the value of the client CCAIN LOCATION parameter (which is also the value of the CLNT field in the LOGWHO command output)

Note: You must specify the location; you cannot include local files in an ENABLE SUBSYSTEM command.

Disabling a subsystem file

You can disable a subsystem file to keep the file itself, or the subsystem to which it belongs, inaccessible for a short period of time, without shutting down the subsystem by using the STOP SUBSYSTEM command.

To disable a subsystem file, use this command:

```
DISABLE SUBSYSTEM subsysname [FILE | GROUP] name
        AT location
```

where*:*

- *subsysname* is the name of the client subsystem
- *location* is the value of the client CCAIN LOCATION parameter (which is also the value of the CLNT field in the LOGWHO command output)

The consequence of disabling a subsystem file depends on whether the file is an optional or mandatory group member:

- If you disable a mandatory file, you effectively disable the subsystem, because users attempting to access the disabled file cannot access the subsystem.
- If you disable an optional file, users can log in to the subsystem, but cannot access the disabled file.

Trust

As an alternative to the privilege settings normally available through the Subsystem Management facility, each service subsystem manager can control access by specifying that the client subsystem is trusted. If a client subsystem is *trusted*, its definition is unmodified and used as is at the service node (with possible maximum values specified for file privileges and field security).

Why use a trust definition?

Using full or partial trust relieves the system administrator from having to create and maintain the file and SCLASS definitions for remote subsystems. (Full or partial trust requires entries on just one screen, rather than the five screens needed for a subsystem service definition.)

Trust information resides on the service node and is controlled by the service subsystem manager or administrator. For example, the administrator of a service node in Chicago creates and manages trust definitions for any client nodes linked to the Chicago service node.

Parallel Query Option/204 offers four levels of trust:

Table 6-1. Parallel Query Option/204 trust levels

Level	Meaning
Full trust	Only the subsystem name, location, and status (active or suspended) appear on the service node's definition, which you create on the Subsystem Trust screen.

Table 6-1. Parallel Query Option/204 trust levels (Continued)

Level	Meaning
Partial trust	Along with the subsystem name, location, and status, you can specify maximum privileges. The client subsystem is trusted, but the maximum file privileges and field level security levels specified on the Subsystem Trust screen cannot be exceeded: • If a user requests file privileges that would exceed the maximum, the attempt to open the file fails. • If a user requests a field level access that would exceed the maximum, PQO automatically resets the request to the allowed level (the maximum), and opens the file.
Restricted trust	For a subsystem with restricted trust, the subsystem service definition is based on entries you make on these five screens: • Subsystem Activity • Subsystem File Use • Operational Parameters • Subsystem Classes • Subsystem Class Users The client subsystem's access is determined by the SCLASS, user, and file privileges that you specify on these screens. For a subsystem that has restricted trust, you *do not* make any entries on the Subsystem Trust screen. You can specify full or partial trust, *or specify* restricted trust.
No trust	On the service node, if there is neither a subsystem service definition nor a trust definition for the client subsystem. The client subsystem cannot access any files on the service node *as subsystem files*. However, the files on the service node can be accessed from within a client subsystem as individual, nonsubsystem files, if the following criteria are met: • PQO is installed at both sites, and there are link, processgroup, and process definitions connecting the client node to the service node. • Horizon Interface is installed at both sites, and there are link, processgroup, and process definitions connecting the client node to the service node. • OPENCTL parameter setting allows remote access for any given file (with a setting of '08,' '04,' or '02'). For information about the OPENCTL parameter, see the discussion "Allowing access to remote files" on page 49, and consult the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/OPENCTL_parameter).

In Figure 6-1 on page 91, Subsystem D has no trust, because neither an active trust definition nor a subsystem service definition for it exist on the service node.

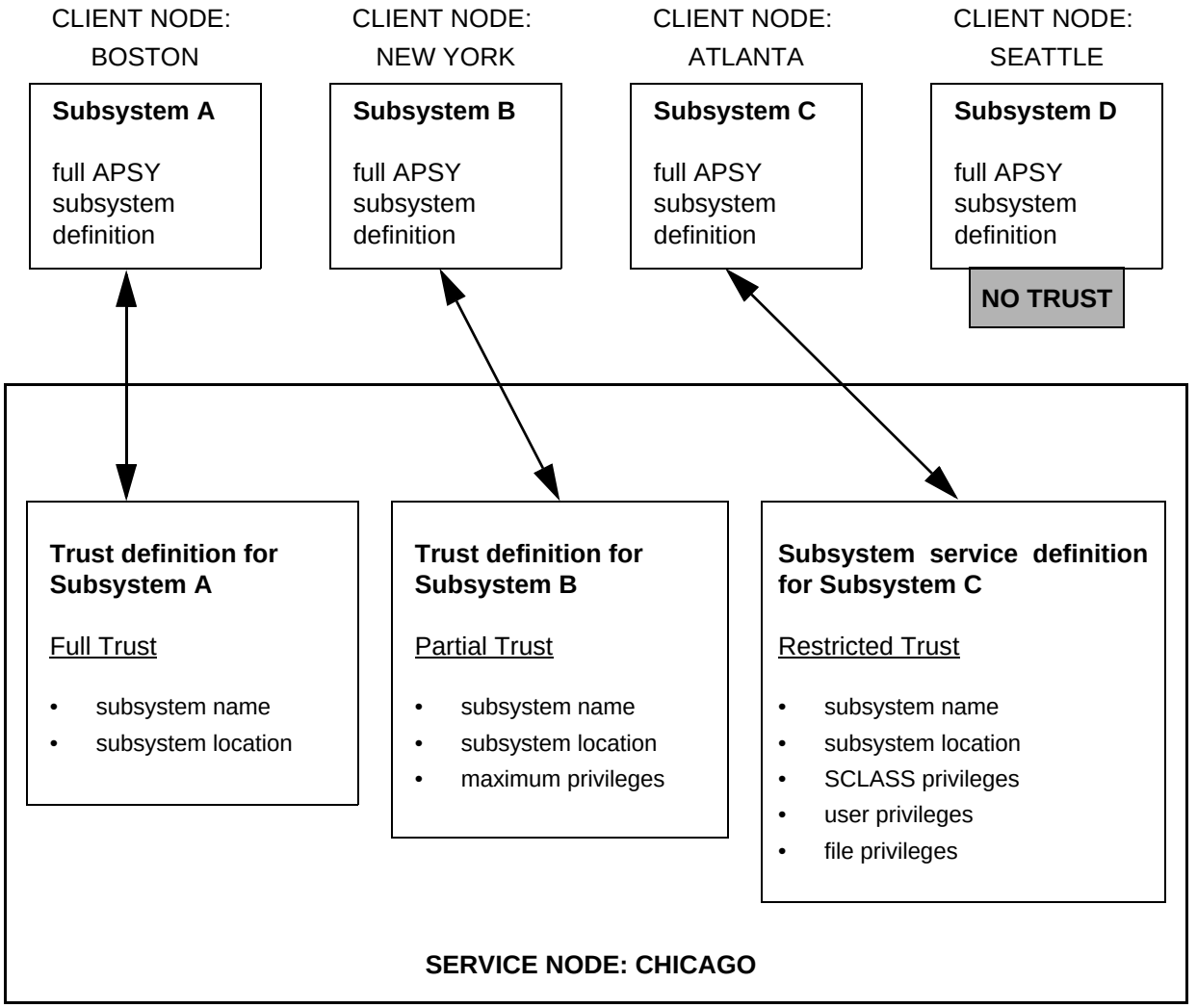
Where to create definitions

To create a definition that has...	Make entries on...
Full trust *or* Partial trust	Subsystem Trust screen
Restricted trust	Subsystem Activity screen Subsystem File Use screen Operational Parameters screen Subsystem Classes screen Subsystem Class Users screen

For details on adding, modifying, suspending, and deleting trust definitions, see “Managing the trust definition” on page 97.

Store trust definitions associated with remote users in the CCASYS file.

Figure 6-1. Subsystems and trust levels



Subsystem command processing

This section describes aspects of START and STOP SUBSYSTEM processing that are unique to distributed applications.

The TEST SUBSYSTEM command works the same way with distributed and nondistributed applications.

START SUBSYSTEM command processing

When a subsystem defined to support remote access is started, a service subsystem of the same name is started when the first file on the node is opened during automatic open processing. With the START request, each server node receives all necessary subsystem definition information, such as a location ID to uniquely identify it within the network.

If all remote file references on the client node are to manual members, the service subsystem is not started until the first manual member is opened. If any remote files need to be opened automatically on the server node, then the following message is written to the server node audit trail for each service subsystem that is successfully started:

```
M204.2330: SUBSYSTEM subsysname FROM location STARTED
```

CREATE GROUP command must precede the START command

If a CREATE GROUP command is required for an automatic and mandatory group, the command must be issued before starting the subsystem. This is necessary so that all remote nodes containing group members can be included in START processing. Group definitions must be local, but files in a group can be remote.

File privileges

User file privileges for remote files are partially determined by the SCLASSes and PRIVDEF settings defined on the server node. Privileges defined on the server node serve as the *maximum* privileges a client user can request. This mechanism gives the service subsystem administrator control over maximum privileges to ensure file security. If a client user requests more than the maximum privileges defined on the server node, then the file open fails.

Procedure dictionary allocation

During START processing, an in-core procedure dictionary is allocated and built on the client and on each server node. The size of the client procedure dictionary is used for all server node dictionaries. There is no procedure file on the remote node, but the procedure dictionary is allocated anyway to keep track of saved compilations.

File and group definitions

File synonyms, including those used indirectly through a group, and permanent group definitions are locked in share mode once a subsystem is started. This prevents the definitions from changing after a subsystem has been successfully started at a remote node. If such a change is needed, the subsystem must be stopped and restarted so that all the internal structures can be rebuilt and retransmitted.

Automatic and optional members

During START processing, APSY tries to open subsystem automatic members. If an automatic mandatory member cannot be opened, START processing is terminated. If an automatic optional member cannot be opened, the member is marked as disabled and START processing continues.

STOP SUBSYSTEM command processing

The STOP SUBSYSTEM command stops both client and service subsystems.

When a STOP command specifies a subsystem that uses remote files, the following actions are taken:

- Subsystem definition on the client node is examined to determine whether the user has STOP command privileges.
- If the user does not already have a conversation, a conversation is established with each remote node on which the subsystem has been started.
- If there are no remaining users in the subsystem, the service subsystems are stopped.
- If there are remaining users in the subsystem, the service subsystems are placed in a drain state and the usual message is produced on the client node.
- If there is a communication failure with any node accessed by the client subsystem, the client subsystem is stopped or placed in a drain state. The following error message is produced for each failure:

  M204.*nnnn*: SUBSYSTEM *subsysname* COULD NOT BE STOPPED AT *location*.

- If the subsystem is not active on any of the nodes accessed by the client subsystem, the following error message is produced for each node:

  M204.*nnnn*: SUBSYSTEM *subsysname* NOT ACTIVE AT *location*

- If the service subsystem is placed in a drain state, then it is stopped when the last user leaves the client subsystem. This happens either through

normal exit conditions or through thread timeouts, which can occur depending on the setting of the TIMEOUT parameter.

STOP SUBSYSTEM stops or places in a drain state the local subsystem and all service subsystems with the same name on their nodes.

The following syntax is available for stopping only the service subsystems associated with a specific client node(s):

```
STOP SUBSYSTEM subsysname FROM location
```

where:

- *subsysname* is the name of the subsystem being stopped.
- *location* is the value of the CCAIN LOCATION parameter of the client node that activated the subsystem (which is also the value of the CLNT in the LOGWHO command output), or is the symbolic name specified by the process definition.

Stopping a subsystem from a remote node where a mandatory file resides causes the node to become disabled to the subsystem.

When client users try to log in to the service subsystem after it has been stopped, a remote START SUBSYSTEM call is sent if there are no mandatory members on the node. If there are mandatory members, the user is not allowed to enter the subsystem on the client until it is stopped and restarted on the client.

Stopping a subsystem from a remote node requires system manager privileges.

Compiling and running procedures

When a non-precompiled procedure that references remote files is invoked, one or more remote nodes participate in the compilation and evaluation. When a precompiled procedure is invoked, Model 204 loads the procedure on each of the nodes that participated in the original compilation.

When a subsystem member becomes unavailable during evaluation, the appropriate ON unit is activated.

Error messages associated with remote procedure processing while loading a request have the prefix RMT in the global error variable.

Saving compilations

As part of the compilation process, a list of remote nodes referenced in the request is generated with the compiled code. When compilation is complete, the compilation is saved along with the list of nodes. Each remote node referenced in the request is sent a signal to save the compilation.

If for any reason a compilation cannot be saved by a server node, the entire save operation fails.

Loading saved compilations

At the client node, the saved remote node reference list is checked to see which nodes are to load the request. When the request is loaded on the client, a signal is transmitted to each referenced server node to load the compilation.

New and missing nodes

A temporary group can be changed so that a node is *new* (not previously referenced) or *missing* (referenced but no longer available). Table 6-2 shows how new and missing nodes affect recompilations and saves.

Table 6-2. Temporary groups with new and missing nodes

	And there are missing nodes...	**And there are no missing nodes....**
If there are new nodes...	Recompile the procedure, do not save	Recompile and save again
If there are no new nodes...	Just load and evaluate	Just load and evaluate

Recompiling saved requests

Saved requests are always recompiled if a new node is introduced into a temporary group. Recompilation can cause a noticeable delay in response time.

In addition, the following changes in the composition of a subgroup also force recompilation of a request. A *subgroup* is the group of files at a server node referenced as a part of a group.

Recompilation is required when:

- Number of files in the subgroup has increased (for example, if a user's request includes a file that was unavailable to the previous user)
- Maximum number of segments in a subgroup has increased

The need to recompile a request is decided automatically.

SUBSYSMGMT overview

This section shows how to define server and client subsystems using the SUBSYSMGMT interface. The screen examples show definitions for a subsystem called SALES. The client system is in Boston; it accesses remote files in Dallas.

The following changes to the SUBSYSMGMT interface are specific to PQO application subsystems:

- To add, modify, suspend, or delete trust definitions, use the Subsystem Trust screen, available from the Subsystem Management Menu screen.
- Several screens have From fields denoting location of client subsystems. You enter a From value on the Activity screen when defining a service subsystem; the From field is then prefilled on secondary screens.
- There are Location fields on the File Use and Subsystem Classes screens. You enter them to indicate the location of remote files to be used by the client subsystem.
- Fields on the File Use screen indicate automatic and mandatory subsystem members.
- Operational Parameters screen has a field indicating whether a client subsystem can access remote files.

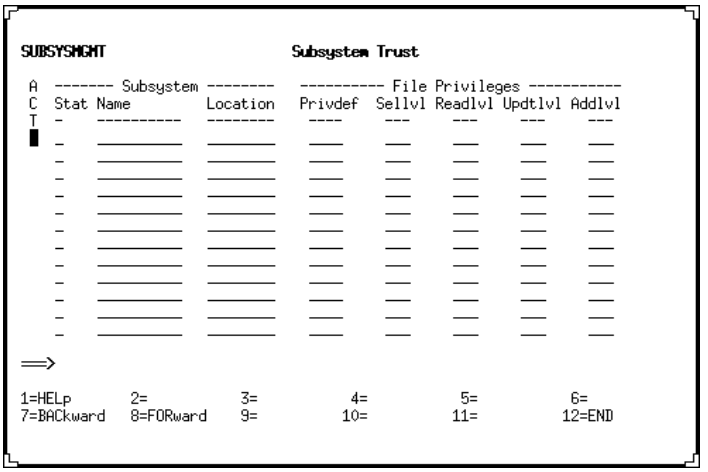
The section "Parameters and login/logout processing" on page 104 explains the effect of various parameters on login and logout processing in a distributed environment.

For complete information on SUBSYSMGMT, see the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/System_requirements_for_Application_Subsystems#Overview_of_the_SUBSYSMGMT_interface

Defining a service subsystem

Subsystem Trust screen



Use the Subsystem Trust screen to view or manage trusted subsystem definitions. The Status field tells you the definition status:

- A = Active
- S = Suspended

Managing the trust definition

To manage trust definitions, you must have system manager privileges at the service node.

Adding a trust definition

1. Enter A, for Add, in the ACT column. The line you choose might be blank or it might already contain information for another definition.
2. Specify the subsystem name and location.
   - If you are assigning full trust, stop here.
   - If you are assigning partial trust, continue to step 3.
3. If necessary, specify maximum file privileges. For information about valid PRIVDEF settings, see the Rocket Model 204 documentation wiki:

   http://m204wiki.rocketsoftware.com/index.php/Establishing_and_maintaining_security.
4. If necessary, specify maximum access levels. If you enter an access level value, you must enter information in each of the access level fields.

5. When you are finished entering trust definitions, press Enter or scroll in either direction to update the file, and then use the END function (PF12 key) to return to the Subsystem Management Facility screen.

Modifying a trust definition

You can modify:

- Status, active or suspended
- File privileges
- Access levels

If you change the status of a trust definition to *suspended*, you temporarily remove trust for the subsystem. The CCASYS trust definition is marked as unavailable, and is not deleted from the CCASYS file. You *cannot* modify the name or location of an existing definition—you must delete the old definition and add a new one with the correct name and location for the subsystem:

1. Find the definition to be modified. If necessary, use the PF7 and PF8 keys to scroll through the list.
2. Enter M, for Modify, in the ACT column.
3. Modify status, file privileges, and/or access levels.
4. When you are finished modifying trust definitions, press Enter to save your work and then use the END function (PF12 key).

Deleting a trust definition

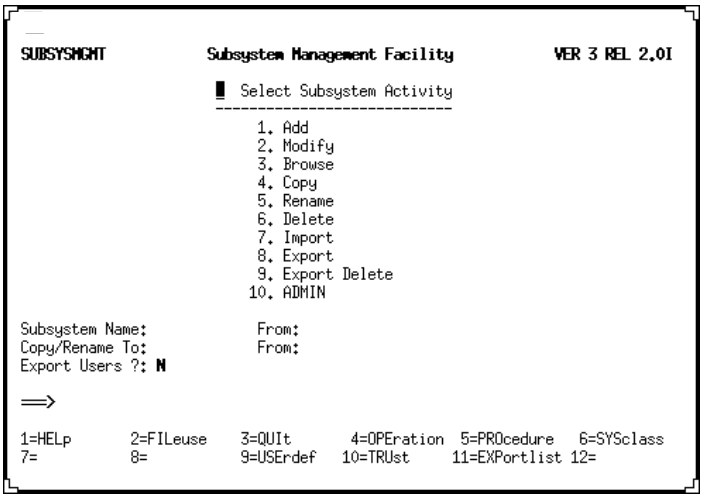
1. Find the definition to be deleted.
2. Enter D, for Delete, in the ACT column. Be sure to save your work before exiting the screen with the END function (PF12 key).

The trust definition is deleted from the CCASYS file, for the subsystem name and location specified. Because the trust definition is used only at subsystem start, deletion does not affect a subsystem that is already started.

Handling errors

If PQO encounters an error, it displays a message and positions the cursor at the line causing the error. All changes to lines preceding the one in error are automatically saved and processed.

Subsystem Activity screen (service definition)



The Subsystem Activity screen is the primary SUBSYSMGMT screen, used to access all SUBSYSMGMT functions.

The From fields specify the location of the client subsystem. If a From field is specified, the Subsystem Name field might not be blank.

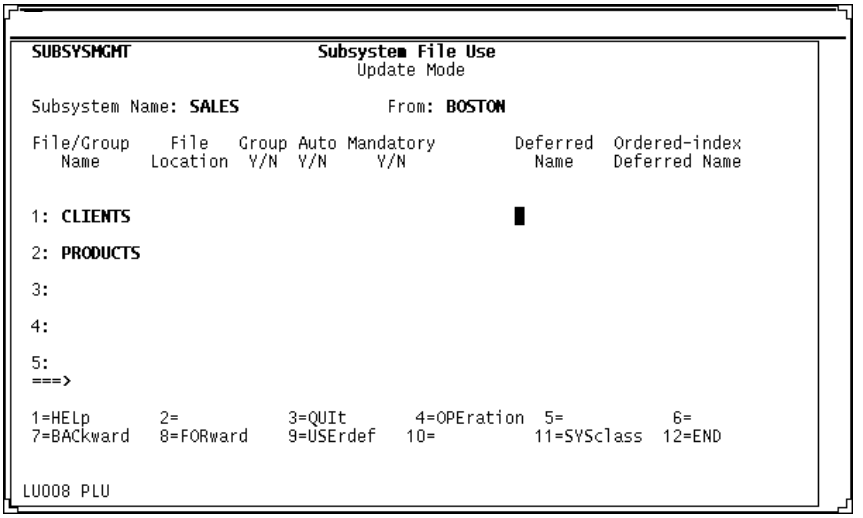
If you are going to the Subsystem Trust screen, the From and Subsystem Name values are used to determine the starting position in the scrollable list of trusted subsystems, which are listed in order by name and location.

The From field is prefilled on all the secondary screens and protected from input.

To define the service subsystem:

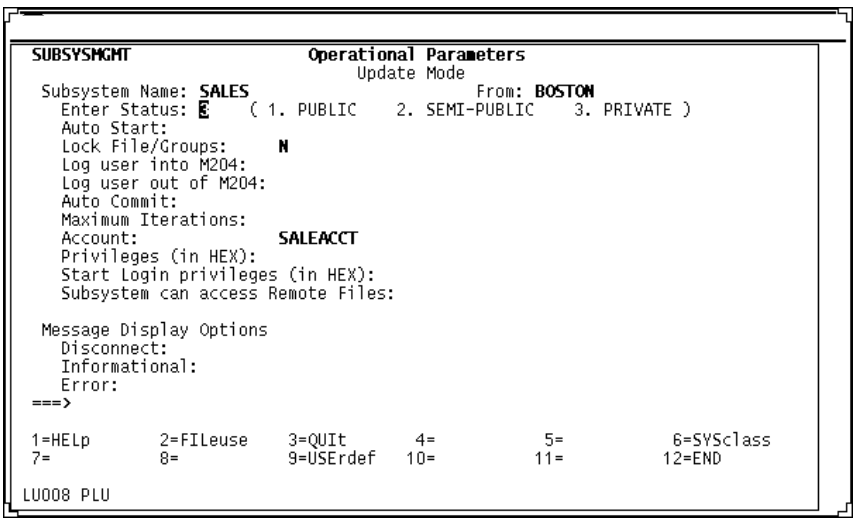
1. Select Add on the Main Menu, specifying the subsystem name and the From location.
2. Select the FILeuse, OPEration, SYSclass, and USErdef functions and fill in the appropriate fields as shown in the following sections.

Subsystem File Use screen (service definition)



The only definitions allowed when defining a service subsystem are file name, deferred name, and ordered index name. All other fields are defined in the client subsystem and protected from input when you are entering service definitions.

Operational Parameters screen (service definition)



Selecting PF4 on the Activity screen brings up the Operational Parameters screen. Use it to specify the operating parameters of the subsystem.

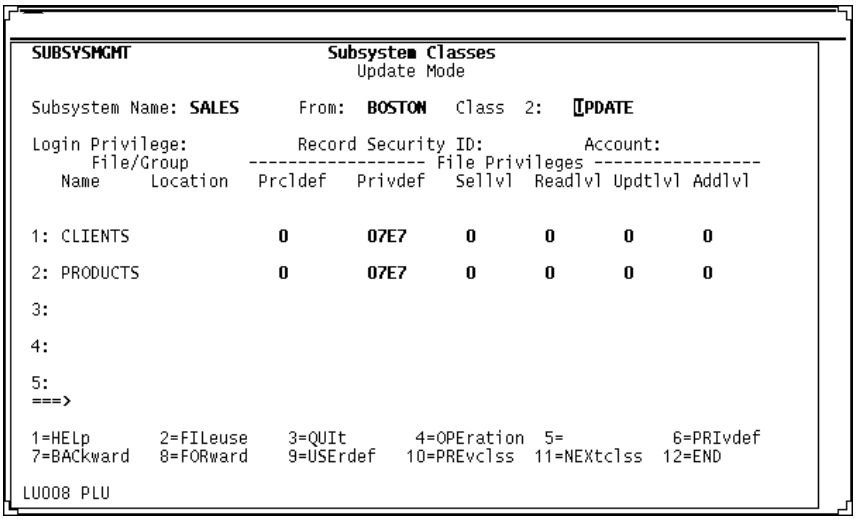
The From field specifies the location of the client node that the definition is being created for. It is prefilled with the From value on the Main Menu and protected from input.

Only the Status, Lock File/Groups, Account, Privileges, and Start Login privileges fields can be modified for the service subsystem. All options that are

not allowed for service subsystem definitions are protected from screen input. These fields are defined in the client subsystem.

Message Display is not modifiable on the server node, because messages are not displayed on the service thread.

Subsystem Classes screen (service definition)

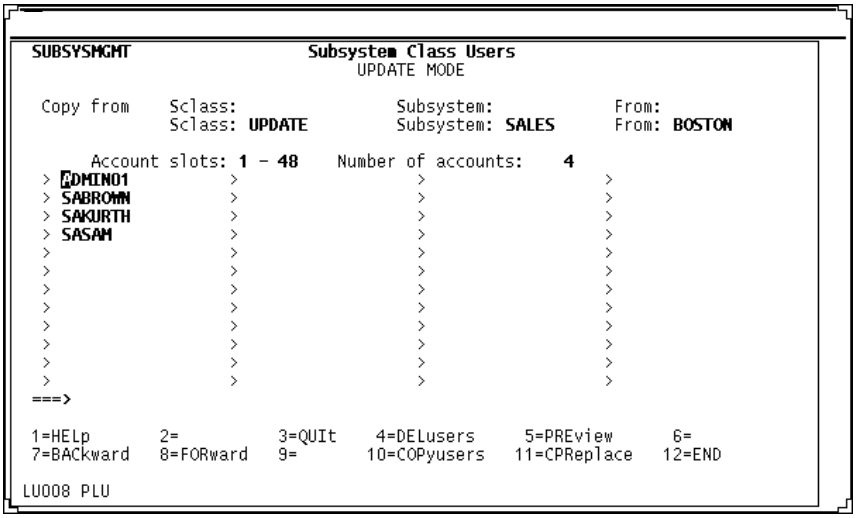


Command and file privileges are defined for each class of subsystem users on the Subsystem Classes screen. Each class of user requires a separate screen. User class privileges defined to the subsystem override settings for OPENCTL and file privileges that reside in the password table.

The Location field is protected from input because there is no remote access on the server node. The files shown here, CLIENTS and PRODUCTS, are local (they reside on the service node).

Parts of this screen pertaining to procedure files are masked, because PQO does not support remote procedure files.

Subsystem Class Users screen (service definition)



User accounts assigned to a specific class are defined on the Subsystem Class Users screen. A user can belong to only one class within a subsystem. If using more than one subsystem, the user can belong to a different class (with different privileges) in each subsystem.

Each user must be in the same class in the client and service subsystems. Otherwise the login fails when the service subsystem login call is sent.

Duplicate account entries result in the display of the class in which the original name resides.

The client and server nodes must have separate CCASYS files. Technical Support recommends that CCASYS files not be shared by different Onlines.

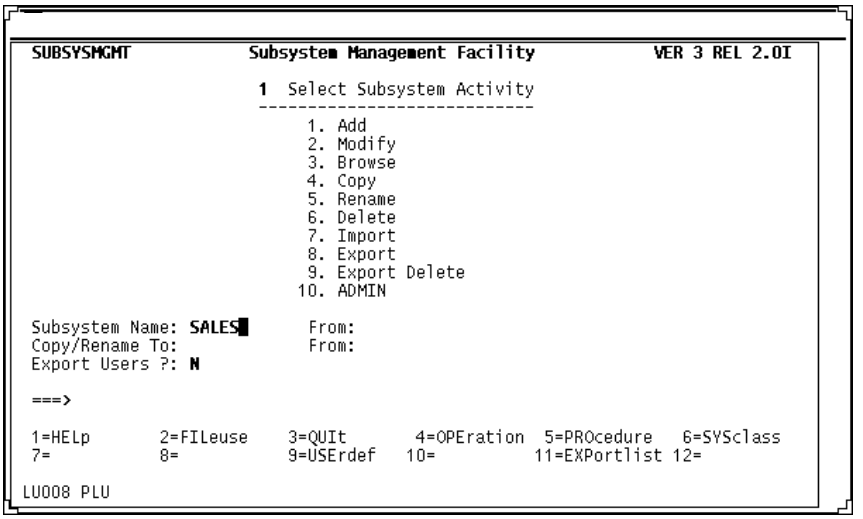
If users have different SCLASSes on the client and server nodes, the following error messages are generated:

```
M204.2314: Error on remote node <indamine>

M204.2309: User not defined in SCLASS <name>
```

Defining a client subsystem

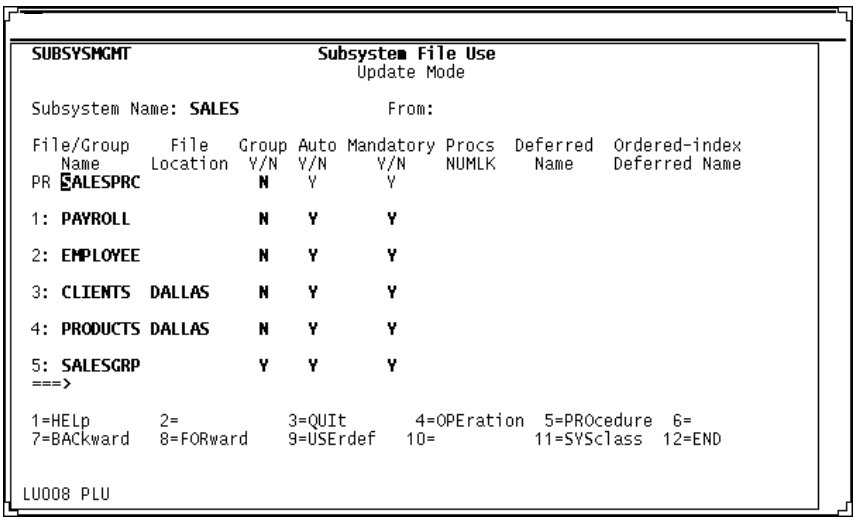
Subsystem Activity screen (client definition)



To define a client subsystem that accesses remote files:

1. Select Add on the Main Menu. Do not specify a From input value for the client.
2. Select the FILeuse, OPEration, SYSclass, and USErdef functions and fill in the appropriate fields as shown in the following sections.

Subsystem File Use screen (client definition)



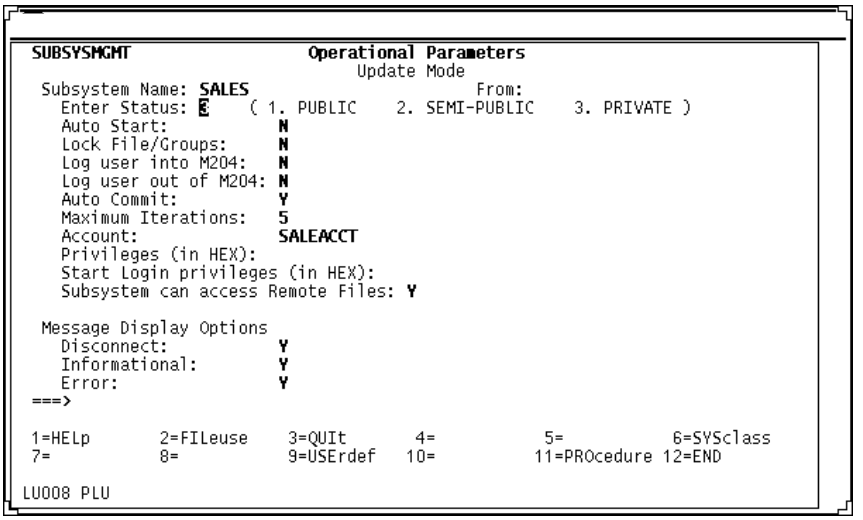
New flags indicate automatic and mandatory members. If the Automatic flag is N, then the Mandatory flag must be N.

The File Location field specifies the location of each file. If this new field is omitted, then the file is assumed to be local. Specify location only for client subsystems.

If you are referring to remote files by means of file synonyms, then it is not necessary to specify remote location.

This example shows the remote files CLIENTS and PRODUCTS at location DALLAS.

Operational Parameters screen (client definition)



When you are defining a client subsystem, you can modify any of the fields on this screen. See the Rocket Model 204 documentation wiki for information on parameters:

http://m204wiki.rocketsoftware.com/index.php/System_requirements_for_Application_Subsystems#Operational_Parameters_screen

The field Subsystem Can Access Remote Files is new. N is the default. Change the value to Y if you want the subsystem to access remote files.

Parameters and login/logout processing

The following subsystem operational parameters affect the way users can log into and out of a subsystem:

- *Automatic start* has no bearing on the service subsystem because the service subsystem is started by Model 204 when the client starts.
- Specifying *Automatic Login* causes the user to be logged out of Model 204 (but not disconnected) and logged back in with a login ID equal to the subsystem name.

Any remote files that the user had open prior to entering the subsystem are closed and all the user's service threads are logged out. The client subsystem establishes new service threads for the user on the appropriate remote nodes. The user's logon privileges, account, and record security key are reset to those specified for the subsystem. The user is then logged in to the service subsystems associated with the client using the new privileges, login ID, account, and record security key.

When the user leaves an Automatic Login subsystem, all local and remote files are closed and the user's service threads are logged out.

- Specifying *Automatic Logout* causes the user to be logged out of Model 204 and disconnected if the SYSOPT disconnect bit is on. Service threads are logged out if the corresponding users were in a subsystem that accessed remote files.
- If you specify *Automatic Commit*, the client node determines that a commit is required due to execution of a SOUL statement, an end of request, or a return to command level.

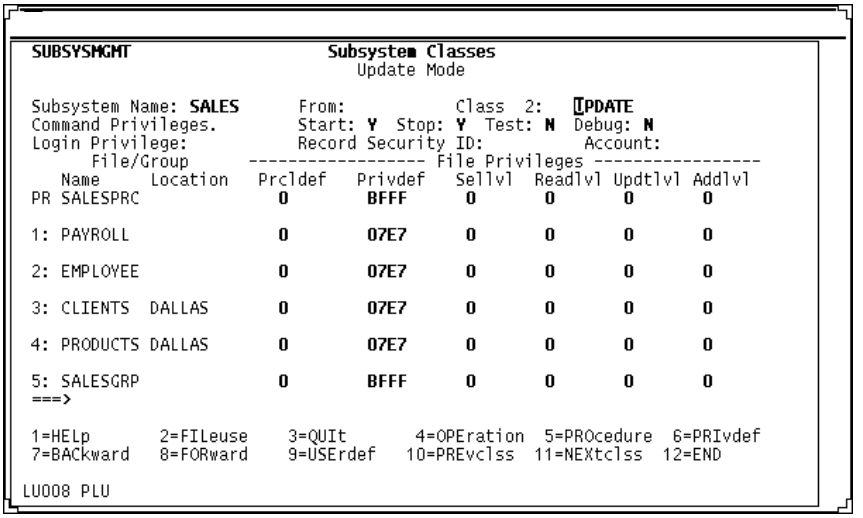
Locked files

If a remote file is opened by a subsystem that locks its files, the file is locked on the client node (only users of this subsystem can use the file on this node). Users of other nodes can also open and use the file.

If the service subsystem locks its files, the file is locked on the service node (only users of the client subsystem can open and use it through the service subsystem).

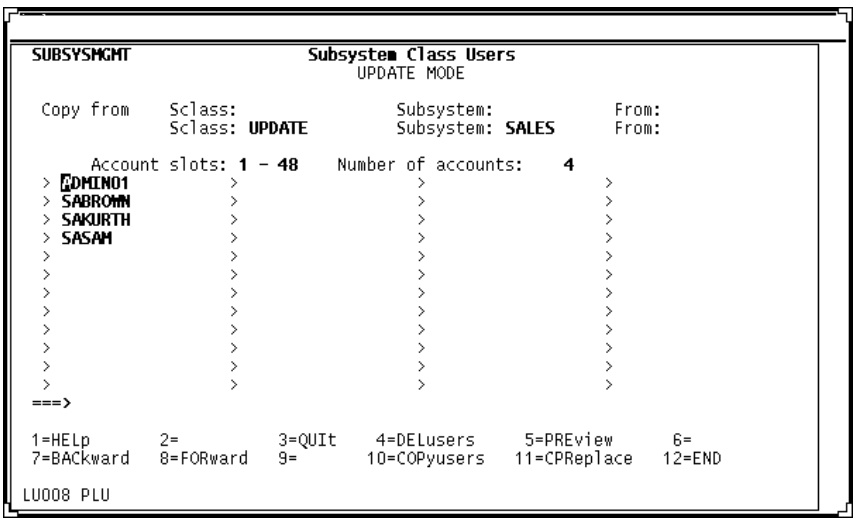
If a subsystem locks its files and an optional member of a subsystem group cannot be locked because it is already open outside of the subsystem, the group can still be opened provided that the number of unavailable group members does not exceed the value of the MAXFAIL parameter.

Subsystem Classes screen (client definition)



When you define a client subsystem, you can modify any of the fields on this screen. In the client definition, the Location field value refers to the node where a file is located. This example shows CLIENTS and PRODUCTS at location DALLAS.

Subsystem Class Users screen (client definition)



Users whoare authorized to access remote files must be included in the same subsystem class definitions for both the client and service subsystems.

The only difference in screen contents on the client side is that the From field is not prefilled.

A

Three-Node Network Example

Overview

This appendix provides an example that demonstrates the use of the network definitions for an expanded Parallel Query Option/204 network. In the expanded network, communication is initiated in either direction.

This example is provided for illustrative purposes only. The options shown apply to the sample application. When setting up your PQO network, refer to Chapter 2 for a detailed description of the definitions and their use.

Reporting application

The example is based on the insurance company reporting application that requires communication between two field offices (Detroit and Dallas) and one main office (Boston). Each office system operates independently, running under its own version of Model 204 in an z/OS environment.

In the example, the network definitions support one office location as a client only, one office location as a server only, and one office as both a client and a server. The main office produces a policy report based on information from data files in the field offices.

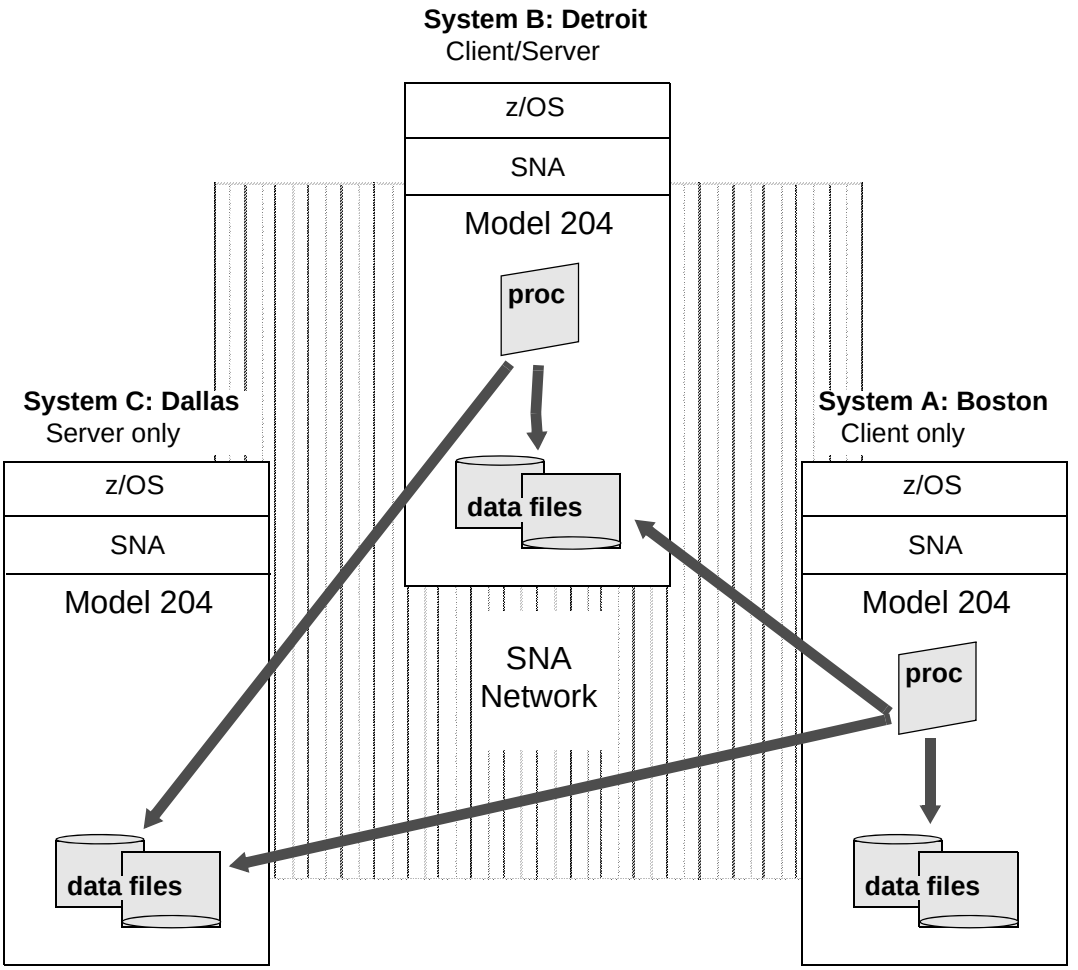
The SOUL procedure that produces the policy report uses locally stored customer data and accesses the remote files in Detroit and Dallas containing vehicle data.

The Detroit office can produce a regional policy report using data files from both Detroit and Dallas.

Figure A-1 on page 108 shows the physical configuration of the expanded network.

Figure A-2 on page 109 shows the interrelationship of the network entities defined for the expanded network.

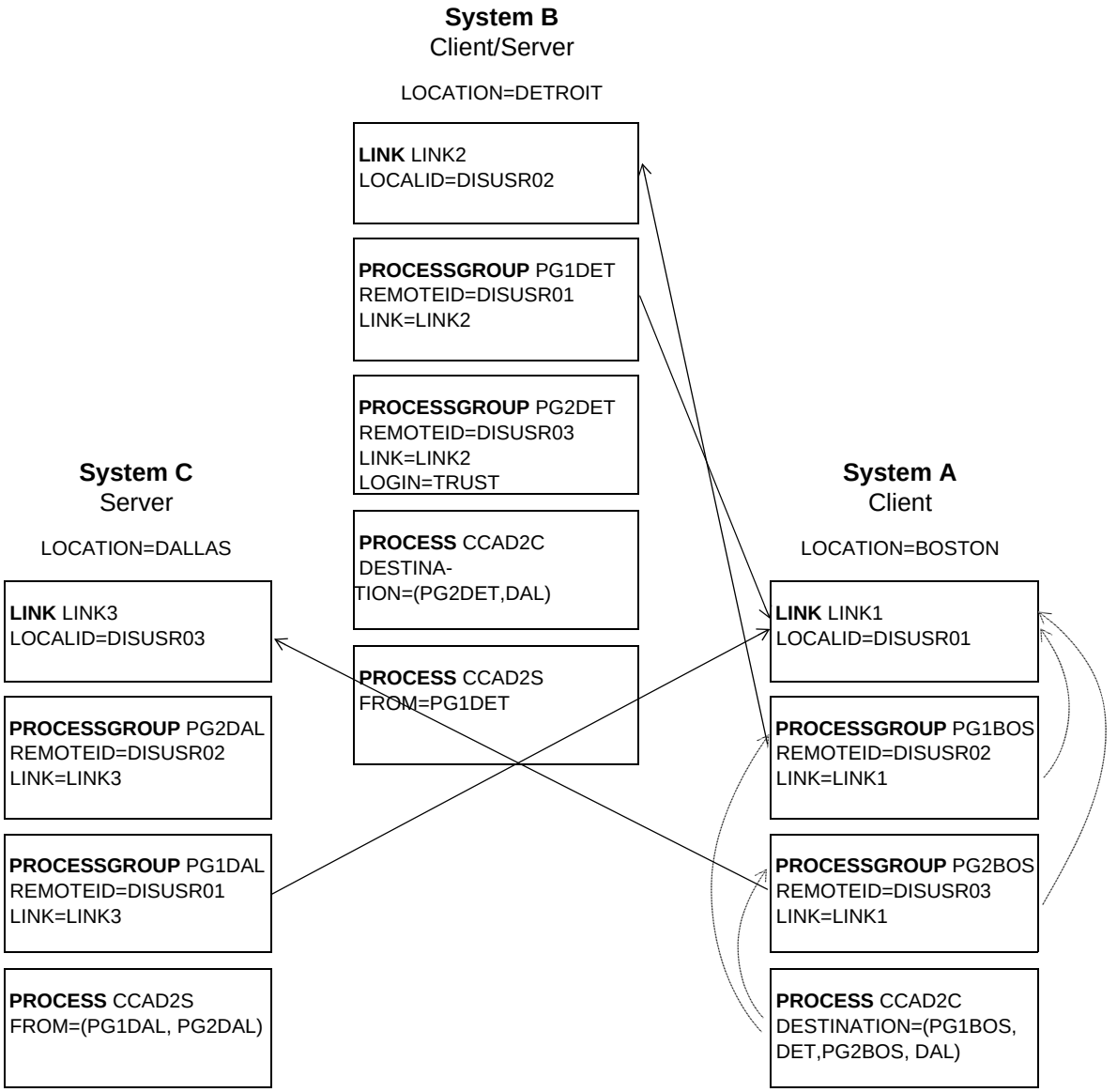
Figure A-1. Physical configuration of expanded network



Network characteristics:

- Three nodes: one client-only, one server-only, one client/server
- Three two-way paths, when links are activated
- Communication initiated by client

Figure A-2. Network entity relationships for System A



Defining the network

The steps required to define the expanded PQO network are listed in the checklist in Table A-1. A detailed description of each step is provided in the sections following the checklist.

Table A-1. Defining the PQO network

Step	Definition	Page
Define the network to Model 204:		
1. Define System A's link.	LINK	111
2. Define System A's processgroups.	PROCESSGROUP	111
3. Define System A's client process.	PROCESS	113
4. Define System B's link.	LINK	113
5. Define System B's processgroups.	PROCESSGROUP	114
6. Define System B's server process.	PROCESS	115
7. Define System B's client process.	PROCESS	116
8. Define System C's link.	LINK	116
9. Define System C's processgroups.	PROCESSGROUP	114
10.Define System C's server process.	PROCESS	118
Define the Online environment:		
11.Set System A's Model 204 runtime and user parameters.	LOCATION NRMTFILE NRMTLOCS NSUBTKS	118
12.Set System B's Model 204 runtime and user parameters.	LOCATION NRMTFILE NRMTLOCS NSUBTKS IODEV=51	119
13.Set System C's Model 204 runtime and user parameters.	LOCATION NRMTLOCS NRMTFILE NSUBTKS IODEV=51	120
Define the network to SNA Communications Server:		
14.Define System A's SNA Communications Server application.	APPL	121
15.Define System B's SNA Communications Server application.	APPL	122
16.Define System C's SNA Communications Server application.	APPL	122

Note: In an actual implementation, each site performs the steps appropriate for each set of definitions. In this section, to show the interrelationships between

the systems, each set of definitions is presented for each system, first for System A (Boston), then for System B (Detroit), and then for System C (Dallas).

The locations (Boston, Detroit, Dallas) are the values specified in the CCAIN LOCATION parameter in the System A, B, and C Onlines, respectively.

The sample CCAIN input streams for the systems are provided in “Sample CCAIN input streams” on page 122.

Specifying DEFINE commands for System A

Defining the link for System A

The first step is to set up the DEFINE LINK command on System A:

```
DEFINE  LINK LINK1 WITH
        SCOPE=SYSTEM
        LOCALID=DISUSR01
        TRANSPORT=VTAM
        PROTOCOL=LU62
        INBUFSIZE=2048
        SESSIONS=6
```

This definition provides the following specifications:

LINK1	Identifies LINK1 as System A’s connection to the SNA network.
LOCALID	The identity of the Boston office’s Model 204 system as an application node in the SNA network is DISUSR01. The LOCALID value matches the name specified in the APPL statement label (see page 121).
TRANSPORT and *PROTOCOL*	The transport type SNA Communications Server and the protocol LU62 are used for communications.
INBUFSIZE	The size of the “receive any” buffer for processing initial inbound conversations requests is 2048 bytes. This matches the mode table RUSIZES value. For more information about SNA Communications Server mode tables, see page 26.
SESSIONS	A maximum of six sessions can be activated concurrently for this link. Note that PARSESS=YES is specified in the APPL statement for System A (page 121) to support concurrent sessions.

Defining the processgroups for System A

The next step is to set up two processgroup definitions for System A: for conversations with Detroit and for conversations with Dallas.

Defining the processgroup for conversations with Detroit

```
DEFINE  PROCESSGROUP PG1BOS WITH
        SCOPE=SYSTEM
        LINK=LINK1
        OUTLIMIT=5
        INLIMIT=0
        REMOTEID=DISUSR02
        LOGIN=TRUST
```

This definition provides the following specifications:

PROCESSGROUP, LINK, and *REMOTEID*	The processgroup PG1BOS provides for conversations with System B at the Detroit office, known as DISUSR02, using the LINK1 connection. Notice the following: • The LINK value matches the DEFINE LINK name specified in the link definition on page 111. • The REMOTEID value matches the LOCALID value specified in System B's DEFINE LINK command, on page 113.
OUTLIMIT	PG1BOS allows a maximum of five concurrent outbound conversations with Detroit (DISUSR02) for the local client process.
INLIMIT	System A is a client only. No inbound conversations are serviced. System A's Online parameters (see page 118) include no IODEV=51 statements.
LOGIN	For the local client process, the current user ID is shipped in an outbound conversation request. *TRUST*, which is required, specifies that no password is required when the remote server system logs the user in. *TRUST* must also be the LOGIN value specified in System B's DEFINE PROCESSGROUP command on page 115.

Defining the processgroup for conversations with Dallas

```
DEFINE  PROCESSGROUP PG2BOS WITH
        SCOPE=SYSTEM
        LINK=LINK1
        OUTLIMIT=5
        INLIMIT=0
        REMOTEID=DISUSR03
        LOGIN=TRUST
```

The following specifications in this definition require comment:

PROCESSGROUP, LINK, and *REMOTEID*	PG2BOS processes converse with System C at the Dallas office, known as DISUSR03, using the LINK1 connection. Notice the following: • Both Boston processgroups use the same link. • The REMOTEID value matches the LOCALID value specified in System C's DEFINE LINK command (see page 116).

Defining the client process for System A

The next step for defining network entities on System A is to set up the DEFINE PROCESS command for the local client process:

```
DEFINE  PROCESS CCAD2C WITH
        SCOPE=SYSTEM
        DESTINATION=(PG1BOS,DET,PG2BOS,DAL)
```

This definition provides the following specifications:

PROCESS and *DESTINATION*	The client process CCAD2C belongs to the local PG1BOS processgroup, which directs conversations to the Detroit office. The DET symbolic name is a reference to the remote system specified by the REMOTEID parameter in the PG1BOS definition. System A client remote file specifications must specify AT DET.

Specifying DEFINE commands for System B

Defining the link for System B

The first step for defining network entities on System B is to set up the DEFINE LINK command:

```
DEFINE  LINK LINK2 WITH
        SCOPE=SYSTEM
        LOCALID=DISUSR02
        TRANSPORT=VTAM
        PROTOCOL=LU62
        INBUFSIZE=2048
        SESSIONS=12
```

This definition provides the following specifications:

LINK2	Identifies LINK2 as System B's connection to the SNA network.

LOCALID	The identity of the Detroit office's Model 204 system as an application node in the SNA network is DISUSR02. The LOCALID value matches the name specified in the APPL statement label (see page 122).
TRANSPORT and *PROTOCOL*	The SNA Communications Server transport type and the LU 6.2 protocol are used for communications.
INBUFSIZE	The size of the "receive any" buffer for processing initial inbound conversations requests is 2048 bytes. This matches the mode table RUSIZES value. For more information about SNA Communications Server mode tables, see page 26.
SESSIONS	A maximum of 12 sessions can be activated concurrently for this link. In the APPL statement on page 122, note that PARSESS=YES is specified to support concurrent sessions.

Defining the processgroups for System B

The next step is to set up two processgroup definitions for System B: for conversations with Boston and for conversations with Dallas.

Defining the processgroup for conversations with Boston

```
DEFINE  PROCESSGROUP PG1DET WITH
        SCOPE=SYSTEM
        LINK=LINK2
        OUTLIMIT=0
        INLIMIT=5
        REMOTEID=DISUSR01
        LOGIN=TRUST
```

This definition provides the following specifications:

PROCESSGROUP, LINK, and *REMOTEID*	The processgroup PG1DET provides for conversations with System A at the Boston office, known as DISUSR01, using the LINK2 connection. Notice the following: • The LINK value matches the DEFINE LINK name specified on page 113. • The REMOTEID value matches the LOCALID name specified in System A's DEFINE LINK command on page 111.
OUTLIMIT	PG1DET provides service to System A. No outbound conversations are defined for the local client process.
INLIMIT	PG1DET allows a maximum of five concurrent inbound conversations (from Boston) for the local server process. In System B's Online parameters on page 120, note that NOTERM specifies five threads.

LOGIN	TRUST specifies that the local server process logs in an incoming client user without a password and assigns privileges from the CCASTAT password table. The SYSOPT parameter is set to include a value of X‘10’.

Defining the processgroup for conversations with Dallas

```
DEFINE  PROCESSGROUP PG2DET WITH
        SCOPE=SYSTEM
        LINK=LINK2
        OUTLIMIT=5
        INLIMIT=0
        REMOTEID=DISUSR03
        LOGIN=TRUST
```

The following specifications in this definition require comment:

PROCESSGROUP, LINK, and *REMOTEID*	PG2DET processes converse with System C at the Dallas office, known as DISUSR03, using the LINK2 connection. Notice the following: • Both Detroit processgroups use the same link. • The REMOTEID value matches the LOCALID value specified in System C’s DEFINE LINK command (see page 116).
LOGIN	For the local client process, the current user ID is shipped in an outbound conversation request. TRUST specifies that no password is required when the remote server system logs the user in. TRUST must also be the LOGIN value specified in System C’s DEFINE PROCESSGROUP command (see page 117).

Defining the processes for System B

The last step for defining network entities on System B is to set up the DEFINE PROCESS commands. System B is a server for System A and a client for System C. Two process definitions are required.

Defining the server process

```
DEFINE  PROCESS CCAD2S WITH
        SCOPE=SYSTEM
        FROM=PG1DET
```

This definition provides the following specifications:

PROCESS and *FROM*	The server process CCAD2S belongs to the local PG1DET processgroup, which associates the process with internal resource limits such as a maximum of five concurrent inbound conversations when responding to client requests from System A.

Defining the client process

```
DEFINE  PROCESS CCAD2C WITH
        SCOPE=SYSTEM
        DESTINATION=(PG2DET,DAL)
```

This definition provides the following specifications:

PROCESS and *DESTINATION*	The client process CCAD2C belongs to the local PG2DET processgroup, which directs the conversation to the Dallas (System C) office. The DAL symbolic name is a reference to the remote system specified by the REMOTEID parameter in the PG2DET definition. System B client remote file specifications must specify AT DAL.

Specifying DEFINE commands for System C

Defining the link for System C

The first step for defining network entities on System C is to set up the DEFINE LINK command:

```
DEFINE  LINK LINK3 WITH
        SCOPE=SYSTEM
        LOCALID=DISUSR03
        TRANSPORT=VTAM
        PROTOCOL=LU62
        INBUFSIZE=2048
        SESSIONS=12
```

This definition provides the following specifications:

LINK3	Identifies LINK3 as System C's connection to the SNA network.
LOCALID	The identity of the Dallas office (System C) as an application node in the SNA network is DISUSR03. The LOCALID value matches the name specified in the APPL statement label (see page 122).
TRANSPORT and *PROTOCOL*	The SNA Communications Server transport type and the LU 6.2 protocol are to be used for communications.

INBUFSIZE	The size of the “receive any” buffer for processing initial inbound conversations requests is 2048 bytes. This matches the mode table RUSIZES value. For more information about SNA Communications Server mode tables, see page 26.
SESSIONS	A maximum of 12 sessions can be activated concurrently for this link. Note that PARSESS=YES is specified in the APPL statement on page 122 to support concurrent sessions.

Defining the processgroups for System C

The next step is to set up two processgroup definitions for System C: for conversations with Boston and for conversations with Detroit.

Defining the processgroup for conversations with Boston

```
DEFINE  PROCESSGROUP PG1DAL WITH
        SCOPE=SYSTEM
        LINK=LINK3
        OUTLIMIT=0
        INLIMIT=5
        REMOTEID=DISUSR01
        LOGIN=TRUST
```

This definition provides the following specifications:

PROCESSGROUP, LINK, and *REMOTEID*	The processgroup PG1DAL provides for conversations with System A at Boston, known as DISUSR01, using the LINK3 connection. Notice the following: • The LINK value matches the DEFINE LINK name specified on page 116. • The REMOTEID value matches the LOCALID name specified in System A’s DEFINE LINK command (see page 111).
OUTLIMIT	PG1DAL provides service to System A. No outbound conversations are defined for the local client process.
INLIMIT	PG1DAL allows a maximum of five concurrent inbound conversations (from Boston) for the local server process. Note that NOTERM in System C’s Online parameters on page 120 specifies ten threads: five for PG2DAL and five for PG1DAL.
LOGIN	TRUST specifies that the local server process logs in an incoming client user without a password and assigns privileges from the CCASTAT password table. The SYSOPT parameter is set to include a value of X‘10’.

Defining the processgroup for conversations with Detroit

```
DEFINE  PROCESSGROUP PG2DAL WITH
        SCOPE=SYSTEM
        LINK=LINK3
        OUTLIMIT=0
        INLIMIT=5
        REMOTEID=DISUSR02
        LOGIN=TRUST
```

The following specifications in this definition require comment:

PROCESSGROUP, LINK, and *REMOTEID*	PG2DAL processes converse with System B at the Detroit office, known as DISUSR02, using the LINK3 connection. Notice the following: • The LINK value matches the DEFINE LINK name specified on page 116. • The REMOTEID value matches the LOCALID name specified in System B's DEFINE LINK command (see page 113).
OUTLIMIT	PG2DAL provides service to System B. No outbound conversations are defined for the local client process.
INLIMIT	PG2DAL allows a maximum of five concurrent inbound conversations (from Detroit) for the local server process. Note that NOTERM in System C's Online parameters on page 121 specifies ten threads: five for PG2DAL and five for PG1DAL.

Defining the server process for System C

The last step for defining network entities on System C is to set up the DEFINE PROCESS command for the local server process:

```
DEFINE  PROCESS CCAD2S WITH
        SCOPE=SYSTEM
        FROM=(PG1DAL,PG2DAL)
```

This definition provides the following specifications:

PROCESS and *FROM*	The server process CCAD2S belongs to both local processgroups.

Specifying the Online environment

Defining System A's Model 204 Online environment

To define the Model 204 Online environment to support the PQO network in System A, set the runtime and user parameters.

1. Enter the User 0 runtime parameters in System A's CCAIN, as follows:

```
LOCATION=BOSTON
```

BOSTON specifies the unique symbolic name of System A's Model 204 Online for the PQO network.

```
NRMTFILE=4
```

NRMTFILE=4 specifies that a maximum of four remote file save areas are allocated by Model 204 for System A's Online for the local client process. No more than four remote Model 204 files can be open at the same time by users on the local system.

```
NRMTLOCS=2
```

NRMTLOCS=2 specifies the number of remote nodes that can be accessed by users on the client Online.

```
NSUBTKS=3
```

NSUBTKS=3 specifies the number of pseudo-subtasks for Model 204. Two additional subtasks are added for the single PQO link that is used for the client process running under z/OS in System A. Two added to the original value of one makes the total equal to three.

2. No additional IODEV parameter lines are required for System A, because it is exclusively a client. Only service threads require an additional IODEV=51 statement.

Defining System B's Model 204 Online environment

Define the Model 204 Online environment to support the PQO network in System B. Set the runtime and user parameters:

1. Enter the User 0 runtime parameter in System B's CCAIN, as follows:

```
LOCATION=DETROIT
```

DETROIT specifies the unique symbolic name of System B's Model 204 Online for the PQO network.

```
NRMTFILE=2
```

NRMTFILE=2 specifies that a maximum of two remote file save areas are allocated by Model 204 for System B's Online for the local client process. No more than two remote Model 204 files can be open at the same time by users on the local system.

```
NRMTLOCS=1
```

NRMTLOCS=1 specifies the number of remote nodes that can be accessed by users on the client Online.

```
NSUBTKS=3
```

NSUBTKS=3 specifies the number of pseudo-subtasks for Model 204. Two subtasks are added for the PQO link running under z/OS. Two added to the original value of one makes the total equal to three.

2. Enter the IODEV statements in System B's CCAIN to support the local server process. Enter the user parameter lines after the User 0 parameters, as follows:

```
IODEV=51,NOTERM=5
IODEV=51
IODEV=51
IODEV=51
IODEV=51
```

IODEV=51 specifies a PQO service thread.

NOTERM=5 specifies that five threads are allocated for PQO inbound conversations in System B's Online, indicating that five server applications can run concurrently.

Defining System C's Model 204 Online environment

Define the Model 204 Online environment to support the PQO network in System C. Set the runtime and user parameters.

1. Enter the User 0 runtime parameter in System C's CCAIN, as follows:

```
LOCATION=DALLAS
```

DALLAS specifies the unique symbolic name of System C's Model 204 Online for the PQO network.

```
NRMTFILE=0
```

NRMTFILE=0 (the default) specifies that no remote file save areas are allocated by Model 204 for System C's Online. System C is exclusively a server and is not accessing any remote files.

```
NRMTLOCS=0
```

NRMTLOCS=0 (the default) specifies that no remote PQO nodes are accessed by System C. System C is exclusively a server and is not accessing any remote files.

```
NSUBTKS=3
```

NSUBTKS=3 specifies the number of pseudo-subtasks for Model 204. Two subtasks are added for the PQO link running under z/OS. Two added to the original value of one makes the total equal to three.

2. Enter the IODEV statements in System C's CCAIN to support the local server process. Enter the user parameter lines after the User 0 parameters, as follows:

```
IODEV=51,NOTERM=10
.
.
.
IODEV=51
```

IODEV=51 specifies a PQO service thread.

NOTERM=10 specifies that ten threads are allocated for PQO inbound conversations in System C's Online, indicating that ten server applications can run concurrently. This number exactly accommodates the maximum of five threads specified by each of System C's clients. This exact accommodation is not required, however.

Specifying the SNA Communications Server environment

Defining the APPL statement for System A

The next step is to define the PQO network to SNA Communications Server. An APPL statement is required for LINK1, which is defined on page 111. The APPL statement is coded in System A's VTAMLST data set to define it as an LU in the network, as follows:

```
DISUSR01      APPL        AUTH=ACQ,
                          PARSESS=YES,
                          MODETAB=LU62,
                          VPACING=5
```

This definition provides the following specifications:

DISUSR01	*S*pecifies the unique name that identifies System A as an LU application node in this SNA network. This name matches the name used for LOCALID in System A's Model 204 DEFINE LINK command.
AUTH=ACQ	*S*pecifies that Model 204 is authorized to acquire sessions with the remote conversation partner LU when initiating or responding to a request for a conversation. This setting authorizes SIMLOGON, the Model 204 SNA Communications Server request.
PARSESS=YES	Specifies SNA Communications Server parallel session support between the System A LU and a partner. In the sample network, this supports concurrent sessions between System A and System B and between System A and System C.
MODETAB=LU62	Points SNA Communications Server to an existing mode table, called LU62, that contains the mode definition used by System A's Horizon facility. An entry in this mode table specifies LU 6.2 session parameters for PQO.

VPACING=5	Specifies five messages for SNA Communications Server pacing. This is the maximum number of chain elements received by the System A LU during a session before an acknowledgment is sent to the partner LU. An average message length is assumed.

Defining the APPL statement for System B

An APPL statement is required for LINK2, which is defined on page 113. The APPL statement is coded in System B's VTAMLST data set to define it as an LU in the network, as follows:

```
DISUSR02    APPL    AUTH=ACQ,
                    PARSESS=YES,
                    MODETAB=LU62,
                    VPACING=5
```

This definition provides the same specifications as for System A except for the APPL label name DISUSR02.

Defining the APPL statement for System C

To complete the SNA Communications Server definition for the sample network, an APPL statement is required for LINK3, which is defined on page 116. The APPL statement is coded in System C's VTAMLST data set to define it as an LU in the network, as follows:

```
DISUSR03        APPL        AUTH=ACQ,
                            PARSESS=YES,
                            MODETAB=LU62,
                            VPACING=5
```

This definition provides the same specifications as for Systems A and B except for the APPL label name DISUSR03.

Sample CCAIN input streams

Excerpts from the sample CCAIN input streams for Systems A, B, and C are provided in this section. These excerpts show the Model 204 Online environment and network definitions for PQO.

System A input stream (z/OS environment)

```
//CCAIN DD  *
*       User zero parameters
.
.
.
      NRMTFILE=4,
```

```
      NRMTLOCS=2,
      NSUBTKS=3,
      LOCATION=BOSTON

*       User parameter lines

*     Single link, multiple process network definition

DEFINE LINK LINK1 WITH SCOPE=SYSTEM LOCALID=DISUSR01     -
    TRANSPORT=VTAM PROTOCOL=LU62 INBUFSIZE=2048          -
    SESSIONS=12

DEFINE PROCESSGROUP PG1BOS WITH SCOPE=SYSTEM             -
    LINK=LINK1 OUTLIMIT=5 INLIMIT=0                      -
    REMOTEID=DISUSR02 LOGIN=TRUST

DEFINE PROCESSGROUP PG2BOS WITH SCOPE=SYSTEM             -
    LINK=LINK1 OUTLIMIT=5 INLIMIT=0                      -
    REMOTEID=DISUSR03 LOGIN=TRUST

DEFINE PROCESS CCAD2C WITH SCOPE=SYSTEM                  -
    DESTINATION=(PG1BOS,DET,PG2BOS,DAL)
.
.
.
/*
```

System B input stream (z/OS environment)

```
//CCAIN DD  *
*       User zero parameters
.
.
.
      NRMTFILE=2,
      NRMTLOCS=1,
      NSUBTKS=3,
      LOCATION=DETROIT

*      User parameter lines

IODEV=51,NOTERM=5
IODEV=51
IODEV=51
IODEV=51
IODEV=51

*     Single link, multiple process network definition
```

```
DEFINE LINK LINK2 WITH SCOPE=SYSTEM LOCALID=DISUSR02   -
   TRANSPORT=VTAM PROTOCOL=LU62 INBUFSIZE=2048         -
   SESSIONS=6

DEFINE PROCESSGROUP PG1DET WITH SCOPE=SYSTEM           -
   LINK=LINK2 OUTLIMIT=0 INLIMIT=5                     -
   REMOTEID=DISUSR01 LOGIN=TRUST

DEFINE PROCESSGROUP PG2DET WITH SCOPE=SYSTEM           -
   LINK=LINK2 OUTLIMIT=5 INLIMIT=0                     -
   REMOTEID=DISUSR03 LOGIN=TRUST

DEFINE PROCESS CCAD2S WITH SCOPE=SYSTEM                -
      FROM=PG1DET

DEFINE PROCESS CCAD2C WITH SCOPE=SYSTEM                -
   DESTINATION=(PG2DET,DAL)
.
.
.
/*
```

System C input stream (z/OS environment)

```
//CCAIN DD  *
*       User zero parameters
.
.
.
      NRMTFILE=0,
      NRMTLOCS=0,
      NSUBTKS=3,
      LOCATION=DALLAS

*      User parameter lines

IODEV=51,NOTERM=10
IODEV=51
IODEV=51
IODEV=51
IODEV=51
IODEV=51
IODEV=51
IODEV=51
IODEV=51
IODEV=51
*     Single link, multiple process network definition
```

```
DEFINE LINK LINK3 WITH SCOPE=SYSTEM LOCALID=DISUSR03   -
   TRANSPORT=VTAM PROTOCOL=LU62 INBUFSIZE=2048         -
   SESSIONS=12

DEFINE PROCESSGROUP PG1DAL WITH SCOPE=SYSTEM           -
   LINK=LINK3 OUTLIMIT=0 INLIMIT=5                     -
   REMOTEID=DISUSR01 LOGIN=TRUST

DEFINE PROCESSGROUP PG2DAL WITH SCOPE=SYSTEM           -
   LINK=LINK3 OUTLIMIT=0 INLIMIT=5                     -
   REMOTEID=DISUSR02 LOGIN=TRUST

DEFINE PROCESS CCAD2S WITH SCOPE=SYSTEM                -
      FROM=(PG1DAL,PG2DAL)
.
.
.
/*
```

B

Restricted Commands, $Functions, and DML Statements

Overview

The Model 204 commands and $functions that are not supported in remote context for Parallel Query Option/204 processing are listed in this appendix.

Restricted Model 204 commands

Some of the Model 204 commands that specify a file name, or that operate on a default file or on a file specified in an IN clause, are not supported in remote context for PQO processing.

The following commands cannot be used in reference to a remote file. See Table 5-1 on page 57 for a description of the commands that can be used to reference files in remote context:

```
ASSIGN (specifying a procedure alias)

BROADCAST FILE

CREATE FILE

DEASSIGN
DECREASE
DEFINE FIELD
DELETE PROCEDURE
DESECURE
DESECURE PROCEDURE
```

```
DISPLAY FILE
DISPLAY LIST
DISPLAY PROCEDURE
DUMP

EDIT
ENQCTL

FILELOAD
FLOD

INCLUDE
INCREASE
INITIALIZE

PROCEDURE

REDEFINE FIELD
REGENERATE
RENAME FIELD
RENAME PROCEDURE
REORGANIZE
RESET (file parameters)
RESTORE

SECURE
SECURE PROCEDURE

TABLEB
TABLEC
TRANSFORM

Z
```

Restricted SOUL $functions

The following SOUL $functions cannot be used in reference to a remote file:

```
$LSTPROC
$RDPROC
```

If either of these $functions is used in reference to a remote file, Model 204 displays an error message.

See “Using $functions” on page 79 for a description of the $functions whose functionality or usage is different for PQO.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

Z