Connect:Direct®
for UNIX

User's Guide

Version 3.8

Sterling Commerce
An IBM Company

Connect:Direct for UNIX User's Guide
Version 3.8

First Edition

Sterling Commerce, Inc.

4600 Lakehurst Court Dublin, OH 43016-2000 *
614/793-7000

CDUNXUG611

Contents

Chapter 1

About Connect:Direct for UNIX

Connect:Direct links technologies and moves all types of information between networked systems and computers. It manages high-performance transfers by providing such features as automation, reliability, efficient use of resources, application integration, and ease of use. Connect:Direct offers choices in communications protocols, hardware platforms, and operating systems. It provides the flexibility to move information among mainframe systems, midrange systems, desktop systems, and LAN-based workstations.

Connect:Direct is based on client-server architecture. The Connect:Direct server components interact with the user interfaces (API, CLI, Connect:Direct Browser User Interface, and Sterling Control Center) to enable you to submit, execute, and monitor Connect:Direct statements and commands.

Server Components

Connect:Direct has the following server components:

Process Manager

The Process Manager (PMGR) is the daemon that initializes the Connect:Direct server environment. The PMGR provides the following functions:

- Initializes Connect:Direct
- Accepts connection requests from Connect:Direct client APIs and remote nodes
- Creates Command Manager and Session Manager child Processes to communicate with APIs and remote nodes
- Accepts requests from Command Managers and Session Managers when centralized Connect:Direct functions are required
- Stops Connect:Direct

Note: Any application, including End User Applications (EUA), can run on any computer as long as it can connect to the PMGR.

Command Manager

A Command Manager (CMGR) is created for every API connection that is successfully established. The number of Command Managers that a PMGR can create is system-dependent and limited by the number of file descriptors available for each UNIX Process. The number of file descriptors set up by the UNIX operating system may affect Connect:Direct operation. You must define enough file descriptors to handle the number of concurrent Connect:Direct sessions allowed, which can be as many as 999.

The CMGR provides the following functions:

- Executes commands sent by the API and sends the results back to the API
- Carries out the Connect:Direct authentication procedure, in conjunction with the API, to determine access to Connect:Direct
- Interacts with the PMGR when executing commands

Session Manager

The Session Manager (SMGR) is created and invoked by the PMGR when resources are available and either a Process is ready to run or a remote node requests a connection with a local node. The SMGR provides the following functions:

- Performs the necessary Connect:Direct work
- Acts as a primary node (PNODE) and initiates Process execution
- Acts as a secondary node (SNODE) to participate in a Process initiated by the PNODE

When an SMGR is created to execute a Process submitted to a node, it creates the connection to the remote node. If the SMGR is started by the PMGR to execute local Processes, the SMGR runs each Process on this session until all Processes are completed.

If an SMGR is created because a remote node initiated a connection, the SMGR completes the connection. If the SMGR is started by the PMGR to execute remote Processes, the SMGR executes remote Process steps supplied by the remote SMGR until the remote SMGR completes all of its Processes.

The SMGR depends on the PMGR for Transmission Control Queue (TCQ) services and other centralized services. Refer to the *Transmission Control Queue* on page 1 for an overview of the TCQ.

File Agent

Connect:Direct File Agent is a feature of Connect:Direct, which provides unattended file management. File Agent monitors *watched* directories to detect new files. When File Agent detects a new file, it either submits a default Process or evaluates the file using rules to override the default Process and to determine which Process to submit. You create rules to submit different Processes based on the following properties:

- Specific or partial file names
- File size
- System events

You create the Processes used by File Agent on Connect:Direct; you cannot create them using File Agent.

To achieve optimum performance, configure File Agent to communicate with the Connect:Direct node where it is installed. File Agent can be installed on UNIX, Windows, and z/OS operating systems. For information to help you plan how to implement File Agent, see the *Managing Files with Connect:Direct File Agent* chapter in your Connect:Direct administration guide or getting started guide. The Connect:Direct File Agent Help contains instructions for configuring File Agent.

Connect:Direct Secure+ Option for UNIX

The Connect:Direct Secure+ Option application provides enhanced security for Connect:Direct and is available as a separate component. It uses cryptography to secure data during transmission. You select the security protocol to use with Secure+ Option.

To use Connect:Direct Secure+ Option for communications with remote nodes, you must have node records in the Secure+ Option parameters file (SPNODES) that duplicate the adjacent node records in the Connect:Direct network map. You can populate the Secure+ Option parameters file from entries defined in an existing network map. For more information about creating the Connect:Direct Secure+ Option parameters file (SPNODES) and configuring nodes for Secure+ Option, refer to the *Connect:Direct Secure+ Option for UNIX Implementation Guide*.

User Interfaces

Connect:Direct has the following user interfaces, which enable you to create, submit, and monitor

Applications Programming Interface

The UNIX Applications Programming Interface (API) enables you to write programs that work with Connect:Direct. Four API functions are provided to allow an End User Application (EUA) to perform the following tasks:

- Establish an API connection to the Connect:Direct server
- Terminate an API connection to the Connect:Direct server
- Send a command to Connect:Direct
- Receive responses from commands

Command Line Interface

The Command Line Interface (CLI) enables you to perform the following tasks:

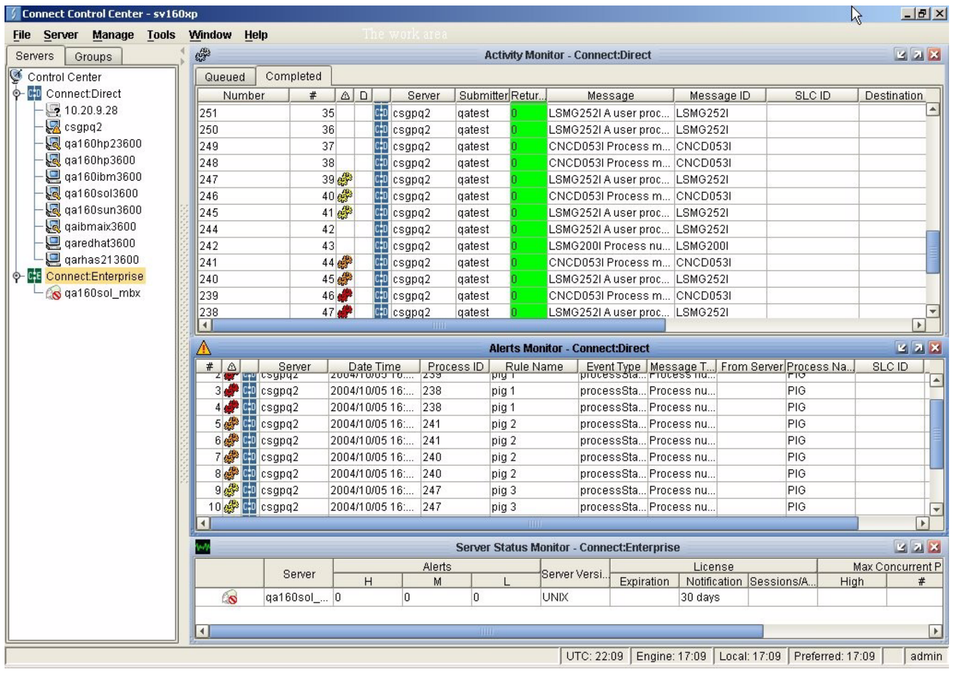
- Issue Connect:Direct commands
- Monitor Processes

The CLI command prompt is **Direct** >. Refer to Chapter 2, *Controlling and Monitoring Processes* in the *Connect:Direct User's Guide* for additional information about the CLI.

Sterling Control Center

Sterling Control Center is a centralized management system that provides operations personnel with continuous enterprise-wide business activity monitoring capabilities for Connect:Direct for z/OS, Connect:Direct for UNIX, Connect:Direct for Windows, Connect:Direct for HP NonStop, Connect:Direct Select, and Connect:Direct OS/400 (iSeries) servers, and Connect:Enterprise for UNIX and Connect:Enterprise for z/OS servers. Sterling Control Center enables you to:

- Manage multiple servers
  - Group individual servers into server groups for a single view of system-wide activity
  - View status and statistics on active or completed processing
  - Suspend, release, stop, and delete Connect:Direct Processes on z/OS, UNIX, Windows, Select, and HP NonStop platforms
  - Stop Connect:Direct servers on z/OS, Windows, HP NonStop, and UNIX platforms.
- Monitor service levels
  - View processing across Connect:Direct for z/OS, UNIX, Select, Windows, HP NonStop, and OS/400 (iSeries) servers, and Connect:Enterprise for UNIX and Connect:Enterprise for z/OS servers within your network and retrieve information about active and completed processing


Connect Control Center - sv160xp
File Server Manage Tools Window Help
The work area
Servers Groups
Control Center
Connect:Direct
10.20.9.28
csgpq2
qa160hp23600
qa160hp3600
qa160ibm3600
qa160sol3600
qa160sun3600
qaibmaix3600
qaredhat3600
qarhas213600
Connect:Enterprise
qa160sol_mbx
Activity Monitor - Connect:Direct
Queued Completed
Alerts Monitor - Connect:Direct
Server Status Monitor - Connect:Enterprise
UTC: 22:09 Engine: 17:09 Local: 17:09 Preferred: 17:09 admin

Number	#	⚠	D		Server	Submitter	Retur...	Message	Message ID	SLC ID	Destination
251	35				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
250	36				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
249	37				csgpq2	qatest	0	CNCD053I Process m...	CNCD053I		
248	38				csgpq2	qatest	0	CNCD053I Process m...	CNCD053I		
247	39				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
246	40				csgpq2	qatest	0	CNCD053I Process m...	CNCD053I		
245	41				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
244	42				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
242	43				csgpq2	qatest	0	LSMG200I Process nu...	LSMG200I		
241	44				csgpq2	qatest	0	CNCD053I Process m...	CNCD053I		
240	45				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		
239	46				csgpq2	qatest	0	CNCD053I Process m...	CNCD053I		
238	47				csgpq2	qatest	0	LSMG252I A user proc...	LSMG252I		

#	⚠		Server	Date Time	Process ID	Rule Name	Event Type	Message T...	From Server	Process Na...	SLC ID
3			csgpq2	2004/10/05 16:...	238	pig 1	processSta...	Process nu...		PIG	
4			csgpq2	2004/10/05 16:...	238	pig 1	processSta...	Process nu...		PIG	
5			csgpq2	2004/10/05 16:...	241	pig 2	processSta...	Process nu...		PIG	
6			csgpq2	2004/10/05 16:...	241	pig 2	processSta...	Process nu...		PIG	
7			csgpq2	2004/10/05 16:...	240	pig 2	processSta...	Process nu...		PIG	
8			csgpq2	2004/10/05 16:...	240	pig 2	processSta...	Process nu...		PIG	
9			csgpq2	2004/10/05 16:...	247	pig 3	processSta...	Process nu...		PIG	
10			csgpq2	2004/10/05 16:...	247	pig 3	processSta...	Process nu...		PIG	

	Server	Alerts			Server Versi...	License			Max Concurrent P	
		H	M	L		Expiration	Notification	Sessions/A...	High	#
	qa160sol_...	0	0	0	UNIX		30 days			

  - Receive notification of data delivery events that occur or do not occur as scheduled
  - Define rules that, based on processing criteria, can generate an alert, send an e-mail notification, generate a Simple Network Management Protocol (SNMP) trap to an Enterprise Management System (EMS), run a system command, or issue a server command

  - Monitor for alerts, such as a server failure or a Process not starting on time
  - Create service level criteria (SLCs) that define processing schedules, monitor Processes, files within Processes, and file transfers for compliance with these schedules, and generate alerts when the schedules are not met
- Analyze key operational metrics
- Create customized reports to document and analyze processing activity based on criteria you define
- Validate the authenticity of a user logging on to Sterling Control Center, using one or more of four authentication methods including password validation, host name identification, Windows domain, and TCP/IP address
- Identify additional servers that may need to be monitored based on communications with a currently monitored server using the Guided Node Discovery feature

Sterling Control Center enhances operational productivity and improves the quality of service by:

- Ensuring that critical processing windows are met
- Reducing impact on downstream processing by verifying that expected processing occurs
- Providing proactive notification for at-risk business processes
- Consolidating information for throughput analysis, capacity planning, post-processing operational or security audits, and workload analysis
- Reducing the risk of errors associated with manual system administration, including eliminating individual server logon to view activity, and the need to separately configure each server for error and exception notifications

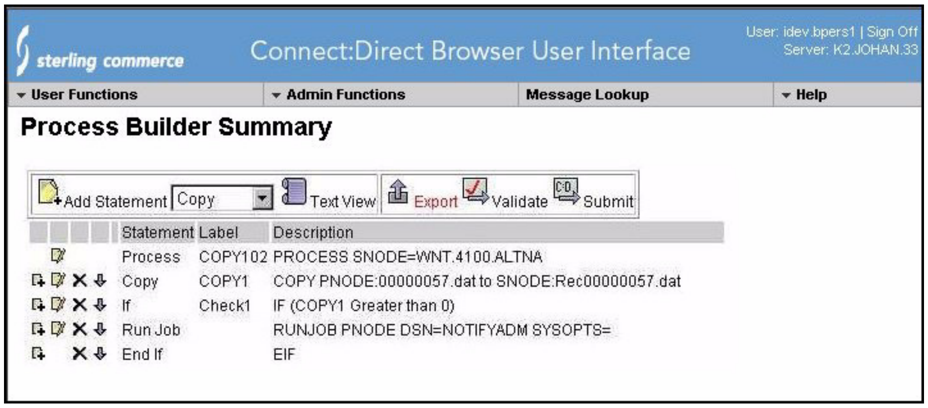
Sterling Control Center is available for purchase as a separate product. Contact your Sterling Commerce representative to learn more about Sterling Control Center.

Connect:Direct Browser User Interface

Connect:Direct Browser User Interface allows you to build, submit, and monitor Connect:Direct Processes from an Internet browser, such as Microsoft Internet Explorer.

You can also perform Connect:Direct system administration tasks, such as viewing and changing the network map or initialization parameters, from Connect:Direct Browser. The specific administration tasks that you can perform depend on the Connect:Direct platform that your browser is signed on to and your security level.

Connect:Direct Browser is distributed on CD-ROM with Connect:Direct for z/OS, Connect:Direct for Windows, Connect:Direct for UNIX, and Connect:Direct for HP NonStop. It can also be downloaded from the Sterling Commerce Web site. Connect:Direct Browser is installed on a Web server and can be accessed by administrators and users through a URL. The following example shows the page used to graphically build a Process:



Process Builder Summary

Statement	Label	Description
Process	COPY102	PROCESS SNODE=WNT.4100.ALTNA
Copy	COPY1	COPY PNODE:00000057.dat to SNODE:Rec00000057.dat
If	Check1	IF (COPY1 Greater than 0)
Run Job		RUNJOB PNODE DSN=NOTIFYADM SYSOPTS=
End If		EIF

To learn more about Connect:Direct Browser, see the documentation on the Connect:Direct Browser CD-ROM or available online from the Sterling Commerce Documentation Library.

Connect:Direct Concepts

This section introduces concepts and definitions to help you understand system operations.

Local and Remote Nodes

Each data transfer involves a local and a remote node. The Connect:Direct server is the local node and the partner node is the remote node. The two servers (local and remote) function together to perform the work. Either Connect:Direct node can initiate the work.

Local and remote node connections are set up in the network map file. Refer to Chapter 5, *Maintaining the Network Map File*, in the *Connect:Direct for UNIX Administration Guide* for a description of the network map file.

Connect:Direct must be installed on each node. When Connect:Direct establishes a session between the local node and remote node, the node that initiates the session has primary control (PNODE). The other serves as the partner and has a secondary function (SNODE). The node that initiates the session has primary control, regardless of the direction of information flow. The Process can specify work destined for either the local or remote node. When Connect:Direct establishes a session, the two nodes work together to transfer the information.

Processes

The Connect:Direct Process language provides instructions for transferring files, running programs, submitting jobs on the adjacent node, and altering the sequence of Process step execution. You can include one or more steps in a Process. A Process consists of a Process definition statement (**process** statement) and one or more additional statements. Parameters further qualify Process instructions.

The following table lists Process statements and their functions:

Process Statement	Function
copy	Copies files from one node to another.
conditionals	Alters the sequence of Process execution based on the completion code of previous steps with the **if**, **then**, **else**, **eif** (end if), **goto**, and **exit** statements.
process	Defines general Process characteristics.
run job	Enables you to specify UNIX commands in a Process. The Process does not wait until the job has finished running before executing the next step in the Process.
run task	Enables you to specify UNIX commands in a Process. The Process waits until the job has finished running before executing the next step in the Process.
submit	Starts another Process to either the local or remote node during execution of a Process.
pend	Marks the end of a Connect:Direct for UNIX Process.

Following is a sample Process:

```
ckpt01  process  snode=unix.node
step01 copy from (
             file=file1
             snode
             )
         ckpt=1M
         to   (
             file=file2
             disp=new
             pnode
             )
    pend;
```

Refer to the Connect:Direct Processes Web site at http://www.sterlingcommerce.com/documentation/processes/processhome.html for instructions on building a Process.

Transmission Control Queue

The Transmission Control Queue (TCQ) controls when Processes run. Connect:Direct stores submitted Processes in the TCQ. The TCQ is divided into four logical queues: Execution, Wait, Timer, and Hold. Processes are run from the Execution queue.

Connect:Direct places a Process in the appropriate queue based on Process statement parameters, such as the **hold**, **retain**, and **startt** parameters.

Connect:Direct runs Processes based on their priority and when the Process is placed in the Execution queue. Higher priority Processes are selected for execution ahead of Processes with a lower priority. You can access the queues and manage the Processes through Connect:Direct commands.

Refer to the *Connect:Direct for UNIX User's Guide* for more information on scheduling Processes in the TCQ.

Commands

You use commands to submit Processes to the TCQ. You can also use commands to perform the following tasks:

- Manage Processes in the queue
- Monitor and trace Process execution
- Produce reports on Process activities
- Stop Connect:Direct operation

The following table lists the commands and their functions:

Command	Function
change process	Changes the status and modifies specific characteristics of a nonexecuting Process in the TCQ.
delete process	Removes a nonexecuting Process from the TCQ.
flush process	Removes an executing Process from the TCQ.
trace	Runs Connect:Direct traces.
select process	Displays or prints information about a Process in the TCQ.
select statistics	Displays or prints statistics in the statistics log.
stop	Stops Connect:Direct operation.
submit	Submits a Process for execution.

For example, the following command submits the Process called **onestep** to the TCQ with a hold status of yes:

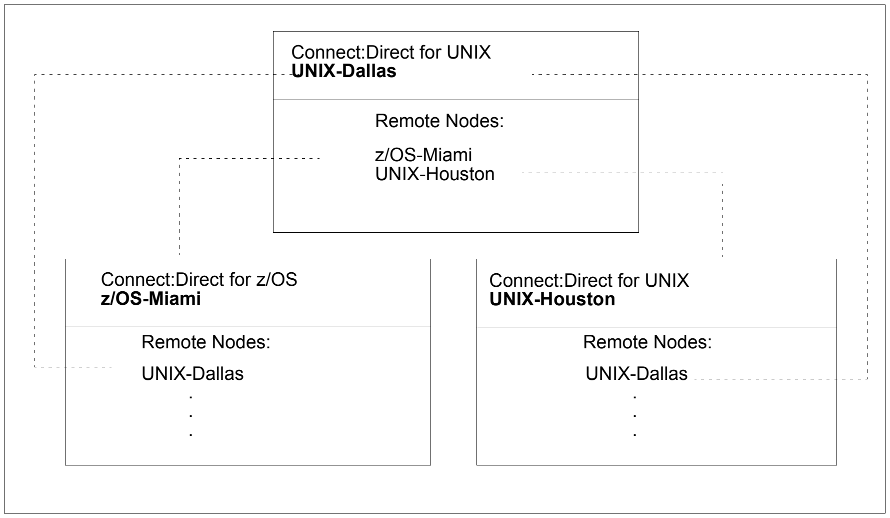
```
Direct> submit file=onestep hold=yes;
```

Network Map

During the transfer of data, the Connect:Direct server where the Process is submitted is the primary node and the secondary node is the partner node. Connect:Direct identifies the remote nodes that the local node is able to communicate with through the use Connect:Direct network map (or netmap).

The network map includes the names of all the remote nodes that the Connect:Direct local node can communicate with, the paths to contact those remote nodes, and characteristics of the sessions for communication.

The remote Connect:Direct nodes also have network maps for their remote nodes. The following sample displays the corresponding network map entries of UNIX-Dallas, z/OS-Miami, and UNIX-Houston:



User Authorization

Connect:Direct can authorize local and remote users to perform certain Connect:Direct tasks. In order to use Connect:Direct, each user must have a record defined in the user authorization file, called userfile.cfg. Each local user must have a record in the user authorization file, and each remote user must be mapped to a local user ID in a proxy relationship.

To provide a method of preventing an ordinary user from gaining root access through Connect:Direct for UNIX, a second access file called the Strong Access Control file (SACL) is created when you install Connect:Direct and is named sysacl.cfg. Creating an SACL file improves the security of Connect:Direct UNIX and prevents a user from obtaining root access control. At installation, the default value for the root:deny.access parameter is y (yes). If you want to allow root access, you must access the sysacl.cfg and specify root:deny.access=n. If the file is deleted or corrupted, access to Connect:Direct is denied to all users.

The Connect:Direct administrator maintains these authorizations. Refer to Chapter 6, *Maintaining Access Information Files* in the *Connect:Direct for UNIX Administration Guide* for more information about authorizations.

Process Restart

Several facilities are provided for Process recovery after a system malfunction. The purpose of Process recovery is to resume execution as quickly as possible and to minimize redundant data transmission after a system failure. The following Connect:Direct facilities are available to enable Process recovery:

- Process step restart—As a Process runs, the steps are recorded in the TCQ. If a Process is interrupted for any reason, the Process is held in the TCQ. When you release the Process to continue running, the Process automatically begins at the step where it halted.
- Automatic session retry—Two sets of connection retry parameters are defined in the remote node information record of the network map file: short-term and long-term. If you do not specify a value for these parameters in the remote node information record, default values are used from the local.node entry of the network map file. The short-term parameters allow immediate retry attempts. Long-term parameters are used after all short-term retries are attempted. Long-term attempts assume that the connection problem cannot be fixed quickly and retry attempts occur after a longer time period, thus saving the overhead of connection retry attempts.
- Checkpoint restart—This feature is available with the **copy** statement.

  Checkpoint restart can be explicitly configured within a **copy** step through the **ckpt** parameter. (Refer to the Connect:Direct Processes Web site at http://www.sterlingcommerce.com/documentation/processes/processhome.html) If it is not configured in the **copy** step, it can be configured in the Initparms through the **ckpt.interval** parameter. (See the *Connect:Direct for UNIX Administration Guide* for more information on this parameter.)
- Run Task restart—This feature is used if the PNODE and SNODE cannot resynchronize during a restart after a run task interruption. If a Process is interrupted when a run task on an SNODE step is executing, Connect:Direct attempts to synchronize the previous run task step on the SNODE with the current run task step. Synchronization occurs in one of the following ways:
  - If the SNODE is executing the task when the Process is restarted, it waits for the task to complete, and then responds to the PNODE with the task completion status. Processing continues.
  - If the SNODE task completes before the Process is restarted, it saves the task results. When the Process is restarted, the SNODE reports the results, and processing continues.

  If synchronization fails, Connect:Direct reads the **restart** parameter in the **run task** step or the initialization parameters file to determine whether to perform the **run task step** again. The **restart** parameter on the **run task** step overrides the setting in the initialization parameter.

  For example, if the SNODE loses the run task step results due to a Connect:Direct cold restart, Connect:Direct checks the value defined in the **restart** parameter to determine whether to perform the **run task** again.

Note: Run task restart works differently when Connect:Direct runs behind a connection load balancer. For more information on the considerations you need to know when running Connect:Direct in a load balancing environment, see the *Connect:Direct Release Notes* and the *Connect:Direct Administration Guide*.

- Interruption of Process activity when the SNODE is a Connect:Direct for UNIX node—When the SNODE is a Connect:Direct for UNIX node and the PNODE interrupts Process activity by issuing a command to suspend Process activity, deleting an executing Process, or when a link fails or an I/O error occurs during a transfer, the Process is placed in the Wait queue in WS status.

If Process activity does not continue, you must manually delete the Process from the TCQ. Refer to the *Connect:Direct for UNIX User's Guide* for command syntax and parameter descriptions for the **delete process** command.

Note: You cannot issue a **change process** or the **flush process** command from the SNODE to continue Process activity; the Process can only be restarted by the PNODE, which is always in control of the session.

Archive Statistics Files

Connect:Direct provides a utility to archive and purge statistics files. When you configure Connect:Direct, you identify when to archive a statistics file by setting the parameter, **max.age**, in the stats record of the initialization parameters file. The **max.age** parameter defines how old a statistics file must be before you want to archive the file.

Once a day, the script called statarch.sh is started. This script identifies the statistics files that are equal to the **max.age**. It then runs the tar command and the compress command to create a compressed archived file of all the statistics records that match the **max.age** parameter. Once the statistics files are archived, these files are purged. For files archived on a Linux computer, the archived statistics files have the .gz suffix since these files are compressed with the gzip format. Archived files on all other UNIX platforms have the .Z suffix to indicate they are compressed using the compress format.

The archived files are stored in the directory where the statistics files and TCQ are stored. The shell script, statarch.sh, is located in the ndm/bin directory. If necessary, modify the script to customize it for your environment.

If you want to restore statistics files that have been archived, run the **statrestore.sh** script. It uses the **uncompress** and **tar** commands to restore all the statistics files in the archive. You supply two arguments to the **statrestore** command. The first argument is the directory path where the statistics files are located and the second argument identifies the archived file name followed by as many archived file names as you want to restore. Below is a sample **statrestore** command:

```
qa160sol: ./statrestore.sh /export/home/users/cd3800/ndm/bin archive1
```

After files are restored, the statistics records can be viewed using the select statistics command.

Sample Processes, Shell Scripts, and API Processes

Connect:Direct provides sample Processes and shell scripts in *d_dir*/ndm/src, where *d_dir* indicates the destination directory of the Connect:Direct software. You can create similar files with a text editor. In addition, instructions for creating shell scripts are in the README file in the same directory.

The following table displays the file names of sample Processes. Modify the Processes as required.

File Name	Type of Process
cpunx.cd	copy
rtunx.cd	run task
rjunx.cd	run job
sbunx.cd	submit

The following table displays the file names of sample shell scripts. Modify the shell scripts as required.

File Name	Type of Shell Script
selstat.sh	select statistics
send.sh	send
recv.sh	receive
wildcard	send multiple files to an OS/390 PDS
statarch.sh	archive statistics files
statrestore.sh	restore statistics files that have been archived
lcu.sh	launch the Local Connection Utility tool
spadmin.sh	launch the Secure+ Option Admin Tool
spcli.sh	launch the Secure+ Option CLI (SPCLI)
spcust_sample1.sh	configure Secure+ Option for the STS protocol
spcust_sample2.sh	configure Secure+ Option for the STS protocol
spcust_sample3.sh	configure Secure+ Option to use the SSL or TLS protocol

Connect:Direct Files

This section describes the configuration files, illustrates the directory structure of Connect:Direct, and lists the individual files that are installed.

Connect:Direct for UNIX Configuration Files

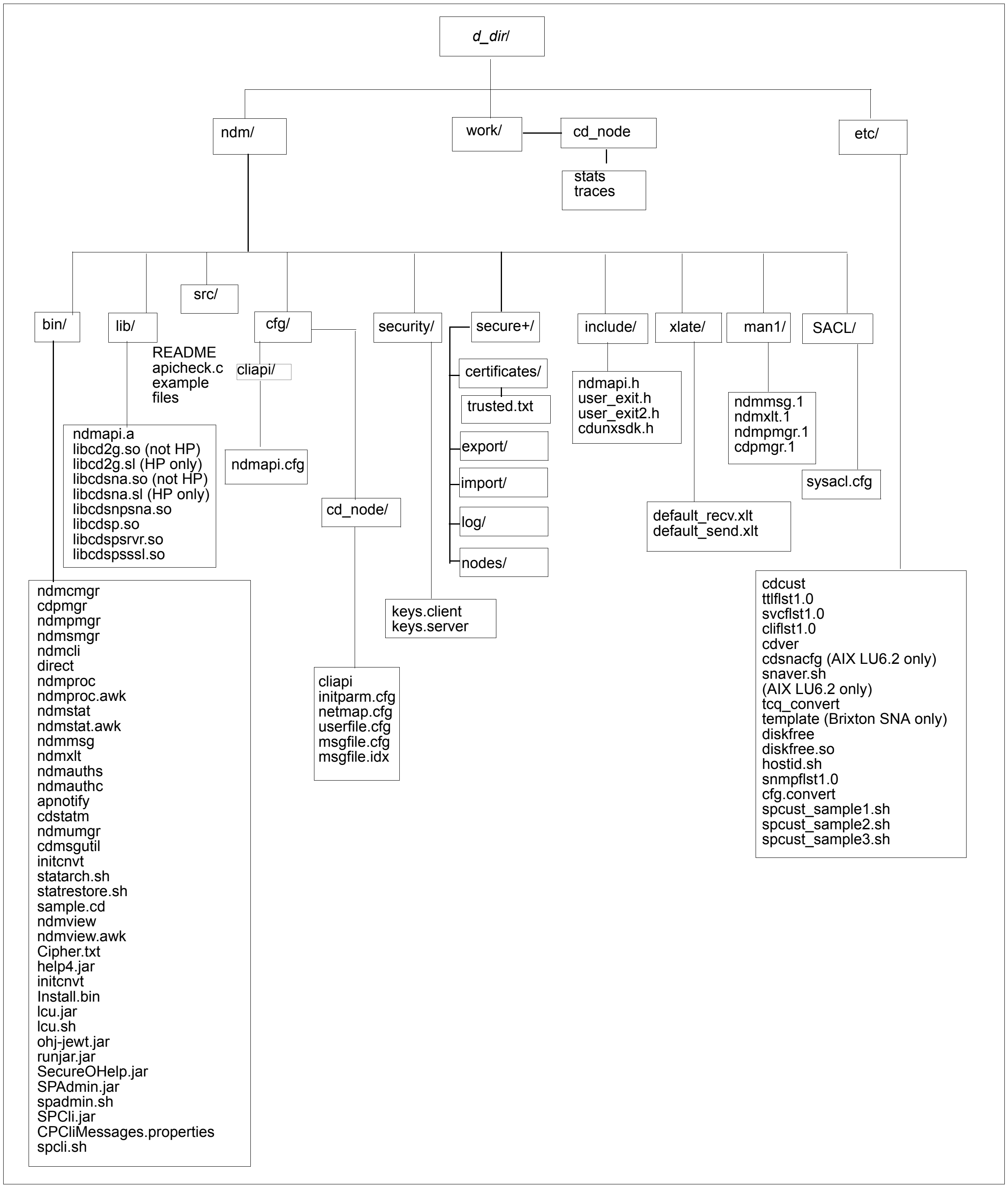
Connect:Direct creates the following configuration files during installation and customization. These files are required for the Connect:Direct server to operate correctly.

Configuration File	Description
Initialization parameters file	Provides information to the server to use at start up. During the installation, you identify the settings necessary for the initialization parameters file.
User authorization information file	Contains the local user information and remote user information record types. You customize this file during installation to map remote user IDs to local user IDs and create remote user information records in the user authorization information file.
Strong access control file	Improves the security of Connect:Direct UNIX and prevents a user from obtaining root access control. Initially, the file is empty and contains no restrictions. However, if you want to prevent the root user or any user with root permissions from gaining access to Connect:Direct, you define a restriction parameter in this file. If the file is deleted or corrupted, access to Connect:Direct is denied to all users.
Network map file	Describes the local node and other Connect:Direct nodes in the network. You can define a remote node record for each node that Connect:Direct for UNIX communicates with.
Server authentication key file	Verifies client API connection requests. Only clients are granted a connection.
Client configuration file	Identifies the port and host name used by a client to connect to Connect:Direct.
Client authentication key file	Identifies Connect:Direct servers that a Connect:Direct node connects to. You can have multiple entries for multiple servers.

Connect:Direct for UNIX Directory Structure

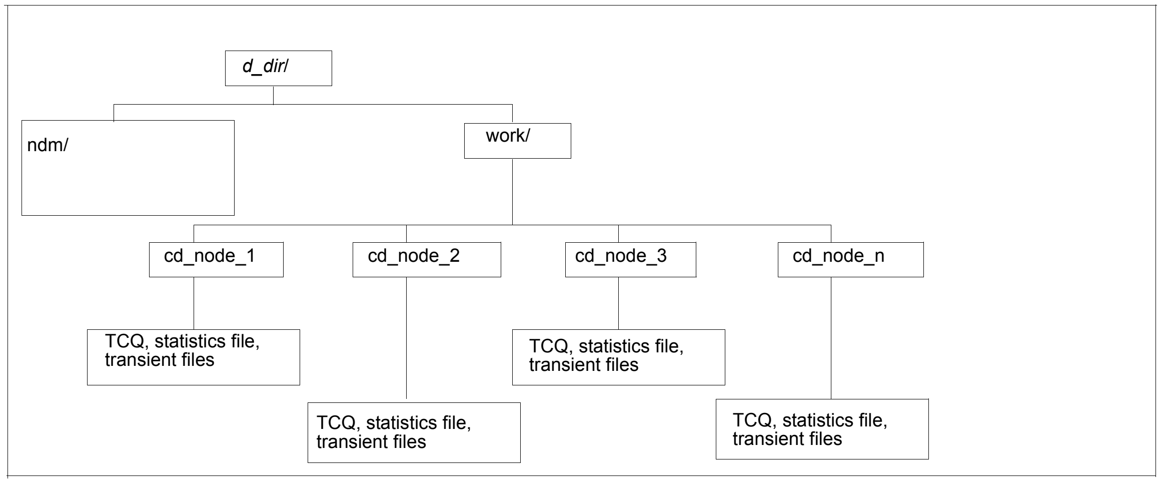
The following figure illustrates the Connect:Direct for UNIX directory structure. The directory tree starts at *d_dir/*, the destination directory where the software is installed. This directory structure provides for multiple nodes on the same network and possibly on the same computer. The directory structure organization enables you to share Connect:Direct programs, such as cdpmgr and dmcmgr. If multiple nodes exist, each node must have its own *d_dir*/ndm/cfg/*cd_node*/ directory structure for configuration files, where *cd_node* is the Connect:Direct node name.

Note: The secure+ directory is available only when Secure+ Option is purchased.



Note: See the following figure to view the work directory for a node.

A *d_dir*/work/*cd_node* directory is created for each node. The following figure displays the work directory for multiple nodes and illustrates the working files created for each node, such as TCQ files:


d_dir/
ndm/
work/
cd_node_1
cd_node_2
cd_node_3
cd_node_n
TCQ, statistics file, transient files
TCQ, statistics file, transient files
TCQ, statistics file, transient files
TCQ, statistics file, transient files

Note: See the previous figure for details on the ndm directory structure.

Connect:Direct Documentation

See *Connect:Direct for UNIX Release Notes* for a complete list of the product documentation.

About This Guide

Connect:Direct for UNIX User's Guide is for programmers and network operations staff who use Connect:Direct for UNIX.

This guide assumes knowledge of the UNIX operating system, including its applications, network, and environment. If you are not familiar with the UNIX operating system, refer to the UNIX library of manuals.

Task Overview

The following table guides you to the information required to perform Connect:Direct tasks:

Task	**Reference**
Understanding Connect:Direct	Chapter 1, *About Connect:Direct for UNIX*
Understanding the parameters required to invoke the Command Line Interface (CLI)	Chapter 2, *Controlling and Monitoring Processes*
Understanding the TCQ and the commands necessary to manage Processes in the queue	Chapter 3, *Process Queuing*
Understanding the Connect:Direct translation table and message file utilities	Chapter 4, *Using Connect:Direct Utilities*
Understanding function calls necessary to write custom programs to use with Connect:Direct	Chapter 5, *Writing Custom Programs*
Understanding user exits and user exit function calls to use with Connect:Direct	Chapter 6, *User Exits*

Chapter 2

Controlling and Monitoring Processes

This chapter provides instructions on using the Command Line Interface (CLI) to submit Processes and commands.

Note: You can also use the Connect:Direct Browser User Interface to perform some of the procedures in this chapter. To learn more about the Connect:Direct Browser, see the user's guide on the Connect:Direct Browser CD-ROM or available online from the Sterling Commerce Documentation Library.

Using the Command Line Interface

The Command Line Interface (CLI) enables you to submit Connect:Direct Processes and commands from a native command line environment. This section describes the commands used during CLI operation.

Starting CLI

Perform the following steps to start the CLI:

1. If you have not defined the NDMAPICFG environment variable, type the following command for the appropriate shell, where *d_dir* is the path to the Connect:Direct subdirectory.
   - In the C shell:

   ```
   % setenv NDMAPICFG d_dir/ndm/cfg/cliapi/ndmapi.cfg
   ```

   - In the Bourne or Korn shell:

   ```
   $NDMAPICFG=d_dir/ndm/cfg/cliapi/ndmapi.cfg
   $export NDMAPICFG
   ```

2. Type the following command to invoke Connect:Direct CLI. Type options as required:

```
$ direct [-P string -s -t n -e nn -n name -p nnnnn -x -r -h -z]
```

Refer to the following table for a description of the command options and sample command entries:

Option	Description	Value	Sample Command Entry
-P	Identifies the custom string to use at the command line prompt. If the prompt string includes spaces or special characters, enclose it in single or double quotation marks. The prompt string can also be specified in the ndmapi.cfg file. If a prompt string is specified on the command line and in the ndmapi.cfg file, -P takes precedence. When the default prompt (“Direct”) is overridden, the new prompt string is shown at the command line prompt and in the welcome banner display.	text string Up to 32 characters.	$ direct -PNewPrompt $ direct -P”Test CD on Medea”
-s	Suppresses standard output. Use this option to view only the completion status of a command.	none	$ direct -s
-t n	Enables the CLI/API trace option. The level number, **n**, identifies the level of detail in the trace output.	1 | 2 | 4 Specify one of the following level numbers: 1—Provides function entry and function exit. 2—Provides function entry and exits and basic diagnostic information, such as displaying values of internal data structures at key points in the execution flow. 4—Enables a full trace. All diagnostic information is displayed.	$ direct -t 4

Option	Description	Value	Sample Command Entry
-e nn	Defines the error level above which the CLI automatically exits. If the returned error code is greater than the error level specified, the CLI automatically exits. Use this command within shell scripts. This parameter prevents unwanted execution of commands following a command that generates an error above the specified level. When the CLI terminates, it returns a UNIX exit code that can be tested by the shell.	0 | 4 | 8 | 16 Valid values in the error level code are: 0—Indicates successful completion. 4—Indicates warning. 8—Indicates error. 16—Indicates catastrophic error.	$ direct -e 16
-n name†	Identifies the host name of the computer where the Connect:Direct server (PMGR) is running. **Note:** Invoking **direct** with **-p** or **-n** overrides the settings in the ndmapi.cfg file.	Connect:Direct host name	$ direct -n *hostname*
-p nnnnn†	Identifies the communications port number for the Connect:Direct node. **Note:** Invoking **direct** with **-p** or **-n** overrides the settings in the ndmapi.cfg file.	1024–65535. The format is **nnnnn**.	$ direct -p 2222
-x	Displays command input on standard out. Use this command when debugging scripts.	none	$ direct -x
-r	Makes the Process number available to user-written shell scripts. The CLI displays a special string, _CDPNUM_ followed by a space, followed by the Process number.	none	$direct -r | grep "_CDPNUM_"
-h	Displays command usage information if a Connect:Direct command is typed incorrectly.	none	$ direct -h
-z	Appends a newline character after a prompt.	none	$ direct -z

Using Job Control

Connect:Direct enables you to switch the CLI Process between the foreground and the background in shells that support job control. This capability enables you to edit the text of saved Processes, issue UNIX commands, and resolve Process errors without exiting and reentering the CLI. Use the following commands to switch the CLI Process:

- ✦ Press the suspend character (**Control-Z**) to stop or suspend the CLI Process.
- ✦ Issue the **fg** command to move the CLI Process to the foreground.

Note: If you experience problems with job control, contact your system administrator for suggestions on additional UNIX commands to use.

Using History with the CLI

Connect:Direct enables you to use the history commands available with UNIX. History commands do not need the semicolon (;) at the end of the command. The following table lists the available history commands:

Command	Description
!!	Repeat the last command one time.
!#n	Set the number of commands to store in the history buffer. The default history buffer size is 50 commands.
!n	Repeat command number <n> in the history buffer.
!<string>	Repeat command beginning with the string <string>.
!?	List the contents of the history buffer.

Stopping the CLI

Stop the CLI operation by typing **Control-D** or **quit;** at the prompt.

Guidelines for Using Connect:Direct Commands

Refer to the information in this section for guidelines for using Connect:Direct commands:

Command Overview

You control and monitor Connect:Direct Processes using the following commands:

Command	Abbreviation	Description
submit	sub	Makes Processes available for execution.
change process	cha pro	Changes the status and modifies specific characteristics, of a nonexecuting Process in the TCQ.
delete process	del pro	Removes a nonexecuting Process from the TCQ.
flush process	flush pro	Removes an executing Process from the TCQ.
stop	stop	Stops Connect:Direct and returns control to the operating system.
select process	sel pro	Monitors both executing Processes and Processes waiting for execution. You can specify the search criteria and the form in which the information is presented.
select statistics	sel stat	Retrieves information from the statistics file. You can specify the search criteria and the form in which the information is presented.
view process	view pro	View a Process in the TCQ where the local node is the Pnode. View process can only display Processes running on the local node since only the Pnode has the information required to display a Process.

Note: The CMGR currently limits the size of a Process file to 60K bytes.

Parameter Abbreviations

The following table lists valid abbreviations for commonly used parameters:

Parameter	Abbreviation
detail	det
quit	q
recids	rec
release	rel
pname	pnam, pna
pnumber	pnum
sunday	sun
monday	mon
tuesday	tue
wednesday	wed

Parameter	Abbreviation
thursday	thu
friday	fri
saturday	sat
today	tod
tomorrow	tom

Limitations on Using Programs and Scripts

System administrators and other network operations staff can restrict the scripts and UNIX commands that you can execute with the run task and run job Process statements.

System administrators and other network operations staff can enforce the following limits on the capabilities you have with Connect:Direct:

- The capability to send or receive files; you may be limited either to sending files only or to receiving files only.
- The locations to or from which you can send or receive files; you may be limited to specific local or remote nodes.

Check with the system administrator for a list of specific restrictions for your user ID.

Command Syntax

Use the same command syntax for commands typed at the CLI prompt or used as the command text parameter for an **ndmapi_sendcmd()** function. Refer to *Writing Custom Programs* on page 75, for details on function calls. The following conventions are used when typing commands:

- When selecting a password or user ID, do not use Connect:Direct keywords.
- Be aware that user names and file names are case sensitive.
- Type an individual command keyword in uppercase, lowercase, or mixed-case characters.
- Terminate all commands with a semicolon (;).
- When typing commands, type the entire command name or type the first three characters or abbreviate specific parameters. Refer to the table on page 27 for a list of abbreviations.
- Do not abbreviate Process statements and parameters.
- File names, group names, user IDs, and passwords are variable length strings and can be any length.
- If defined in the netmap.cfg file, a node name can be 1–16 characters long. If not defined in the netmap.cfg file, a node name can be 1–256 alphanumeric characters long.

Generic

When the word **generic** is specified as a parameter value in a syntax definition, provide a string that can include the asterisk (*****) and question mark (**?**) characters. These characters provide a pattern matching or wildcard facility for parameter values. The asterisk matches zero or more characters, and the question mark matches any single character. The following sample illustrates the use of the asterisk and question mark characters:

```
PNAME = A?PROD5*
```

The generic Process name specified in the previous sample shows a specification that matches all Processes beginning with the letter **A**, followed by any single character in position two with the string **PROD5** in positions three through seven. The asterisk takes the place of zero or more characters beginning in position eight.

Lists

When **(list)** is a parameter value, you can specify multiple parameter values by enclosing the group in parentheses and separating each value with a comma. A list can also include generic values. The following command illustrates a list:

```
(pnumber1, pnumber2, pnumber3)
```

Submitting a Process

Use the **submit** command to make Processes available for execution and to enable the software to interpret the Process statements contained in the specified files.

Parameters for submit Command

Parameters specified in the submit command override the same parameters specified on the process statement. There are no required parameters. However, if you do not specify a file name for the file parameter, the text of the Connect:Direct Process must follow the submit command. Following are the parameters for the submit command:

Parameter	Description	Values
file	The name of the Process file. The file name can include a path name indicating the location of the Process. This parameter must be the first parameter.	file name including the path name
class	The node-to-node session on which a Process can execute. A Process can execute on the class specified or any higher session class. The default class is specified as the **sess.defaul**t parameter of the local.node record in the initialization parameters file.	1 | n

Parameter	Description	Values
crc	Determines if crc checking is performed. This parameter overrides settings in the initialization parameter, the network map, and the Process. **Note:** The user must be assigned authority to change the crc settings in the user authority file.	on | off on—Turns on crc checking. off—Turns off crc checking.
hold	Determines if the Process is placed in the Hold queue. When a Process is submitted with retain=yes or retain=call, Connect:Direct ignores the **hold** parameter.	yes | no | call **yes**—Specifies the Process is placed in the Hold queue in HI status until it is released by a change process command. A Process submitted with hold=yes is placed on the Hold queue even if you specify a start time. **no**—Specifies that the Process executes as soon as resources are available. **call**—Specifies that the Process is held until a connection is established between the remote node and the local node. At that time, the Process is released for execution.
maxdelay	How long the submit command waits for the submitted Process to complete execution. This parameter is useful when the command is issued by a shell script. When this parameter is specified, the script waits until the Process completes before it continues execution. The return code of the Process is stored in the $? variable if you are using the Bourne or Korn shell and in $status variable if you are using the C shell, which the shell script can use to test the results of Process execution. If you do not specify maxdelay, no delay occurs. If the time interval expires, the submit command returns a warning status code and message ID to the issuing Process or CLI/API. The Process is not affected by the time interval expiration and executes normally.	unlimited | hh:mm:ss | 0 **unlimited**—Waits until the Process completes execution. **hh:mm:ss**—Waits for an interval no longer than the specified hours, minutes, and seconds. **0**—Waits until the Process completes execution. If you specify maxdelay=0, you get the same results as when you specify maxdelay=unlimited.
newname	A new Process name that overrides the name in the submitted Process.	A name up to 256 characters long

Parameter	Description	Values
notify	The user E-mail to receive Process completion messages. This parameter uses the rmail utility available in the UNIX System V mail facility to deliver the completion messages. **Note:** Connect:Direct does not validate the E-mail address or user ID supplied to the notify parameter. Invalid E-mail addresses and failed E-mail attempts are handled according to the local mail facilities configuration.	username@hostname or user@localhost
pacct	A string containing information about the PNODE. Enclose the string in double quotation marks.	“pnode accounting data” up to 256 characters
pnodeid	Security user IDs and passwords at the PNODE. The pnodeid subparameters can contain 1–64 alphanumeric characters.	id [, pswd] **id**—Specifies a user ID on the PNODE. **pswd**—Specifies a user password on the PNODE. If you specify pnodeid, you must also specify id. Identify the ID first and the pswd last.
prty	The priority of the Process in the Transmission Control Queue (TCQ). A Process with a higher priority is selected for execution before a Process with a lower priority. The prty value does not affect the priority during transmission.	1–15 where fifteen is the highest priority. If you do not specify **prty**, the default is **10**.
retain	Determines if Connect:Direct retains a copy of the Process in the TCQ. Connect:Direct assigns a Process number to the Process when it is placed in the retain queue. When the Process is run, the Process number assigned to the retain Process is incremented by one. For example, if the Process is assigned the Process number of 1445 in the retain queue, the Process number is 1446 when the Process is executed. If you specify a start time and set **retain=yes**, the Process remains in the Timer queue in HR status and is submitted at the appropriate interval. For example, when **startt=(Monday,2:00)**, the Process runs each Monday at 2:00 AM. When **startt=(,1:00)**, the Process runs daily at 1:00 AM. Connect:Direct for UNIX does not provide a way to run a Process hourly. To do this, you must use the UNIX cron utility. If no start time is identified, you must issue a **change process** command to release the Process for execution. Do not code the **startt** parameter when you specify **retain=initial**.	yes | <u>no</u> | initial yes—Specifies that the system retains the Process in the Hold queue in HR status after execution. no—Specifies that the system deletes the Process from the TCQ after execution. The default is no. initial—Specifies that the system retains the Process in the Hold queue in HR status for automatic execution every time the Process Manager initializes.

Parameter	Description	Values
sacct	Specifies accounting data for the SNODE. Setting this value in the submit statement overrides any accounting data specified in Process.	“snode accounting data” up to 256 characters. Enclose the string in double quotation marks.
snode	Identifies the name of the secondary node. Setting this value overrides the snode value in the process statement. The **snode** parameter is required either on the submit command or process statement.	name | host name | nnn.nnn.nnn.nnn[;port name|nnnnn]] **name—**Specifies the node name of the remote node. The secondary node name corresponds to an entry in the network map file. **host name**—Specifies the name of the host computer where the remote Connect:Direct node is running. **nnn.nnn.nnn.nnn**—Specifies the IP address of the remote node. Each nnn is a decimal number from 0–255, inclusive. **[;port name|nnnnn]**—Identifies the communications port. You can only use this parameter with the host name or IP address parameters. The nnnnn value is a decimal number from 1,024–65,535.

Parameter	Description	Values
snodeid	Specifies security user IDs and security passwords on the SNODE. The snodeid subparameters can contain one or more alphanumeric characters. If Connect:Direct finds that a Process has no **snodeid** parameter or defines a **snodeid** parameter and the initialization parameter **proxy.attempt** is set to **y**, then any password specified on the **snodeid** parameter is ignored. A proxy user record is a remote user record in the userfile.cfg, which corresponds to the user name specified on the snodeid parameter. If no proxy user record exists, the **snodeid** parameter must contain a valid user name and password for a UNIX user who has a corresponding local user record in the userfile.cfg. When **proxy.attempt=n** and no snodeid is defined, Connect:Direct uses the submitting ID and node to find a Remote User Information record in the User Authorization Information file. If Connect:Direct cannot find a match, then that user cannot send or receive files. If the initialization parameters file parameter **proxy.attempt** is set to y, users are not required to specify a password for the **snodeid** parameter. This capability enables the id subparameter to contain a dummy user ID to be used for translation to a local user ID on the remote system. The use of a dummy user ID offers improved security because neither the sender nor the receiver are required to use an actual user ID. Reserved keywords cannot be used in the snodeid field.	id [,pswd [,newpswd]] **id**—Specifies a user ID on the SNODE. **pswd**—Specifies a user password on the SNODE. If you specify id, you do not have to specify pswd. This capability enables the id parameter to contain a dummy ID to be used for translation to a local ID on the remote system. **newpswd**—Specifies a new password value. On certain platforms, the user password changes to the new value on the SNODE if the user ID and old password are correct (refer to documentation on the specific platform). If the SNODE is a UNIX node, the password does not change. If you specify pswd, you must also specify id. If you specify newpswd, you must also specify pswd. Type the values in the order of id, pswd, and newpswd.

Parameter	Description	Values
startt	Identifies the date, day, and time to start the Process. Connect:Direct places the Process in the Timer queue in WS (Waiting for Start Time) status. The date, day, and time are positional parameters. If you do not specify date or day, a comma must precede time. Do not code the **startt** parameter when you **specify retain=initial**.	[date | day] [,hh:mm:ss [am | pm]] **date**—Specifies the day (dd), month (mm), and year (yy), which you can code as mm/dd/yyyy or mm-dd-yyyy. If you only specify date, the time defaults to 00:00:00, which indicates midnight. The current date is the default. **day**—Specifies the day of the week. Values are today, tomorrow, yesterday, monday, tuesday, wednesday, thursday, friday, saturday, and sunday. **hh:mm:ss [am | pm]—**Specifies the time of day in hours (hh), minutes (mm), and seconds (ss). You can specify the hour in either 12- or 24-hour format. If you use 12-hour format, then you must specify am or pm. The default is the 24-hour format. The default value is 00:00:00, which indicates midnight. If you specify only the day value, the time defaults to 00:00:00. This means that if you submit a Process on Monday, with monday as the only **startt** parameter, the Process does not run until the following Monday at midnight.
&symbolic name1 &symbolic name 2 . . &symbolic name n	Specifies a symbolic parameter assigned a value. The value is substituted within the Process when the symbolic parameter is encountered. The value for the symbolic parameter must be in double quotation marks if it is a keyword or contains special characters. The symbolic name itself must not be a subset of any other symbolic name (you cannot have, for example, a symbolic name called &param and another symbolic name called &parameter in the same Process).	variable string 1 variable string 2 variable string n The symbolic name cannot exceed 32 characters.

Parameter	Description	Values
tracel	Specifies the level of trace to perform for a Process. Tracing by Process can be turned on in the submit command or as part of the Process definition. If you identify the snode or pnode immediately after the trace level definition, the trace level is turned on for all Processes submitted to and from the node identified.	level = 0 |1 | 2 | 4 snode | pnode file=name **level**—Specifies the level of detail displayed in the trace output. The default is 4. **0**—Terminates the trace. **1**—The basic level that provides function entry and function exit. **2**—includes level 1 plus function arguments. **4**—Enables a full trace. Basic diagnostic information, such as values of internal data structures at key points in the execution flow, are displayed. **snode**—Specifies to trace only the SNODE SMGR. **pnode**—Specifies to trace only the PNODE SMGR. **file**—Specifies the name of a file where the trace output is directed. If you do not specify a file name, the file is created in the Connect:Direct working directory with the file name CMGR.TRC. The length of the name value is unlimited.

Sample submit Commands

Following are sample **submit** commands.

Submitting a Process That Runs Every Week

The following command submits the Process named payroll:

```
submit file=payroll retain=yes startt=monday pacct="1959,dept-27";
```

Because **retain=yes** is specified in this sample, the Process is retained in the TCQ after execution. The Process starts next Monday at 00:00:00 and runs every Monday thereafter. Process accounting data is specified for the PNODE. See Chapter 3, *Process Queuing*, for additional information about the TCQ.

Submitting a Process with a Start Time Specified

The following command submits the Process named copyfil:

```
submit file=copyfil snode=vmcent startt=(01/01/2004, 11:45:00 am);
```

Because **startt** is specified, the Process executes on the first day of January 2004 at 11:45 a.m.

Submitting a Process with No File Value

The following command submits a Process without a **file** parameter value, but with the Process statements typed at the CLI command prompt:

```
Direct> sub do_copy process  snode=node1
step01 copy from (
                      file=data.data
                      pnode
                 )
            to   (
                      file=b
                      snode
                 )
      pend ;
Process Submitted, Process Number = 5
```

Submitting a Process and Turning On Tracing

The following command submits the Process named copy.cdp:

```
submit file=copy.cdp tracel=4 pnode;
```

Because **tracel** is specified and the **pnode** parameter is included, an SMGR and COMM full trace is performed on the Process. Trace information is written to the default file SMGR.TRC.

Changing Process Parameters

Use the change process command to modify specified parameters for a *nonexecuting* Process.

Specify the Processes to be changed by Process name, Process number, secondary node name, and submitter.

You can change the class, destination node, and priority. You can place a Process on the Hold queue or release a Process from the Hold queue by issuing a change process command with either the **release** or **hold=no** parameter.

If you submit a Process with a **startt** parameter, Connect:Direct places the Process on the Timer queue. If a Process fails, you can move it to the Hold queue by specifying the change process command with hold=yes. Connect:Direct then places the Process in the Hold queue in HO status. You can release the Process for execution at a later time.

You can set tracing for an existing Process by setting the **tracel** parameter to 1, 2, or 4. You can turn off tracing for a Process by setting trace1 to 0.

Specify at least one of the following search criteria parameters:

Parameter	Description	Value
pname	Locate the Process to be changed by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name, up to 8 alphanumeric characters. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.
pnumber	Locate the Process to be changed by Process number. Connect:Direct assigns the Process number when the Process is submitted.	number from 1–99,999 | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma.
snode	Locate the Process to be changed by the secondary node name. If defined in the netmap.cfg file, the secondary node name can be 1–16 characters long. If not defined in the netmap.cfg file, the secondary node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.
submitter	Locate the Processes to be changed by the node specification (the Connect:Direct node name) and user ID of the Process owner. The character length of this parameter is unlimited.	(node specification, userid) | generic | (list) **node specification, userid**—Specifies the node specification (the Connect:Direct node name) and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.

The optional parameters for the **change process** command are the following:

Parameter	Description	Value
class	Changes the node-to-node session on which a Process can execute. A Process can execute on the class specified or any higher session class. The default class is specified as the **sess.default** parameter of the local.node record in the initialization parameters file.	The default is **1**.
hold	Moves the Process to the Hold or Wait queue.	yes | no | call **yes**—Places the Process in the Hold queue in HO status until it is released by another **change process** command. **no**—Places the Process in the Wait queue in WC (Waiting for Connection) status; the Process executes as soon as resources are available. **call**—Places the Process in the Hold queue in HC (Hold for Call) status until the remote node (SNODE) connects to the local node (PNODE) or another Process is submitted. At that time, Connect:Direct releases the Process for execution
newsnode	Specifies a new remote node name to assign to the Process.	new remote node specification
prty	Changes the priority of the Process on the TCQ. Connect:Direct uses the **prty** parameter for Process selection. A Process with a higher priority is selected for execution before a Process with a lower priority. The **prty** value does not affect the priority during transmission.	1–5, where 15 is the highest priority. If you do not specify **prty**, the default is **10**.
release	Releases the Process from a held state. This parameter is equivalent to **hold=no**.	none

Parameter	Description	Value
tracel	Changes the level of trace to perform for a Process. If you identify the snode or pnode immediately after the trace level definition, the trace level is turned on for all Processes submitted to and from the node identified.	level = 0 |1 | 2 | 4 **level**—Specifies the level of detail displayed in the trace output. The default is 4. **0**—Terminates the trace. **1—**Is the basic level that provides function entry and function exit. **2** —Includes level 1 plus function arguments. **4**—Enables a full trace. Basic diagnostic information, such as values of internal data structures at key points in the execution flow, are displayed.

The following command changes the remote node name for the Process named cdproc to a new remote node, paris:

```
change process pname=cdproc newsnode=paris;
```

Deleting a Process

Use the delete process command to remove a *nonexecuting* Process from the TCQ.

You select the Process to delete by Process name, Process number, secondary node name, submitter, or any combination of the search criteria parameters. Specify at least one of the following search criteria parameters:

Parameter	Description	Value
pname	Identify the Process to delete by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name up to 8 alphanumeric characters long. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
pnumber	Identify the Process to delete by Process number. Connect:Direct assigns the Process number when the Process is submitted. Valid Process numbers range from 1–99,999.	number | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma (,).
snode	Identify the Process to delete by the secondary node name. If the secondary node name is defined in the netmap.cfg file, the name can be 1–16 characters long. If not, the node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.
submitter	Identify Processes to delete by the node specification and user ID of the Process owner. The character length of this parameter is unlimited.	(node specification, userid) | generic | (list) **node specification, userid**—Specifies the node specification and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.

The following command deletes all nonexecuting Processes submitted by user ID cduser on node dallas:

```
delete process submitter=(dallas, cduser);
```

Removing a Process from the Execution Queue

Use the **flush process** command to remove Processes from the Execution queue. You select the Process to remove by Process name, Process number, secondary node name, submitter, or any combination of the search criteria parameters. Specify at least one of the following search criteria parameters:

Parameter	Description	Value
pname	Locate the Process to remove by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name, up to 8 alphanumeric characters. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.
pnumber	Locate the Process to remove by Process number. Connect:Direct assigns the Process number when the Process is submitted.	number from 1–99,999 | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma.
snode	Locate the Process to remove by the secondary node name. If defined in the netmap.cfg file, the secondary node name can be 1–16 characters long. If not defined in the netmap.cfg file, the secondary node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.
submitter	Locate the Processes to remove by the node specification (the Connect:Direct node name) and user ID of the Process owner.	(node specification, userid) | generic | (list) **node specification, userid**—Specifies the node specification (the Connect:Direct node name) and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.

Following are the optional parameters for the **flush process** command:

Parameter	Description	Value
force	Forcibly terminates an executing Process or terminates a Process in an orderly fashion as the step completes. This parameter is useful if a Process is in the executing state and waiting for unavailable resources.	yes | no **yes**—Specifies to forcibly and immediately terminate the Process. The SMGR also terminates immediately. **no**—Specifies to terminate the Process in an orderly fashion as the step completes. The SMGR closes the statistics file and then terminates.
hold	Places the terminated Process in the Hold queue where it can be released for re-execution.	yes | no **yes**—Specifies to place the Process in the Hold queue in HS status after the Process is terminated. **no**—Specifies to delete the Process from the TCQ after the Process is terminated.

The following command flushes all executing Processes named from the Execution queue:

```
flush process pname=rome force=yes;
```

The following command flushes all executing Processes submitted by user ID jones on node alma:

```
flush process submitter=(alma, jones);
```

Stopping Connect:Direct

Use the stop command to initiate an orderly Connect:Direct shutdown sequence or forcibly terminate the software. After running the stop command, no new Processes are allowed to run and no new connections with remote systems are established. Commands can be issued and users can sign on until the server terminates.

You can specify the force, immediate, quiesce, or step parameters with the stop command.

Note: The **force** parameter is required when running Connect:Direct with the LU6.2 feature on any supported platform other than AIX.

Following are the parameters for the **stop** command:

Parameter	Description
force	Forcibly terminates Connect:Direct and returns control to the operating system.
immediate	Begins an immediate, but orderly shutdown of all activity and terminates Connect:Direct. The software terminates connections, writes statistics records, closes files, and shuts down.
quiesce	Runs all executing Processes to completion before shutting down Connect:Direct. No new Processes are started. This is the default value.
step	Shuts down Connect:Direct after all currently executing Process steps are complete. The software writes statistics records, closes files, and shuts down. All active Processes are retained in the TCQ. Processes restart when the software is re-initialized.

The following command forcibly terminates Connect:Direct and returns control to the operating system:

```
stop force;
```

Viewing a Process on the TCQ

Use the **view process** command to view Processes in the TCQ when the local node is the Pnode. You can search by Process name, Process number, queue, secondary node, status, owner of the Process, or any combination of the search criteria parameters.

You also can specify more than one Process in the search criteria.

There are no required parameters for this command. If you do not specify an optional parameter, Connect:Direct selects all Processes executing or waiting for execution. Following are the optional parameters for the **view process** command:

Parameter	Description	Value
pname	Locate the Process to view by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name, up to 8 alphanumeric characters. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
pnumber	Locate the Process to view by Process number. Connect:Direct assigns the Process number when the Process is submitted.	number from 1–99,999 | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma.
queue	Specifies the Processes to be viewed by the specified queue names. The default is **all**.	<u>all</u> | exec | hold | wait | timer **all**—Selects Processes from all queues. **exec**—Selects Processes from the Execution queue. **hold**—Selects Processes from the Hold queue. **timer**—Selects Processes from the Timer queue. **wait**—Selects Processes from the Wait queue.
snode	View the Process by the secondary node name. If defined in the netmap.cfg file, the secondary node name can be 1–16 characters long. If not defined in the netmap.cfg file, the secondary node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
status	Specifies the Processes to be viewed by Process status. If you do not specify a status value, information is generated for all status values.	EX | HC | HE | HI | HO | HR | HS | PE | WC | WR | WS | (list) **EX** (Execution)—Specifies to select Processes from the Execution queue. **HC** (Held for Call)—Specifies to select Processes submitted with hold=call. **HE** (Held due to Error)—Specifies to select Processes held due to a connection error. **HI** (Held Initially)—Specifies to select Processes submitted with hold=yes. **HO** (Held by Operator)—Specifies to select Processes held by a **change process** command issued with hold=yes. **HR** (Held Retain)—Specifies to select Processes submitted with retain=yes or retain=initial. **HS** (Held Due to Execution Suspension)—Specifies to select Processes suspended by a **flush process** command issued with hold=yes. **PE** (Pending Execution)—Specifies to select Processes submitted with a **maxdelay** parameter and assigned PE status by the Process Manager just before a Session Manager is created to execute the Process. After the Session Manager initializes, the Process is placed on the Execution queue and the status is changed to EX. **WC** (Waiting for Connection)—Specifies to select Processes that are ready for execution, but that all available connections to the remote node are in use. **WR** (Waiting for Restart)—Specifies to select Processes that are waiting for restart after session failure. **WS** (Waiting for Start Time)—Specifies to select Processes waiting for a start time. These Processes are on the Timer Queue. **list**—Specifies a list of status values. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
submitter	Locate the Processes to view by the node specification (the Connect:Direct node name) and user ID of the Process owner. The length of this parameter is unlimited.	(node specification, userid) | generic | (list) **node specification, userid—S**pecifies the node specification (the Connect:Direct node name) and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.

The following command displays the specified Process number:

```
view process pnumber=1;
```

Monitoring the Process Status on the TCQ

Use the select process command to display information about Processes in the TCQ.

The search criteria provide flexibility in selecting Processes. You can search for a Process by Process name, Process number, queue, secondary node, status, owner of the Process, or any combination of the search criteria parameters.

You also can specify more than one Process in the search criteria. You can request either a detailed report about the selected Process or a short report.

There are no required parameters for this command. If you do not specify an optional parameter, Connect:Direct selects all Processes executing or waiting for execution. Following are the optional parameters for the **select process** command:

Parameter	Description	Value
pname	Locate the Process to select by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name, up to 8 alphanumeric characters. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.
pnumber	Locate the Process to select by Process number. Connect:Direct assigns the Process number when the Process is submitted.	number from 1–99,999 | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma.
queue	Specifies the Processes to be selected by the specified queue names. The default is all.	<u>all</u> | exec | hold | wait | timer **all** selects Processes from all queues. **exec**—Selects Processes from the Execution queue. **hold**—Selects Processes from the Hold queue. **timer**—Selects Processes from the Timer queue. **wai** —Selects Processes from the Wait queue.
snode	Locate the Process by the secondary node name. If defined in the netmap.cfg file, the secondary node name can be 1–16 characters long. If not defined in the netmap.cfg file, the secondary node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
status	Specifies the Processes to be selected by Process status. If you do not specify a status value, information is generated for all status values.	EX | HC | HE | HI | HO | HR | HS | PE | WC | WR | WS | (list) **EX** (Execution)—Specifies to select Processes from the Execution queue. **HC** (Held for Call)—Specifies to select Processes submitted with hold=call. **HE** (Held due to Error)—Specifies to select Processes held due to a connection error. **HI** (Held Initially)—Specifies to select Processes submitted with hold=yes. **HO** (Held by Operator)—Specifies to select Processes held by a **change process** command issued with hold=yes. **HR** (Held Retain)—Specifies to select Processes submitted with retain=yes or retain=initial. **HS** (Held Due to Execution Suspension)—Specifies to select Processes suspended by a **flush process** command issued with hold=yes. **PE** (Pending Execution)—Specifies to select Processes submitted with a **maxdelay** parameter and assigned PE status by the Process Manager just before a Session Manager is created to execute the Process. After the Session Manager initializes, the Process is placed on the Execution queue and the status is changed to EX. **WC** (Waiting for Connection)—Specifies to select Processes that are ready for execution, but that all available connections to the remote node are in use. **WR** (Waiting for Restart)—Specifies to select Processes that are waiting for restart after session failure. **WS** (Waiting for Start Time)—Specifies to select Processes waiting for a start time. These Processes are on the Timer Queue. **list**—Specifies a list of status values. Enclose the list in parentheses, and separate each value with a comma.

Parameter	Description	Value
submitter	Locate the Processes to select by the node specification (the Connect:Direct node name) and user ID of the Process owner. The length of this parameter is unlimited.	(node specification, userid) | generic | (list) **node specification, userid—Specifies** the node specification (the Connect:Direct node name) and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.
detail	Specifies the type of report (short or detailed) that Connect:Direct generates for the selected Processes.	yes | <u>no</u> **yes**—Generates a detailed report. **no**—Generates a short report.

The following command displays a short report for the specified Process number:

```
select process pnumber=9 detail=no;
```

Output from the command is displayed in the following table:

```
===================================================================================
=
                            SELECT  PROCESS
===================================================================================
=
PROCESS NAME  NUMBER   USER    SUBMITTER NODE      QUEUE       STATUS
-----------------------------------------------------------------------------------
-
PR01          9        root    cd.unix.pj          EXEC        EX
===================================================================================
=
```

The following command displays a detailed report for the specified Process number:

```
select process pnumber=9 detail=yes;
```

Output from the command is displayed in the following table:

```
==================================================================================
=
                         SELECT  PROCESS
==================================================================================
=
Process Name    => pr01           Class         => 9
Process Number  => 9              Priority      => 8
Submitter Node  => cd.unix.pj     PNODE         => cd.unix.pj
Submitter       => sub1           SNODE         => cd.unix.pj
Retain Process  => no             Header Type   => p

Submit Time     => 19:52:35       Schedule Time =>
Submit Date     => 05/22/1996     Schedule Date =>

Queue           => EXEC
Process Status  => EX
Message Text    =>
----------------------------------------------------------------------------------
-
```

Determining the Outcome of a Process

Use the select statistics command to examine Process statistics from the Connect:Direct statistics file. The type of information in the statistics report includes copy status and execution events.

The search criteria provide flexibility in selecting information you want to display. The parameters used with the select statistics command determine search criteria and the form in which the information is presented. You can specify records to select by condition code, Process name, Process number, identification type, category, secondary node, start time, stop time, and submitter node specification and user ID.

There are no required parameters for this command. If you do not indicate a search requirement with an optional parameter, Connect:Direct selects all statistics records; however, the volume of records can be excessive. Following are parameters for the select statistics command:

Parameter	Description	Value
ccode	Selects statistics records based on the completion code operator and return code values associated with Step Termination. The condition code operator default is **eq**. You must specify the return code.	operator, nn **operator**—Specifies the completion code operator. Following are the valid completion code operators: **eq** or **=** or **==** (equal) **ge** or **>=** or **=>** (greater than or equal) **gt** or **>** (greater than) **le** or **<=** or **=<** (less than or equal) **lt** or **<** (less than) **ne** or **!=** (not equal) The return code is the exit status of the UNIX command or the Connect:Direct Process or command. nn—Specifies the return code value associated with Step Termination.
pname	Locate the statistics to select by Process name. The Process name is limited to 8 characters on Connect:Direct Windows and Connect:Direct OS/390.	name | generic | (list) **name**—Specifies the Process name, up to 8 alphanumeric characters. **generic**—Specifies a nonspecific value for the Process name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more pname strings. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma.
pnumber	Locate the statistics to select by Process number. Connect:Direct assigns the Process number when the Process is submitted.	number from 1–99,999 | (list) **number**—Specifies the Process number. **list**—Specifies a list of Process numbers. Enclose the list in parentheses, and separate each value with a comma.
reccat	Specifies whether the selection of statistics file records is based on events or related to a Process.	CAEV | CAPR | (CAEV , CAPR) **CAEV**—Specifies that the selection of statistics file records is related to an event, such as a Connect:Direct shutdown. **CAPR**—Specifies that the selection of statistics file records is related to one or more Connect:Direct Processes.

Parameter	Description	Value
recids	Specifies the statistics file records to be selected by record ID. This parameter identifies particular types of statistics records, such as a copy termination records or Connect:Direct initialization event records.	record id | (list) record id—Selects statistics file records for the specified record ID. **list**—Specifies a list of Process names. Enclose the list in parentheses, and separate each value with a comma. Following are the for valid record ID values: APSM—License Management failure generated. CHGP—The **change proces**s command issued. CRHT—Copyright statement. COAC—Listen connection enabled for either API or a remote node. CSPA—Connect:Direct Secure+ Option failure generated. CSTP—Child Process stopped. CTRC—Copy termination record. CTRM—Child Process terminated. CUKN—Child Process unknown status. CXIT—Child Process exited. DELP—The delete Process command issued. FLSP—The flush Process command issued. FMRV—Error occurred in function management. information receive operation. FMSD—Error occurred in function management. information send operation. GPRC—Error occurred while getting Process. IFED—The if statement ended. LSST—The record ID of a step on the local node. LIEX—License expired. LWEX—License expires within 14 days. NINF—Connect:Direct information generated at startup. NMOP—Network map file opened. NMPR—The network map is updated through Browser, Control Center, or KQV Interface. NUIC—Initialization complete. NUIS—Initialization started. NUTC—Termination completed. NUTS—Termination started. PERR—Process error. PFLS—Process flushed. PRED—Process ended. PRIN—Process interrupted. PSAV—Process saved. PSTR—Process started. QCEX—A Process moved from another queue to the EXEC queue.

Parameter	Description	Value
recids (*continued*)		record id (*continued*)
		QCWA—A Process moved from another queue to the WAIT queue.
		QCTI—A Process moved from another queue to the TIMER queue.
		QCHO—A Process moved from another queue to the HOLD queue.
		RJED—The run job ended.
		RNCF—Remote node connection failed.
		RSST—The record ID of a step on the remote node.
		RTED—The run task ended.
		RTSY—Run task restarted. Resyncing with run task that was executing.
		SBED—The submit ended.
		SELP—The select Process command issued.
		SELS—The select statistics command issued.
		SEND—Session ended.
		SERR—System error.
		SFSZ—Size of the file submitted.
		SGON—User signed on using KQV Interface or Command Line Interface.
		SHUD—Shutdown occurred.
		SIGC—Signal caught.
		SSTR—Session start.
		STOP—The stop command issued.
		SUBP—The submit command issued.
		TRAC—The trace command issued.
		TZDI—The time zone of the local node represented as the difference in seconds between the time at the local node and the Coordinated Universal Time.
		UNKN—Unknown command issued.
		USEC—Security check for user ID failed.
		USMG—Connect:Direct is shutting down.
		XCMM—Command manager (CMGR) messages.
		XCPR—Copy receive.
		XCPS—Copy send.
		XIPT—Communication errors.
		XLKL—Low-level TCQ record locking errors.
		XMSG—Message sent to user exit.
		XPAE—Parsing error occurred when a Process or command was submitted.
		XPAM—Parsing error occurred when a Process or command was submitted.
		XPMC—Process manager (PMGR) connection error messages.

Parameter	Description	Value
recids (*continued*)		record id (*continued*) XPML—PMGR statistics log error messages. XPMP—PMGR error messages when checking permission on the Connect:Direct programs. XPMR—PMGR RPC and miscellaneous error messages. XPMT—PMGR termination error messages. XRPM—Run task or run job error messages. XRRF—Relative record file access error messages. File structure is use for TCQ. XSMG—Session manager (SMGR) error messages. XSQF—File access error messages. XSTA—User exit program started. XTQG—A single TCQ error message group. XTQZ—A single TCQ error message group.
snode	Locate the statistics file record by the secondary node name. If defined in the netmap.cfg file, the secondary node name can be 1–16 characters long. If not defined in the netmap.cfg file, the secondary node name can be 1–256 alphanumeric characters long. You can also specify a remote node name as an IP address or hostname and a port number or port name.	remote node specification | generic | (list) **remote node specification**—Specifies the node specification of the remote node. **generic**—Specifies a nonspecific value for the remote node name. This generic value, containing pattern-matching characters, evaluates to a list of zero or more remote node names. **list**—Specifies a list of remote node specifications. Enclose the list in parentheses, and separate each value with a comma.
startt	Selects records produced both at and since the specified time. The date or day and time are positional. If you do not specify date or day, a comma must precede time.	[date | day] [, hh:mm:ss [am|pm]] **date**—Specifies the day (dd), month (mm), and year (yy), which you can code as mm/dd/yyyy or mm-dd-yyyy. If you only specify date, the time defaults to 00:00:00. The current date is the default. **day**—Specifies the day of the week. Values are today, monday, tuesday, wednesday, thursday, friday, saturday, and sunday. **hh:mm:ss [am | pm]**—Specifies the time of day in hours (hh), minutes (mm), and seconds (ss). You can specify the hour in either 12- or 24-hour format. If you use 12-hour format, then you must specify am or pm. The default is the 24-hour format. The default value is 00:00:00, which indicates midnight. If you specify only the day value, the time defaults to 00:00:00.

Parameter	Description	Value
stopt	Specifies that Connect:Direct searches for statistics records up to and including the designated date, day, and time positional parameters. If you do not specify date or day, a comma must precede time.	[date | day] [, hh:mm:ss [am|pm]] **date**—Specifies the day (dd), month (mm), and year (yy), which you can code as mm/dd/yyyy or mm-dd-yyyy. If you only specify date, the time defaults to 00:00:00. The current date is the default. **day**—Specifies the day of the week. Values are today, monday, tuesday, wednesday, thursday, friday, saturday, and sunday. **hh:mm:ss [am | pm]**—Specifies the time of day in hours (hh), minutes (mm), and seconds (ss). You can specify the hour in either 12- or 24-hour format. If you use 12-hour format, then you must specify am or pm. The default is the 24-hour format. The default value is 00:00:00, which indicates midnight. If you specify only the day value, the time defaults to 00:00:00.
submitter	Locate the statistics records to select by the node specification (the Connect:Direct node name) and user ID of the Process owner. The character length of the parameter is unlimited.	(node specification, userid) | generic | (list) **node specification, userid**—Specifies the node specification (the Connect:Direct node name) and user ID. **generic**—Specifies a nonspecific value for node specification and user ID. This generic value, containing pattern-matching characters, evaluates to a list of zero or more node specifications and user IDs. **list**—Specifies a list of node specification and user ID pairs. Enclose the list in parentheses, and separate each value with a comma.
detail	Specifies the type of report (short or detailed) that Connect:Direct generates for the selected Processes.	yes | no **yes**—Generates a detailed report. **no**—generates a short report.

The following section illustrates sample detailed and summary reports generated by the **select statistics** command.

Sample Detailed Output

The following command generates a detailed report for Process number 9:

```
select statistics pnumber=9 detail=yes startt=(08/10/2004);
```

The report consists of all records from August 10, 2004.

A sample statistics output for two steps only is listed in the following section. Use the table in the *recids* on page 52 to interpret the Record ID. The Record ID can change for each Process step displayed. The completion code indicates whether the Process executed successfully or produced an error condition.

To display the long text of the message, issue the **ndmmsg** command described in Chapter 4, *Using Connect:Direct Utilities*.

Sample Summary Output

The following command generates summary statistics for Process number 9:

```
sel stat pnumber=9 detail=no startt=(08/10/2004);
```

The report consists of all records from August 10, 2004.

Sample output that describes all Process steps in summary form is displayed in the following table:

```
==================================================================================
=
                          SELECT STATISTICS
==================================================================================
=
P RECID LOG TIME             PNAME PNUMBER STEPNAME CCOD FDBK MSGID
E RECID LOG TIME             MESSAGE TEXT
----------------------------------------------------------------------------------
-
P PSTR 08/10/2004 09:10:39   PR01  9                0         XSMG200I
P IFED 08/10/2004 09:10:44   PR01  9                0         XSMG405I
P CTRC 08/10/2004 09:10:44   PR01  9                0         XSMG405I
P IFED 08/10/2004 09:10:45   PR01  9                4         XSMG400I
P RTED 08/10/2004 09:10:45   PR01  9                0         XSMG400I
P IFED 08/10/2004 09:10:45   PR01  9                4         XSMG400I
P CTRC 08/10/2004 09:10:45   PR01  9                0         XSMG405I
P CTRC 08/10/2004 09:10:45   PR01  9                8         XSMG405I
P CTRC 08/10/2004 09:10:45   PR01  9                8         XSMG405I
==================================================================================
=
```

To avoid lengthy search times when issuing the **select statistics** command, archive or delete statistics files regularly. Execution of a Process generates multiple statistics records. Connect:Direct closes the current statistics file and creates a new statistics file every midnight. It can also close the current file before midnight if the file size exceeds the value set for the file.size initialization parameter. The default file size is 1 megabyte.

Statistics files are in the *d_dir*/work/*cd_node* directory. Names of the statistics file are in the format **Syyyymmdd.ext**, where **yyyy** indicates year, **mm** indicates month, and **dd** indicates day. The extension (**ext**) begins as 001. The extension is incremented by one each time a new statistics file created in a single day.

Running System Diagnostics

The diagnostic command, trace, enables you to run system diagnostics, to troubleshoot operational problems. Use the trace command to enable and disable runtime traces within the Process Manager, Command Manager, and Session Manager components of the software. For Session Manager traces and Command Manager traces, you can run a trace for a specific node.

The Command Manager trace affects only the Command Manager that executes the Trace command. It does not affect all Command Managers. Session Manager traces go into effect immediately. You must specify one parameter.

Following are parameters for the trace command:

Parameter	Description	Value
cmgr	To trace the Command Manager. The default file name is CMGR.TRC. If neither pnode nor snode is defined, trace is in effect for both SMGRs.	level=0 |1 | 2 | 4 file=name **level**—Specifies the level of detail displayed in the trace output. The default is 4. **0** terminates the trace. **1**—Is the basic level that provides function entry and function exit. **2**—Includes level 1 plus function arguments. **4**—Enables a full trace. Basic diagnostic information, such as values of internal data structures at key points in the execution flow, are displayed. **file**—Specifies the name of a file where the trace output is directed. If you do not specify a file name, the file is created in the Connect:Direct working directory with the file name CMGR.TRC. The length of the name value is unlimited.
comm	To trace the data sent to and received from a remote Connect:Direct system within the Session Manager. You can set this trace independently from or in conjunction with the smgr trace. If you run both the comm and smgr traces, trace output for both traces is directed to the file name of the trace last specified.	level=0 |1 | 2 | 4 file=name **level**—Specifies the level of detail displayed in the trace output. 0—Terminates the trace. 1—Is the basic level that provides function entry and function exit. **2**—Includes level **1** plus function arguments. **4—E**nables a full trace that provides basic diagnostic information, such as values of internal data structures at key points in the execution flow. **file**—Specifies the name of a file where the trace output is directed. If you do not specify a file name, the file is created in the Connect:Direct working directory with the file name COMM.TRC. The length of the name value is unlimited. The default file name is **COMM.TRC**.

Parameter	Description	Value
pmgr	To trace the Process Manager. The default file name is PMGR.TRC.	level=0 |1 | 2 | 4 file=name **level**—Specifies the level of detail displayed in the trace output. The default is 4. **0**—Terminates the trace. **1**—Is the basic level that provides function entry and function exit. **2**—Includes level 1 plus function arguments. **4**—Enables a full trace that provides basic diagnostic information, such as values of internal data structures at key points in the execution flow. **file**—Specifies the name of a file where the trace output is directed. If you do not specify a file name, the file is created in the Connect:Direct working directory with the file name PMGR.TRC. The length of the name value is unlimited.
smgr	To run the trace for Session Managers created after issuing the trace command. Currently executing Session Managers are unaffected. If you run both the comm and smgr traces, trace output for both traces is directed to the file name of the trace last specified.	level=0 |1 | 2 | 4 snode | pnode | tnode file=name **level**—Specifies the level of detail displayed in the trace output. The default is 4. **0**—Terminates the trace. **1**—Is the basic level that provides function entry and function exit. **2**—Includes level 1 plus function arguments. **4**—Enables a full trace that provides basic diagnostic information, such as values of internal data structures at key points in the execution flow. **snode**—Specifies to trace only the SNODE SMGR. **pnode**—Specifies to trace only the PNODE SMGR. **tnode**—Identifies the node on which to perform the trace. If you want to gather trace information for more than one node, identify more than one node in this parameter. **file**—Specifies the name of a file where the trace output is directed. If you do not specify a file name, the file is created in the Connect:Direct working directory with the file name SMGR.TRC. The length of the name value is unlimited. The default file name is **SMGR.TRC**.

The following sample trace commands performs a level 2 trace on the Session Manager for the node called ath3500ry and writes the output to the file Smgp.trc:

```
trace smgr pnode tnode=ath3500ry level=2 file=Smgp.trc;
```

A partial sample trace output is illustrated in the following section. A trace identifies the Process ID and the function, the month and day, and the time in microseconds. The first column contains the Process ID. Column two indicates the month and day in the form of MM/DD. Column three indicates the time in the form of HH:MM:SSSS. The last column indicates the function. An arrow pointing to the right indicates the function was entered. An arrow pointing to the left indicates the function was exited.

Some of the functions are indented, which indicates nesting. An indented arrow indicates that the function was called by the preceding function.

```
==================================================================================
=
498    05/18 15:13:0104   cm_sendcmd_1 entered.
498    05/18 15:13:0206    -> ndm_error_destroy
                         <- ndm_error_destroy: ok
498    05/18 15:13:0506    -> ndm_error_create
                         <- ndm_error_create: ok
498    05/18 15:13:0708    ndm_cmds_free entered.
                         ndm_cmds_free exited.
498    05/18 15:13:0801    ->ndm_parser_jdi
498    05/18 15:13:0806    -> ndm_error_create
                          <- ndm_error_create: ok
498    05/18 15:13:0916     ->Parser: SELPRO
498    05/18 15:13:0926      ->bldexp
                           <-bldexp: Null argument value,
                                  don't add.
498    05/18 15:13:1116      ->bldexp
498    05/18 15:13:1136       -> ndm_crit_comp
498    05/18 15:13:1155        ->compile
                             <-compile
                            <- ndm_crit_comp: Handle
                           <-bldexp: ok
                        .
                        .
                        .
==================================================================================
=
```

Chapter 3

Process Queuing

This chapter contains information about the Transmission Control Queue (TCQ), which includes the following:

- ✦ A description of the TCQ
- ✦ Connect:Direct activity scheduling
- ✦ Process progression through the TCQ
- ✦ Status values for the Wait, Execution, Hold, and Timer queues

About the Transmission Control Queue

The TCQ controls Process execution as Connect:Direct operates. After you submit a Process, it is stored in the TCQ. The TCQ consists of four queues: Execution, Wait, Timer, and Hold.

After you submit a Process, you can monitor the status, modify specific characteristics, and stop execution by using the appropriate commands. The commands listed in the following table allow you to perform these tasks:

Command	Definition
change process	Change the status and modify specific characteristics of a *nonexecuting* Process in the TCQ.
delete process	Remove a *nonexecuting* Process from the Wait, Timer, and Hold queues.
flush process	Remove an *executing* Process from the Execution queue.
select process	Monitor Processes in the TCQ and those Processes that are executing.
view process	View Processes in the TCQ.

Scheduling Connect:Direct Activity

Connect:Direct places a Process in a queue based on the parameters that affect scheduling. You can specify scheduling parameters in the **process** statement or the **submit** command.

Scheduling parameters are listed in the following section:

✦ retain=yes|no|initial

✦ hold=yes|no|call

✦ startt=[([date|day] [, hh:mm:ss | [am | pm]])

The following table shows how scheduling parameters affect the logical queues.

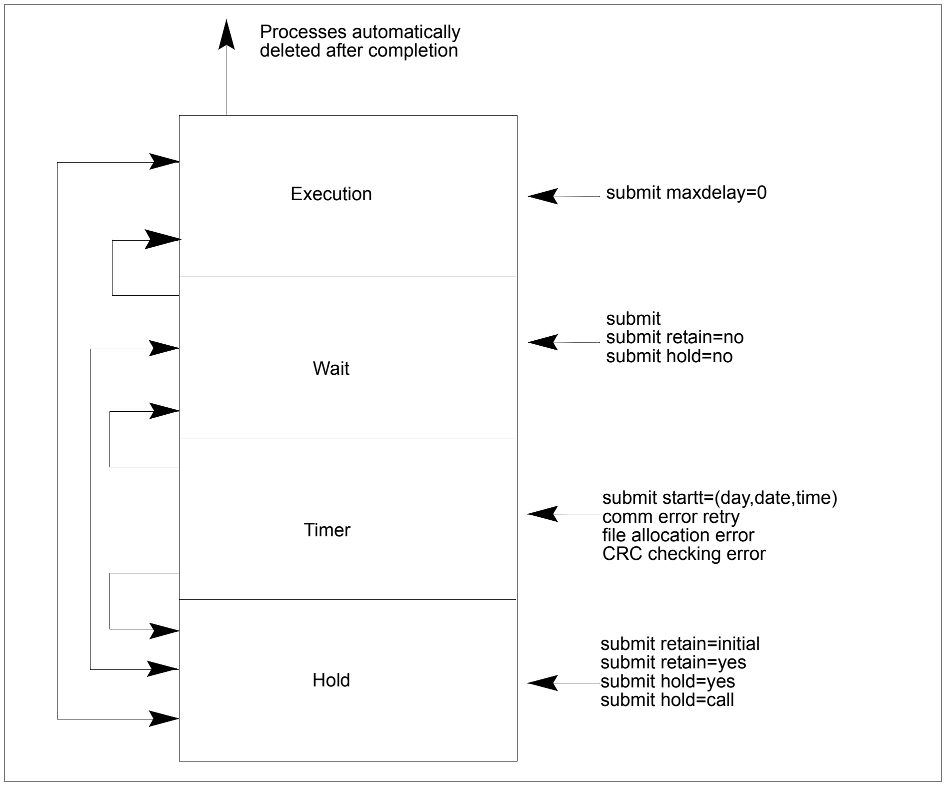
Scheduling Parameter	Queue	Comments
None of the scheduling parameters specified	Wait	The Process remains in the Wait queue until Connect:Direct establishes a session with the remote node. After a session is established, the Process moves to the Execution queue.
retain=yes	Hold	A copy of the Process executes once, unless you specify a startt parameter value. Specify a day or time or both for the Process to start.
retain=no	Wait (if no other parameters are specified)	The Process remains in the Wait queue until Connect:Direct establishes a session with the remote node. The default is no.
retain=initial	Hold	A copy of the Process remains in the Hold queue and executes every time the Process Manager is initiated.
retain=yes and hold=no or hold=call	Hold	A copy of the Process remains in the Hold queue to be executed when released.
hold=yes	Hold	You can execute the Process by specifying the change process command with the release parameter.
hold=no	Wait (if no other parameters are specified)	The default for hold is no.
hold=call	Hold	The Process remains in the queue until the remote node starts a session with the local node or another Process starts a session with that remote node.
startt	Timer	When the scheduled day or time occur, the Process is moved to the Wait queue.

Each Process in the TCQ has an associated status value. Each status value has a unique meaning that is affected by the logical queue in which the Process is placed. Status values for each queue are shown in the tables in the following sections. You can use the select process command to examine that status of Processes in the TCQ. For example, the following command displays all Processes in the TCQ with execution status:

```
select process status=EX;
```

Progression of a Process Through the TCQ

This section describes each logical queue of the TCQ and the progression of a Process through these queues. The following figure illustrates the four logical queues and their associated parameter values.



About the Execution Queue

Processes are placed in the Execution queue after Connect:Direct connects to the remote node. Processes normally come from the Wait queue, but also can be placed in the Execution queue by a **submit** command with maxdelay=0 specified.

Processes in the Execution queue can be in execution (EX) status or pending execution (PE) status. Processes with EX status are exchanging data between two Connect:Direct nodes. Processes with

PE status are waiting for Process start messages to be exchanged between the local node and the remote node. Processes usually have PE status assigned for a very short period of time.

After a Process successfully completes, it is automatically deleted from the Execution queue. A **flush process** command with hold=yes moves a Process from the Execution queue and places it in the Hold queue. When a session is interrupted, the Process moves from the Execution queue to the Timer queue if retry values are specified. If connection is not made before the retry values are exhausted or if retry values are not specified, the Process moves to the Hold queue.

The flush process can only per performed from the Pnode.

The following table shows the status values for the Execution queue:

Status	Comment
PE	Pending Execution is the initial queue status when a Process is submitted with maxdelay=0.
EX	Execution status indicates that the Process is executing.

About the Wait Queue

Processes in the Wait queue are waiting for a new or existing connection to become available between the local node and the remote node.

Processes can come from the Hold queue or the Timer queue. Processes also can be placed in the Wait queue by a submit command with no parameters specified, submit with retain=no, or submit with hold=no.

After the connection is made, Processes automatically move to the Execution queue.

The following table shows the status values for the Wait queue:

Status	Comment
WC	This status indicates the Process is ready to execute as soon as possible, but no session is available. Other Processes may be executing with the SNODE, and no other sessions are available. This Process runs as soon as a new session is created or an existing session becomes available.
WR	This status indicates that the Process is in retry status. The number of retries and intervals between retries is specified in the network map.
WA	This status indicates the initial queue status when a Process is submitted without a hold or retain value. This Process is ready to execute as soon as possible.
WS	This status indicates that the Process is waiting for the PNODE to continue the session.

About the Timer Queue

Processes are placed in the Timer queue by a submit command with the startt parameter specified. Processes in the Wait for Start Time (WS) status are waiting for the start time to arrive before moving to the Wait queue. Processes also are placed in the Timer queue in Retry (WC) status if one of the following error conditions occur:

- ✦ If a file allocation error occurs when a Process is executing on either the local or the remote node, and the file allocation error is identified as a condition to retry, the Process is placed in the Timer queue. The Process is then retried using the short-term and long-term retry parameter definitions. This capability enables a Process that was unable to execute because a file that it called was unavailable to be retried at a later time.
- ✦ If a connection error occurs while a Process is executing, the intelligent session retry facility places all Processes scheduled for the node, including the executing Process, in the Timer queue. This capability eliminates the overhead required to retry each of the Processes on the node even though the connection is lost.
- ✦ If CRC checking is activated, a Process that generates a CRC error is placed in the Timer queue.

Connect:Direct automatically tries to execute the Process again based on the number of times to retry and the delay between retries as specified in the submit command, process statement, network map parameters, or initialization parameters.

Processes move from the Timer queue to the Wait queue. A change process command with hold=yes specified moves the specified Process from the Timer queue to the Hold queue. The following table shows the status values for the Timer queue:

Status	Comment
WR	Retry indicates that the Process is in retry status. The number of retries and intervals between retries is specified in the network map.
WS	Wait for Start Time status indicates that the Process was submitted with a start time or date that has not expired. When startt is reached, the Process is placed in the Wait queue for scheduling for execution.
HR	Held Retain indicates that the Process was submitted with retain=yes or retain=initial specified and has already executed. The Process can be released later by a change process command with release specified.
WC	This status indicates the Process is ready to execute as soon as possible, but no session is available. Other Processes may be executing with the SNODE, and no other sessions are available. This Process runs as soon as a new session is created or an existing session becomes available.

About the Hold Queue

Processes in the Hold queue are waiting for operator intervention before they progress to the Wait queue. This queue enables operators of the local node and remote node to coordinate and control Process execution.

Processes are placed in the Hold queue by a submit command with retain=initial, retain=yes, or hold=yes parameters specified. Processes submitted with hold=call also are placed in the Hold queue. Processes are moved from the Timer queue to the Hold queue by a change process command with hold=yes specified. Additionally, Processes are moved from the Execution queue to the Hold queue by a flush process command with hold=yes specified.

Processes are moved from the Hold queue to the Execution queue by a change process command with the **release** parameter specified.

The following table shows the status values for the Hold queue:

Status	**Comment**
HC	Held for Call indicates that the Process was submitted with hold=call specified. A session started from either node causes the Process to be moved to the Wait queue in WC status. The Process is placed in the Execution queue when the Process is selected for execution.
HI	Held Initially indicates that the Process was submitted with hold=yes specified. The Process can be released later by a change process command with release or hold=no specified.
HE	Held due to error specifies that a session error or other abnormal condition occurred.
HO	Held by Operator indicates that a change process hold=yes was specified.
HR	Held Retain indicates that the Process was submitted with retain=yes or retain=initial specified and has already executed. The Process can be released later by a change process command with release specified.
HS	Held for Suspension indicates that the operator issued a flush process command with hold=yes specified. The Process can be released later by a change process command with release specified.

Chapter 4

Using Connect:Direct Utilities

This chapter contains instructions for using the translation table utility and the message utility.

The Translation Table Utility

Connect:Direct translates data from one character set code to a different character set code, such as from ASCII to EBCDIC, based on a character translation table in the *d_dir*/ndm/xlate directory. Connect:Direct provides a default character translation table for use during file transfer operations or you can modify this table using the utility program called ndmxlt.

To create a translation table, either copy the file called /*cd_dir*/cdunix/ndm/src/def_send.sxlt or /*cd_dir*/cdunix/ndm/src/def_recv.sxlt, where *cd_dir* is the directory where Connect:Direct for UNIX is installed, and rename it or modify this file. Use a text editor to add the new values to the table in the file you created. Compile the updated file with the ndmxlt utility. Replace the default translation table in the *d_dir*/ndm/xlate with the updated table. Each table is 256 bytes long.

Following is a sample translation table:

```
# This file contains an example of defining an ASCII-to-EBCDIC translation table and
# then changing it to translate lowercase to uppercase.
#
# Define the ASCII-to-EBCDIC table.
offset=0
00 01 02 03 04 05 06 07 08 05 15 0B 0C 0D 0E 0F
10 11 12 13 3C 15 16 17 18 19 1A 1B 1C 1D 1E 1F
40 5A 7F 7B 5B 6C 50 7D 4D 5D 5C 4E 6B 60 4B 61
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 7A 5E 4C 7E 6E 6F
7C C1 C2 C3 C4 C5 C6 C7 C8 C9 D1 D2 D3 D4 D5 D6
D7 D8 D9 E2 E3 E4 E5 E6 E7 E8 E9 AD E0 BD 5F 6D
79 81 82 83 84 85 86 87 88 89 91 92 93 94 95 96
97 98 99 A2 A3 A4 A5 A6 A7 A8 A9 C0 4F D0 A1 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
#
# Change the lowercase characters to uppercase.
offset=61
C1 C2 C3 C4 C5 C6 C7 C8 C9 D1 D2 D3 D4 D5 D6 D7
D8 D9 E2 E3 E4 E5 E6 E7 E8 E9
```

Each byte stores the character value for the target character set. The source character set is used as an index into the table. For example, an ASCII blank (Hex 20) would locate the byte at offset Hex 20 in the translation table. If the byte at location Hex 20 contains Hex code 40, that would translate to an EBCDIC code indicating a blank character.

Creating and Modifying a Translation Table

You can create or modify a translation table tailored to your requirements with the **ndmxlt** utility program. To invoke the **ndmxlt** utility, type the following command at the UNIX prompt:

```
$ ndmxlt -ssourcefile -ooutputfile [ -rradix] [ -ffiller] -mxlatefile
```

The parameters for the **ndmxlt** command are listed in the following section:

Parameter	Description	Values
-ssourcefile	The path and file name of the translation table source file. If no value is specified, input is read from STDIN.	Path and name of translation table
-ooutputfile	The path and file name of the translation table output file.	Path and name of translation output file

Parameter	Description	Values

-rradix	The radix or base of the source file input data. All numeric values whether from command line options or input data are interpreted based on the radix setting.	Three possible values are:
		x = hexadecimal
		d = decimal
		o = octal
		The default is x.

-ffiller	A filler byte value. The entire table is initialized to this value before the input data is scanned and applied to the table.	
-m	The path and file name of a model translation table. If specified, the model table is read in and then the input data is scanned and applied to the table. This capability permits creating a number of different tables that are variations from a single base table without having to specify all 256 bytes of input data for each table.	Path and file name of the model translation table

This section provides examples of how to create a translation table or modify an existing translation table.

Example—Creating a Translation Table

Perform the following steps to create a sample translation table that changes lowercase characters to uppercase characters:

1. Make a copy of the sample translation table located at *cd_dir/*ndm/src/def_send.sxlt.
2. Open the new translation table with a text editor.
3. Add the following lines to the bottom of the table. It should look like the table on page 4-1 when you have added this information.

```
#
# Change the lowercase characters to uppercase.
offset=61
C1 C2 C3 C4 C5 C6 C7 C8 C9 D1 D2 D3 D4 D5 D6 D7
D8 D9 E2 E3 E4 E5 E6 E7 E8 E9
```

4. Copy the modified file to *cd_dir*/ndm/src and name it UpperCaseEBC.sxlt.
5. Compile the new translation table using the following syntax:

```
ndmxlt -s../src/UpperCaseEBC.sxlt -oUpperCaseEBC.xlt
```

6. To use this translation table, add the following sysopts parameter to the copy statement:

```
copy from file=filename
     to   file=filename
          sysopts=":xlate.tbl=pathname/UpperCaseEBC.xlt:"
```

Example—Modifying a Model Translation Table

Perform the following steps to modify a model translation table. This method, when implemented, reads the model table and writes it to a new file. It then reads the input data and makes changes to the table created.

1. Create a file called FourLinesUpperCase.sxlt and add the following lines to the file:

```
#
# Change the lowercase characters to uppercase.
offset=61
C1 C2 C3 C4 C5 C6 C7 C8 C9 D1 D2 D3 D4 D5 D6 D7
D8 D9 E2 E3 E4 E5 E6 E7 E8 E9
```

2. Copy the modified file to *cd_dir*/ndm/src.
3. Type the following command to compile this file and create a translation table called fourLineUpperCase.xlt:

```
ndmxlt -s../src/FourLineUpperCase.sxlt -oFourLineUpperCase.xlt -mdef_send.xlt
```

4. To use this translation table, add the following sysopts parameter to the copy statement:

```
copy from file=filename
     to   file=filename
          sysopts=":xlate.tbl=pathname/FourLineUpperCase.xlt:"
```

Using Translation During File Transfer Operations

Translation is specified in the copy statement of a Connect:Direct Process. You can use the default translation table or create a new table. To use the default translation table, type the following copy statement:

```
copy from file=abc to file=xyz sysopts=":xlate=yes:"
```

To specify a customized table for data translation, include the following sysopts subparameter in the copy statement, where *pathname/filename* identifies the translation table:

```
copy from file=filename
     to   file=filename
          sysopts=":xlate.tbl=pathname/filename:"
```

Refer to the UNIX section of the *Connect:Direct Process Guide* for additional details concerning translation table specification with a copy statement.

Diagnostics

The following table displays the error messages that are generated by ndmxlt:

Diagnostic Number	Description
XXLT001I	Invalid directive
XXLT002I	Input file open error
XXLT003I	Model file open error
XXLT004I	Invalid filler value
XXLT005I	Invalid offset value
XXLT006I	Invalid radix value
XXLT007I	Invalid table value
XXLT008I	Table data out of bounds

Using and Modifying the Message File

The Connect:Direct message file contains records with text for all messages, including errors and messages from Connect:Direct servers other than the host server. You can add and delete message records with a text editor. The message file resides in *d_dir*/ndm/cfg/*cd_node*/msgfile.cfg. You can display message text with the **ndmmsg** command.

About the Message File Content

The message file is structured much the same way as other Connect:Direct configuration files. Each record is a logical line in a text file that consists of one or more physical lines. Each record has a unique name, a message ID, and fields that make up the message text.

The message record definitions provide for symbolic substitution, which permits including actual file names or other variable information within the text to more specifically identify a problem. Symbolic variables begin with the ampersand character (&).

The format of Connect:Direct message IDs is listed in the following table:

```
XxxxnnnI

Where:

X Indicates Connect:Direct
xxx is a 3-character Connect:Direct component identifier
nnn is a 3-digit decimal number
I is the standard, though not required, suffix
```

Understanding the Message File Record Format

The following table shows the format of the message file record. Each record can be up to 4K bytes long. Optional parameters and values are in brackets.

```
message id [long.text detailed message explanation] [mod.name issuing module name] short.text message summary
```

Following are the parameters for the message file record:

Parameter	Description	Values
long.text	A string that explains the message in detail.	a text string
mod.name	The name of the source module issuing the message ID.	source module name
short.text	A summary of the message. This field is required.	summary message, up to 72 characters

The following table illustrates a sample message record for XCPS008I.

```
XCPS008I:\ :mod.name=NUSMCP00.C:\
:short.text=File is not VB datatype.:\
:long.text=File is not variable block. Change sysopts datatype to\
either binary or text to transfer this file.\
\nSYSTEM ACTION-> the copy step failed and CD processing\
continued with the next process step.\
\nRESPONSE-> change the sysopts datatype to either\
binary or text.:\
```

Displaying Message Text

Use the **ndmmsg** command to display text in the message file. You can display both short and long text.

The following command illustrates the format for ndmmsg:

```
ndmmsg -f msgfname [-l | -s] msgid1 [msgid2 [msgid3 [...]]]
```

Following are the parameters for the **ndmmsg** command. If you do not specify an **l** or **s** parameter, both short and long text are displayed.

Parameter	Description
-f	Specifies the name of the message file.
-l	Displays the long text of a message.
-s	Displays the short text of a message.

Following is a sample ndmmsg command:

```
ndmmsg -f /usr/ndmunix/msgfile.cfg und012i
```

Output from the command is displayed in the following table:

```
rc=&rc
fdbk=&fdbk
mod.name=NUCMRG00.C
func.name=ndmapi_sendcmd
short.text=CMGR RPC call returns NULL
long.text=The ndmapi_sendcmd RPC call made by the API to the CMGR returns a NULL
pointer.  There is probably an RPC error.
ndm.action=None
user.action=First, check if the ndmcmgr is still running; it could have been killed
accidently.  If so, then abort the current CLI and restart the CLI.  If the same
problem occurs again, try to increase the value of wait time (if set) in the API
configuration file (ndmapi.cfg).
```

Using License Key File Notification

The Server (PMGR) checks for a new license.key file every 30 seconds. If the server is busy because of high cpu usage, you can run the apnotify command to inform the Connect:Direct server that a new license file should be checked immediately. Following is the format for the apnotify command:

```
apnotify [ -t timeout ]  [ -f  ]
```

Following are the parameters for the apnotify command:

Parameter	Description
-t	Set this timeout value to any positive single digit number to wait for a server response.
-f	Request for an immediate response.

The following section illustrates sample output for the apnotify command:

```
#apnotify
License file update: new license file accepted.
```

```
#apnotify -t
Response from Connect:Direct PMGR timed out.
Suggestion: Try again with a higher timeout.
```

```
#apnotify
License file update: license file last updated on Fri Aug 6 08:46:28 2003
No changes to the license file since the last update were detected.
A license file reread can be forced by doing an "apnotify -f".
```

Chapter 5

Writing Custom Programs

The Connect:Direct Application Programming Interface (API) allows you to write custom programs in either C or C++ to use with Connect:Direct. With the C functions or the C++ classes, you can create programs to perform the following tasks:

- ✦ Establish a connection to the Connect:Direct server
- ✦ Send commands to the server
- ✦ Receive command responses from the server
- ✦ Extract data from the command responses
- ✦ Disconnect from the server

This chapter describes the format of the Connect:Direct API functions and classes and provides samples of their use. Sample programs are provided that use the Connect:Direct API functions and classes to issue commands and receive responses from the Connect:Direct server.

Compiling Custom Programs

After you write a custom program, you must compile it, using a C or C++ compiler. Refer to the following table to determine what minimum C++ compiler version to use for each platform:

Platform	C++ Compiler
Compaq	Compaq C++ V6.3-002 for Compaq Tru64 UNIX V5.1
AIX	IBM VisualAge C++ Professional for AIX, Version 6.0
Sun Solaris	SPARC/x86 Sun WorkShop 6 C++ 5.3
HP	HP ANSI C++ A.03.55
HP-Itanium	aC++/ANSI C A.06.00
Linux	c++ version 3.23

Use the commands defined in the following table to compile a custom C++ program:

Platform	C++ Compile Command
Compaq	cxx -x cxx -D_GNU_SOURCE -I../include -o sdksample sdksample.C ../lib/ndmapi.a
AIX	xlC -qinline -I../include -+ -o sdksample sdksample.C ../lib/ndmapi.a -lbsd -ldl -lsrc
Sun	CC -DBSD_COMP -I../include -o sdksample sdksample.C ../lib/ndmapi.a -L/usr/ucblib -L/usr/lib -lsocket -lrpcsoc -lnsl -lelf -ldl **Note:** If /usr/ucblib is not in the LD_LIBRARY_PATH variable, add - R/usr/ucblib to the compile command.
HP	aCC -I../include -o sdksample sdksample.C ../lib/ndmapi.a -lrpcsoc -lnsl -ldld -Wl,+s
HP-Itanium	aCC -I../include -o sdksample sdksample.C ../lib/ndmapi.a -lrpcsoc -lnsl -ldld -Wl,+s -lunwind
Linux	c++ -I../include -O -DLINUX -o sdksample sdksample.C ../lib/ndmapi.a -ldl -lnss_nis -lstdc++

To build a C program, such as the apicheck.c sample program, replace the sdksample.C parameter with the name of the C program and rename the output file parameter, -o sdksample, to the name of the output file you want to create such as apicheck.

Use the commands defined in the following table to compile a C program:

Platform	C Compile Command
Compaq	cc -D_GNU_SOURCE -I../include -o apicheck apicheck.c ../lib/ndmapi.a -lcxx
AIX	cc -I../include -+ -o apicheck apicheck.c ../lib/ndmapi.a -lbsd -ldl -lsrc -lC
Sun	cc -DBSD_COMP -I../include -o apicheck apicheck.c ../lib/ndmapi.a -L/usr/ucblib -L/usr/lib -lsocket -lrpcsoc -lnsl -lelf -ldl -lCrun
HP	cc -I../include -o apicheck apicheck.c ../lib/ndmapi.a -lrpcsoc -lnsl -ldld -Wl,+s -lCsup
HP-Itanium	cc -I../include -o apicheck apicheck.c ../lib/ndmapi.a -lrpcsoc -lnsl -ldld -Wl,+s -lCsup -lunwind
Linux	cc -I../include -O -DLINUX -o apicheck apicheck.c ../lib/ndmapi.a -ldl -lnss_nis -lstdc++

Writing Custom C Programs

If you write a custom program using the C API calls, you must include the header file ndmapi.h and link it with ndmapi.a. A sample program called apicheck.c is provided.

If you are creating a Java program, you must call the shared library ndmapi.so. If you are creating a Java program on an HP platform, you must call the shared library namapi.sl. The C API functions can then be called using JNI through the Java program.

Note: The environment variable NDMAPICFG must be set to the pathname of the client configuration file. Refer to *Starting CLI* on page 23 for instructions on setting the environment variable.

Use the following Connect:Direct API functions for C and C++ programs:

C++ Function	C Function	Description
ndmapi_connect()	ndmapi_connect_c()	Establishes a connection with the server. Specify the node to connect to in the ndm_nodespec pointer or in the CLI/API Configuration Information file. If the call is successful, NDM_NO_ERROR is returned. Control returns to the application when the connection is established and is ready for the first API request.
ndmapi_sendcmd()	ndmapi_sendcmd_c()	Sends commands to Connect:Direct. You must provide the command text. The resp_moreflag is a pointer to the flag indicating that more responses are pending for the executed command. Invoke ndmapi_recvresp_c() for C programs or ndmapi_recvresp() for C++ programs to retrieve the extra responses. Only the select process and select statistics commands require the use of ndmapi_recvresp_c() for use with C and ndmapi_recvresp() for use with C++.
ndmapi_recvresp()	ndmapi_recvresp_c()	Receives responses to commands sent to Connect:Direct. The contents of the response buffer are returned.
ndmapi_disconnect()	ndmapi_disconnect_c()	Terminates the API connection.

Three types of Connect:Direct command responses are returned by these functions.

- Informational responses return information about the submitted command.
- Data responses, stored in the resp_buffer, contain data records.
- Error responses return ERROR_H, a pointer to a linked list of all errors found. The ID field values are fixed for use when debugging. The msgid, feedback, and rc fields are specified by Connect:Direct and are referred to in message text. The subst field points to a string that contains substitution variable information to be inserted appropriately in the message text. The error control structure keeps track of the current and total number of errors. You can move through the errors by using the next pointer in error entry blocks.

The following code defines the ERROR_H structure:

```
#define NDM_ERR_ENT_T struct NDM_ERR_ENT_S
#define NDM_ERR_ENT_H NDM_ERR_ENT_T *
#define NDM_ERR_CTL_T struct NDM_ERR_CTL_S
#define ERROR_H       NDM_ERR_CTL_T *
struct NDM_ERR_ENT_S
{
    int32    id;
    char     msgid[MSGIDLEN];
    int32    feedback;
    int32    rc;
    char     *subst;
    NDM_ERR_ENT_H next;
};
struct NDM_ERR_CTL_S
{
    int32    id;
    int32    cur_entry;
    int32    num_entries;
    NDM_ERR_ENT_H next;
};
```

Creating a Connection to Connect:Direct Using ndmapi_connect() or ndmapi_connect_c()

Use **ndmapi_connect()** or **ndmapi_connect_c()** to create a connection to Connect:Direct so that an application can send commands and receive responses from the commands. Control returns to the application when the connection is established and Connect:Direct is ready for the first API request or when an error condition is set.

Following is the format for the **ndmapi_connect()** or **ndmapi_connect_c()** function:

```
int32 ndmapi_connect ERROR_H error, char *  ndm_hostname, char *  ndm_portname
```

The following table describes the parameters for the **ndmapi_connect()** or **ndmapi_connect_c()** function:

Parameter	Description	Value
error	A pointer to a Connect:Direct-defined structure that contains error information or status information.	Pointer
ndm_hostname	A pointer to the text specification of the Connect:Direct host to which the connection is made. If this parameter is not specified, the host name is determined by first checking the environment variable TCPHOST. If no value is specified, the tcp.host.name field in the CLI/API configuration file is checked. If no value is specified, the **gethostbyname()** command is invoked and the default value of **ndmhost** is used.	Null-terminated string

Parameter	Description	Value
ndm_portname	A pointer to the host port number. If this parameter is not specified, the environment variable TCPPORT is checked. If no value is specified, the value of the tcp.port in the CLI/API configuration file is checked. If no value is specified, the default value **1363** is used.	Pointer

Following are the return codes for the **ndmapi_connect()** or **ndmapi_connect_c()** function:

Return Code	Description
NDM_NO_ERROR	A session was established with the server.
NDM_ERROR	A session was not established with the server. Consult the error structure for detailed error status.

The following sample function illustrates the use of **ndmapi_connect()** to connect to the sun1 host:

```
rc=ndmapi_connect (error, "sun1", "3122");
```

Terminating a Connection Using ndmapi_disconnect() or ndmapi_disconnect_c()

Use **ndmapi_disconnect()** or **ndmapi_disconnect_c()** to terminate a connection to Connect:Direct that was established by a call to **ndmapi_connect()** or **ndmapi_connect_c()**. Following is the format for **ndmapi_disconnect()** or **ndmapi_disconnect_c()**:

```
void ndmapi_disconnect
```

There are no parameters and no return codes for **ndmapi_disconnect()** or **ndmapi_disconnect_c()**. Following is a sample **ndmapi_disconnect()** function:

```
ndmapi_disconnect ();
```

Receiving Responses Using ndmapi_recvresp() or ndmapi_recvresp_c()

Use **ndmapi_recvresp()** or **ndmapi_recvresp_c()** to receive responses that are associated with a previous command sent from the application. Following is the format for **ndmapi_recvresp()** or **ndmapi_recvresp_c()**:

```
int32 ndmapi_recvresp ERROR_H  error int32 *  resp_length, char *   resp_buffer, int32 * resp_moreflag
```

Following are the parameters for **ndmapi_recvresp()** or **ndmapi_recvresp_c()**:

Parameter	Description	Value
error	A pointer to a Connect:Direct-defined structure that contains error information or status information.	Pointer
resp_length	A pointer to the length, in bytes, of the application buffer to receive the response. The API sets this parameter to the number of bytes returned.	Pointer to number of bytes returned or 0 if you no longer want to receive responses. Setting this field to zero purges all queued responses.

resp_buffer	A pointer to the application buffer that receives the command or submit response. This buffer should allocate 4096 bytes.
	The format of **resp_buffer** is a free-form text record structure. Field names are four characters long and all uppercase. The data can be any length and can include blanks. The structure is:
	`field name=data | field name=data |...`
	For example:
	`SUBM = username | PNUM = 12 | PNAM = proc1 |...`
		A local buffer, with a size greater than or equal to that set by **r**esp_length and filled in by **ndmapi_recvresp()** or **ndmapi_recvresp_c()**.
		The CLI passes the resp_buffer to AWK for parsing. Valid values include:
		ADMN—Connect:Direct administrator name
		ADPH—Connect:Direct administrator phone number
		CCOD—Completion code
		CKPT—Checkpoint
		CLAS—Class
		DBYW—Bytes written
		DBYX—Bytes received
		DCOD—Destination completion code
		DDAY—Submit date
		DDS1—Destination disposition 1
		DDS2—Destination disposition 2
		DDS3—Destination disposition 3
		DESC—Connect:Direct administrator description
		DFIL—Destination file
		DMSG—Destination message ID
		DNVL—Destination number of volumes
		DRCW—Records written
		DRUX—RUs received
		DVOL—Destination volume array
		ECMP—Extended compression ON or OFF

Parameter	Description	Value
resp_buffer (*continued*)		Valid values (continued): FROM—Copy sending node LCCD—Local completion code LCLP—Local IP address and port number LKFL—Link fail LMSG—Local message ID LNOD—Local node MSGI—Message ID MSGT—Message text MSST—Short text OCCD—Other completion code OERR—Other node in error OMSG—Other message ID PACC—PNODE account PFIL—Process file PNAM—Process name PNOD—PNODE PNUM—Process number PPMN—PDS member name PRTY—Priority QUEU—Queue RECC—Record category RECI—Record ID RETA—Retain Process RMTP—Remote IP address and port number RSTR—Process restarted RUSZ—RU Size SACC—SNODE account SBID—Submitter node ID SBND—Submitter node name SBYR—Bytes read SBYX—Bytes sent SCMP—Standard compression SCOD— Source completion code SDDY—Schedule date SDS1—Source disposition 1 SDS2—Source disposition 1 SDS2—Source disposition 2 SDS3—Source disposition 3 SELA—Elapsed time of the event SFIL—Source file SMSG—Source message ID SNAM—Step name

Parameter	Description	Value
resp_buffer (*continued)*		Valid values (continued): SSTA—Start time of the event STAR—Start log date/time for record STAT—Process status SNOD—SNODE SNVL—Source number of volumes SOPT—SYSOPTS record SRCR—Records read SRUX—RUs sent STIM—Schedule time STOP—Stop time of the event SUBM—Submitter ID SUBN—Submitter node SUMM—Summary output selector SVOL—Source volume array TIME—Submit time XLAT—Translation
resp_moreflag	Indicates that more **ndmapi_recvresp()** or **ndmapi_recvresp_c()** calls must be issued for more information. This flag occurs only on select process and select statistics commands.	None

Following are the return codes for **ndmapi_recvresp()** or **ndmapi_recvresp_c()**:

Return Code	Description
NDM_NO_ERROR	The function completed successfully.
NDM_ERROR	An error occurred. Consult the error structure for detailed error status.
TRUNCATED	Data is truncated because the receiving buffer is too small.

Following is a sample **ndmapi_recvresp()** function:

```
int32 rc, resp_length;
int32 resp_moreflag;
char resp_buffer[makbuf];

rc= ndmapi_recvresp (error,
                  &resp_length,
                   resp_buffer,
                  &resp_moreflag
                 );
```

Sending a Command to Connect:Direct Using ndmapi_sendcmd() or ndmapi_sendcmd_c()

Use **ndmapi_sendcmd()** or **ndmapi_sendcmd_c()** to allow a command to be sent to a Connect:Direct application. Following is the format of **ndmapi_sendcmd()** or **ndmapi_sendcmd_c()**:

```
int32  rc, resp_moreflag;
struct sendcmd_data ret_data;
rc=ndmapi_sendcmd (error,
                   "select process pnumber=2,",
                   &resp_moreflag,
                   &ret_data
                    );
```

Following are the parameters for **ndmapi_sendcmd()** or **ndmapi_sendcmd_c()**:

Parameter	Description	Value
error	A pointer to a Connect:Direct-defined structure that contains error information or status information.	Pointer
cmd_text	A pointer to the null-terminated text string that specifies the command to send to Connect:Direct. The command text must be followed by a semicolon and terminated with a null. When you use the **submit=filename** command from the API, ensure that you allocate enough storage for the Process text. The text of the Process submitted is returned in the text string associated with this parameter when the function completes. If you do not allocate enough storage for the Process text, a core dump can result.	Pointer to a text string
resp_moreflag	A pointer to the flag that indicates that more responses are pending for the command just executed. Invoke **ndmapi_recvresp()** or **ndmapi_recvresp_c()** to retrieve the extra responses.	Pointer to a flag

ret_data	A pointer to a structure containing internal response information for a command. The structure is:
	`struct sendcmd_data  {`
	`     char  * cmd_name;`
	`     ulong cmd_id;`
	`     long data1;`
	`     long data2;`
	`     long data3;`
	`};`
		Pointer to a structure

sendcmd_data	Provides the caller with some information about the user request. Because parsing of command text occurs at the CMGR, the End User Application (EUA) has no way to identify the command that was submitted, unless it generated the text.	Information about the user request

Parameter	Description	Value
cmd_name	A pointer to a string with the name of the command submitted. The CLI uses this pointer to display completion messages. This field enables you to display unique completion messages without any knowledge of a specific command in the EUA.	Pointer to name of command

cmd_id	A four-byte identifier of the command that was found in the command text. Following are the four-byte identifiers:
	`/*************Command IDs******************/`
	`#define CHANGE_PROCESS 0x43484750  /* "CHGP" */`
	`#define DELETE_PROCESS 0x44454c50  /* "DELP"*/`
	`#define FLUSH_PROCESS 0x464c5350  /* "FLSP" */`
	`#define SELECT_PROCESS 0x53454c50  /* "SELP"*/`
	`#define SELECT_STATISTICS 0x53454c53  /* "SELS" */`
	`#define SUBMIT     0x5355424d  /* "SUBM" */`
	`#define TRACE_API  0x41504920  /* "API " */`
	`#define TRACE_CMGR 0x434d4752  /* "CMGR" */`
	`#define TRACE_SMGR 0x534d4752  /* "SMGR" */`
	`#define TRACE_PMGR 0x504d4752  /* "PMGR" */`
	`#define TRACE_COM  0x434f4d4d  /* "COMM"*/`
	`#define TRACE      0x54524143  /* "TRAC" */`
	`#define STOPNDM    0x53544F50   /* "STOP" */`
	
	The CLI uses these identifiers to ensure that rules are being followed. For instance, if an **ndmapi_sendcmd** returns with the resp_moreflag set and the **cmd_id** is not **SELECT_STATISTICS** or **SELECT_PROCESS**, the CLI generates an error.
		Four-byte identifier

data1, data2, and data3	For future expansion. data1 is used with the **submit** command to return the Process number.	

Following are the return codes for **ndmapi_sendcmd()** or **ndmapi_sendcmd_c()**:

Return Code	Description
NDM_NO_ERROR or Process Number	The function completed successfully.
NDM_ERROR	An error occurred. Consult the error structure for detailed error status.

Following is a sample **ndmapi_sendcmd()** function:

```
int32  rc, resp_moreflag;
struct sendcmd_data ret_data;
rc=ndmapi_sendcmd (error,
                   "select process pnumber=2 ;",
                   &resp_moreflag,
                   &ret_data
                    );
```

Writing Custom C++ Programs

If you write a custom program using the C++ API calls, you must include the class called ConnectDirectSession. The calling program must instantiate ConnectDirectSession and call the send and receive functions. A sample program called sdksample.C is provided. To write a custom C++ program, create a ConnectDirectSession class. The class contains the ConnectDirectSession interface and a constructor and destructor call to allocate and release the storage associated with the class. This class is the interface to the Connect:Direct methods and provides connection, command, data retrieval, and error services. Each method returns either CD_SUCCESS or CD_FAILURE.

Note: The environment variable NDMAPICFG must be set to the pathname of the client configuration file. Refer to *Starting CLI* on page 23 for instructions on setting the environment variable.

To use the ConnectDirectSession class, your application must include the cdunxsdk.h header file provided in the installation and must link with the ndmapi.a file. Following is a sample ConnectDirectSession class program:

```
#include "cdunxsdk.h"
#include <iostream.h>
#include <string.h>
void getError(ConnectDirectSession& cdSess);

main()
{
    ConnectDirectSession cdSess;
    char processText[16384];

    if (cdSess.SessionINF->Connect() == CD_SUCCESS)
    {
        strcpy(processText,"submit maxdelay=unlimited sdksample process snode=SNODENAME
");
        strcat(processText,"copy00 copy from (file=sample.txt pnode)");
        strcat(processText,"               to (file=sample.000 snode disp=rpl) ;");

        if (cdSess.SessionINF->SendCommand(processText) == CD_SUCCESS)
        {
        printf("%s completed, pnumber = %ld.\n",
            cdSess.SessionINF->GetCommandName(),
           cdSess.SessionINF->GetProcessNumber());
sprintf(processText, "SELECT STATISTICS PNUMBER=%ld DETAIL=YES ;",
cdSess.SessionINF->GetProcessNumber());
(cdSess.SessionINF->SendCommand(processText) == CD_SUCCESS)
           {
char *cP = NULL;

do
                {
                    if (cdSess.SessionINF->ReceiveResponse() == CD_SUCCESS)
                    {
```

ConnectDirectSession class *(continued)*

```
if (cP = strstr((char *)cdSess.SessionINF->GetResponse(), "RECI"))
{
                     if (cP = strstr((char *)cdSess.SessionINF->GetResponse(),
"CTRC"))
{
                            printf("Code is %4.4s\n", cP);

                        cP = strstr((char *)cdSess.SessionINF->GetResponse(),
"CCOD");
if
                            if (*(cP + 5) == '0')
                            {
                                printf("COPY Completion Code = %c\n", *(cP + 5));
                                break;
                            }
                     else
                     {
                     cP = strstr((char *)cdSess.SessionINF->GetResponse(), "MSST");
                            }
                        }
                    }
                }
                else
                {
                   getError(cdSess);
                   break;
                }
            } while (cdSess.SessionINF->MoreData());
        }
      else
          {
          getError(cdSess);
      }
    }
        else
        {
      getError(cdSess);
    }
 cdSess.SessionINF->DisConnect();
    }
    else
    {
        getError(cdSess);
    }
}
void getError(ConnectDirectSession& cdSess)
{
    if (cdSess.SessionINF->GetFirstError())
    {
        printf("\nError Message:  %s",   cdSess.SessionINF->GetMsgID());
        printf("\nError Feedback: %d",   cdSess.SessionINF->GetFeedBackCode());
        printf("\nError RC:       %d",   cdSess.SessionINF->GetReturnCode());
        printf("\nError SUBST:    %s\n", cdSess.SessionINF->GetSubstitute());    }
```

ConnectDirectSession class *(continued)*

```
 while(cdSess.SessionINF->GetNextError())
    {
        printf("\nError Message:  %s",   cdSess.SessionINF->GetMsgID());
        printf("\nError Feedback: %d",   cdSess.SessionINF->GetFeedBackCode());
        printf("\nError RC:       %d",   cdSess.SessionINF->GetReturnCode());
        printf("\nError SUBST:    %s\n", cdSess.SessionINF->GetSubstitute());

        }
}
```

The ConnectDirectSession class methods are described in the following table:

Method	Description	Parameter	Return Values
Connect	Provides a connection to the Connect:Direct server. Connect() with a void parameter connects to the hostname and port specified in the client configuration file.	void or a pointer to an IP address and port.	CD_SUCCESS or CD_FAILURE
DisConnect	Disconnects the current session.	void	CD_SUCCESS or CD_FAILURE
SendCommand	Sends a Connect:Direct command to the server for processing.	Pointer to a command text buffer.	CD_SUCCESS or CD_FAILURE
ReceiveResponse	Receives the response from a previously issued command, such as the select statistics command.	void	CD_SUCCESS or CD_FAILURE
GetResponse	Retrieves the response from the ReceiveResponse call.	void	Pointer to a response buffer.
GetResponseLength	Returns the length of the previously received response buffer.	void	Length of the response buffer from the previously issued call.
MoreData	Returns a value indicating if outstanding data from the previously issued send command call is available. If the return value is TRUE, call ReceiveResponse again to retrieve more data.	void	TRUE—If more data is outstanding. FALSE—If no data is outstanding.
GetCommandName	Returns the command name of the previously issued send command, such as the submit command.	void	Pointer to a command name buffer.
GetProcessNumber	Returns the Process number of a previously issued submit command.	void	Process number of a submit command. -1—If no submit command can be found.

Method	Description	Parameter	Return Values
GetProcessCount	Returns the number of Processes affected by the last send command that issued a delete, change, or flush process.	void	Process number of a submit command that issued a delete, change or flush process. -1—If no submit command can be found.
GetCurrentError	Moves the error data pointer to the current error in the list.	void	TRUE—If successful FALSE—If no current error exists.
GetNextError	Moves the error data pointer to the next error in the list.	void	TRUE—If successful FALSE—If no more errors are found.
GetPreviousError	Moves the error data pointer to the previous error in the list.	void	TRUE—If successful FALSE—If no previous error exists.
GetFirstError	Moves the error data pointer to the first error in the list.	void	TRUE—If successful FALSE—If no error is found.
GetLastError	Moves the error data pointer to the last error in the list.	void	TRUE—If successful, otherwise FALSE.
GetMsgID	Retrieves the message of the current error data block. You must call one of the GetXXXXError methods before calling this method in order to retrieve the proper results.	void	Return Value: Pointer to a message ID if data block is value.
GetFeedBackCode	Returns the feedback code of the current error data block.	void	Feedback code.
GetReturnCode	Returns the Connect Direct return code.	void	One of the valid Connect:Direct return code: 1,4,8,16.
GetStatus	Returns the status.	void	Connect:Direct status code.
GetSubstitute	Returns the current substitution buffer associated with the error.	void	Pointer to a substitution buffer.
DisplayError	Displays the current error chain to an output location.	Parameters: Pointer to a file I/O structure.	Return Value: Returns the highest error found in the error chain or -1 on error.

Following is the ConnectDirectSession class header:

```
#include <stdio.h>

// Error enumeration.
typedef enum CDErrorCode
{
    CD_SUCCESS =  0,
    CD_FAILURE = -1

} CDErrorCode;

// <<Interface>>
class CDSession
{
public:
    // Communication methods...
    virtual CDErrorCode Connect(void) = 0;
    virtual CDErrorCode Connect(char *IpAddress, char *IpPort) = 0;
    virtual CDErrorCode DisConnect(void) = 0;
    virtual CDErrorCode SendCommand(char *CmdText) = 0;
    virtual CDErrorCode ReceiveResponse(void) = 0;

    // Methods for retrieving ReceiveResponse data...
    virtual const char *GetResponse(void) = 0;
    virtual int         GetResponseLength(void) = 0;
    virtual bool        MoreData(void) = 0;

    // Methods for retrieving SendCommand return data...
    virtual const char *GetCommandName(void) = 0;
    virtual long        GetProcessNumber(void) = 0;
    virtual long        GetProcessCount(void) = 0;

    // Methods to iterate over error collection ...
    virtual bool        GetCurrentError(void) = 0;
    virtual bool        GetNextError(void) = 0;
    virtual bool        GetPreviousError(void) = 0;
    virtual bool        GetFirstError(void) = 0;
    virtual bool        GetLastError(void) = 0;

    // Methods to retrieve error data...
    virtual const char *GetMsgID(void) = 0;
    virtual int         GetFeedBackCode(void) = 0;
    virtual int         GetReturnCode(void) = 0;
    virtual int         GetStatus(void) = 0;
    virtual const char *GetSubstitute(void) = 0;

    // Method to display error collection...
    virtual int   DisplayError(FILE *Output) = 0;
};

class ConnectDirectSession
{
public:
    // Interface classes
    CDSession *SessionINF;

    ConnectDirectSession();
    ~ConnectDirectSession();
};
```

Chapter 6

User Exits

The user exit API functions allow you to write custom programs to use with Connect:Direct.

Understanding User Exit Functions

The user exit programs are used by Connect:Direct to invoke user-specific logic at strategic points within Connect:Direct execution. User exit programs must be C or C++ language programs and cannot be shell scripts. The PMGR invokes the Statistics user exit program when you start Connect:Direct and the exit runs as long as Connect:Direct runs. The SMGR invokes the File Open and Security user exits for each session and stops them when the particular session terminates.

The user exit programs are described in the following table:

Program	Description
File Open Exit	Connect:Direct sends a message to this user exit program to open the source or destination file during processing of the copy statement. The File Open Exit opens the source file and identifies the file descriptor. This exit can perform any sort of processing to file names or directory names. It can also redirect the open request to other files as needed. The File Open Exit program (named “exit_skeleton” in this example) must be owned by root and the setuid bit must be set. Use the following commands: % chown root exit_skeleton % chmod u+s exit_skeleton
Security Exit	The Security Exit enables you to implement your own security system or provide access to a third-party security system.
Statistics Exit	The Statistics Exit is a notification to the user exit program after any record is written to the statistics file. Whenever a statistics record is written to the statistics file, an exact copy is passed to this exit.

User Exit Functions

A connection between the user exit and Connect:Direct is established when the user exit program calls the **exit_child_init() or exit_child_init_c()** function. The connection is terminated through a specially designated stop message. The types of messages are defined in the include file user_exit.h. The following functions facilitate communications between the user exit and Connect:Direct:

C++ Function	C Function	Description
exit_child_init()	exit_child_init_c()	Use this function as the first line in a user exit program to initialize communications between Connect:Direct and the user exit program.
recv_exit_msg()	recv_exit_msg_c()	Used by both Connect:Direct and the user exit program to receive a message from the other Process. The receive exit messages wait for a response from the other Process.
send_exit_file()	send_exit_file_c()	The user exit program uses this function when it has opened a file for Connect:Direct. This function uses underlying UNIX methods to pass an open file descriptor. from one Process to another.
send_exit_msg()	send_exit_msg_c()	Both Connect:Direct and the user exit program use this function to send a message to the other Process. Send messages are followed with a receive message to get the response from the other Process.

Initializing Communications with exit_child_init() or exit_child_init_c()

Use the **exit_child_init()** or **exit_child_init_c()** function as the first line of code of the user exit program to initialize communications. This function performs a check to verify that each side is ready to communicate. Following is the format of the **exit_child_init()** function:

```
int exit_child_init( char * logfile )
```

Following is the parameter for the **exit_child_init()** or **exit_child_init_c()** function:

Parameter	Description	Value
logfile	The name of the log or trace file that is opened for use by the user exit programs. Because the file open and security exit are started by SMGR, which is running as root, the exits also run as root. Running the exits as root can cause problems with file permissions of the log file, so **logfile** enables you to easily change owner or permissions on the file. See the sample exit in *d_dir*/ndm/src/exit_skeleton.c for more details.	Name of log file or trace file

Following are the return codes for the **exit_child_init()** or **exit_child_init_c()** function. Return codes for the function are defined in ndmapi.h.

Return Code	Description
GOOD_RC	Communications between Connect:Direct and the user exit program were successfully initialized.
ERROR_RC	Communications between Connect:Direct and the user exit program could not be initialized.

Waiting for a Message Using recv_exit_msg() or recv_exit_msg_c()

The **recv_exit_msg()** or **recv_exit_msg_c()** function waits until it receives a message from Connect:Direct. Control is suspended until a message is received or an error occurs. Following is the format of **recv_exit_msg()**:

```
int recv_exit_msg( int exit_flag )
  int * msg_type,
  char * recv_buf,
  int * recv_buf_len
```

Following are the parameters for **recv_exit_msg()** or **recv_exit_msg_c()** functions:

Parameter	Description	Value
exit_flag	A flag to specify the recipient ID. The only valid value a user exit program can use is EXIT_PROGRAM.	EXIT_PROGRAM
msg_type	A pointer to the name of the received message. Messages are requests from Connect:Direct and the associated response from the user exit program.	Pointer to message
recv_buf	A pointer to the memory location of the message.	Pointer to message
recv_buf_len	The length in bytes of the message to be received.	Length of message

Following are the return codes for **recv_exit_msg()**or **recv_exit_msg_c()**. Return codes for the function are defined in ndmapi.h.

Return Code	Description
GOOD_RC	The message was received successfully.
ERROR_RC	An error occurred and the message was not received successfully. Possible causes include: Connect:Direct terminated, an invalid value used for the **exit_flag** parameter, or the receiving buffer not large enough to hold the message received.

Passing a File Descriptor Using send_exit_file() or send_exit_file_c()

Use the **send_exit_file()** or **send_exit_file_c()** function to pass a file descriptor from one Process to another Process. Following is the format of **send_exit_file()**:

```
int send_exit_file int  exit_flag
  int fd
```

Following are the parameters for **send_exit_file()** or **send_exit_file_c()**:

Parameter	Description	Value
exit_flag	A flag to specify the sender ID. The only valid value a user exit program can use is EXIT_PROGRAM.	EXIT_PROGRAM
fd	The file descriptor of a file that the user exit program opened in the place of Connect:Direct, similar to one returned by the open(2) function.	File descriptor

Following are the return codes for **send_exit_file()** or **send_exit_file_c()**. Return codes for the function are defined in ndmapi.h.

Return Code	Description
GOOD_RC	The file descriptor was received successfully.
ERROR_RC	An error occurred and the file descriptor was not sent successfully. Possible causes include: Connect:Direct terminated, an invalid value used for the **exit_flag** or **fd** parameters, or the last message sent was not send_exit_msg.

Send a Message to Connect:Direct Using send_exit_msg() or send_exit_msg_c()

The **send_exit_msg()** or **send_exit_msgc()** function enables the user exit program to send a message to Connect:Direct. This function returns control to the caller immediately after the message is queued.

Following is the format of the **send_exit_msg()** function:

```
int send_exit_msg int exit_flag
  int msg_type,
  char * send_buf,
  int send_buf_len
```

Following are the parameters for **send_exit_msg()** or **send_exit_msg_c()**:

Parameter	Description	Value
exit_flag	A flag to specify the sender ID. The only valid value a user exit program can use is EXIT_PROGRAM.	EXIT_PROGRAM
msg_type	A message name. Messages are requests from Connect:Direct and the associated response from the user exit program.	Pointer to message
send_buf	A pointer to the memory location of the message to be sent.	Pointer to message
send_buf_len	The length in bytes of the message to be sent.	Length of message

Following are the return codes for **send_exit_msg()** or **send_exit_msg_c()**. Return codes for the function are defined in ndmapi.h.

Return Code	Description
GOOD_RC	The message was sent successfully.
ERROR_RC	An error occurred and the message was not sent successfully. Possible causes include: Connect:Direct terminated or an invalid value is used for the **exit_flag** or **msg_type** parameters.

Understanding User Exit Messages

Connect:Direct sends and receives messages, using the **send_exit_msg()** and the **recv_exit_msg()** functions for a C++ program or the **send_exit_msg_c()**and the **recv_exit_msg_c()** functions for a C program. For the exact definition of the data sent in each message, see the include files located in *d_dir*/ndm/include/user_exit.h and *d_dir*/ndm/include/user_exit2.h.

Note: The copy control block is defined in user_exit2.h.

Statistics Exit Message

The statistics exit has only one type of message, the STATISTICS_LOG_MSG.

Connect:Direct sends a STATISTICS_LOG_MSG to the user exit program. Every time Connect:Direct writes a statistic record, this message provides an exact copy of the character string. The STATISTICS_LOG_MSG contains the Connect:Direct statistics record.

File Open Exit Messages

The file open exit has four types of messages:

- FILE_OPEN_OUTPUT_MSG
- FILE_OPEN_OUTPUT_REPLY_MSG
- FILE_OPEN_INPUT_MSG
- FILE_OPEN_INPUT_REPLY_MSG

The file open exit has the following limitations:

- The oflag parameter passed to the user exit is already calculated based on the file disposition, as explicitly specified on the copy statement or using the default value. If the user exit changes the oflag to truncate and the original disposition is mod meaning the copy will append to the end of file if the file already exists, then the user exit causes the Process to behave differently from how the Process language is documented.
- Do not change the file type specified by the Process. For example, if the Process specifies a regular file, the user exit cannot open and return a file descriptor for a pipe. No facility is available to modify contents of the copy control block and return it to Connect:Direct.
- If the oflag specifies opening a file with write access and the user exit changes access to read-only, Connect:Direct will fail when it attempts to write to a read-only file.
- The upload and download parameters that restrict directory access are ignored for this user exit.

  For more information about these parameters, refer to Chapter 6, *Maintaining Access Information Files* in the *Connect:Direct UNIX Administration Guide*.

FILE_OPEN_OUTPUT_MSG

During the copy statement process, Connect:Direct sends a FILE_OPEN_OUTPUT_MSG to the user exit program to open the destination file. The FILE_OPEN_OUTPUT_MSG contains:

- The open function oflag parameter (for example, O_CREAT|O_RDWR|O_TRUNC)
- The open function mode parameter, which controls file permissions
- UNIX user ID that will own the file
- UNIX group ID that will own the file
- UNIX user name
- A copy of the Connect:Direct copy control block
- A copy of the Connect:Direct parsed sysopts structure (the copy control block contains the actual raw version from the process)

FILE_OPEN_OUTPUT_REPLY_MSG

The user exit program sends a reply message to the Connect:Direct FILE_OPEN_OUTPUT_MSG. The FILE_OPEN_OUTPUT_REPLY_MSG contains:

- Status value of zero for successful or non zero for failure
- Status text message (if status value is failure, status text message is included in the error message)

- Pipe pid (for pipe I/O, the UNIX process ID of the shell process that is performing the shell command for pipe I/O)
- Actual file name opened (to be used in statistics log messages)

If the status value is zero for successful, the user exit program must immediately call **send_exit_file()** or **send_exit_file_c()** to send the file descriptor of the opened file to Connect:Direct.

FILE_OPEN_INPUT_MSG

During the copy statement Process, Connect:Direct sends a FILE_OPEN_INPUT_MSG to the user exit program to open the source file. The FILE_OPEN_INPUT_MSG contains:

- The open function oflag parameter (for example, O_RDONLY)
- The open function mode parameter, which controls file permissions
- UNIX user ID that will own the file
- UNIX group ID that will own the file
- UNIX user name
- A copy of the Connect:Direct copy control block
- A copy of the Connect:Direct parsed sysopts structure (the copy control block contains the actual raw version from the Process)

FILE_OPEN_INPUT_REPLY_MSG

This message type is used when the user exit program sends a reply message to the Connect:Direct FILE_OPEN_INPUT_MSG. The FILE_OPEN_INPUT_REPLY_MSG contains:

- Status value of zero for success or non zero for failure
- Status text message (if status value is failure, status text message is included in the error message)
- Pipe pid (for pipe I/O, the UNIX process ID of the shell process that is performing the shell command for pipe I/O)
- Actual file name opened (used in statistics log messages)

Security Exit Messages

The security exit contains four types of messages:

- GENERATE_MSG
- GENERATE_REPLY_MSG
- VALIDATE_MSG
- VALIDATE_REPLY_MSG

Caution: If the security exit is used, Connect:Direct relies on it for user ID authentication. If the security exit is not implemented correctly, the security can be compromised.

GENERATE_MSG

Connect:Direct sends a generate message to the user exit program at the start of a session to establish a security environment. The PNODE sends the GENERATE_MSG to the security exit to determine a user ID and security token to use for authentication on the SNODE. The GENERATE_MSG contains:

- Submitter ID
- PNODE ID
- PNODE ID password, if user specified one
- SNODE ID
- SNODE ID password, if user specified one
- PNODE name
- SNODE name

GENERATE_REPLY_MSG

The user exit program sends a reply message to Connect:Direct. The GENERATE_REPLY_MSG contains:

- Status value of zero for success or non zero for failure
- Status text message (if status value is failure, status text message is included in the error message)
- ID to use for security context on the SNODE side (may or may not be the same ID as in the generate message)
- Security token used in conjunction with ID for security context on the SNODE side

VALIDATE_MSG

Connect:Direct sends a validate message to the user exit program. The SNODE sends the VALIDATE_MSG to the security exit to validate the user ID and security token received from the PNODE. The VALIDATE_MSG contains:

- Submitter ID
- PNODE ID
- PNODE ID password, if user specified one
- SNODE ID
- SNODE ID password, if user specified one
- PNODE name
- SNODE name
- ID to use with security token
- Security token (password, PASSTICKET, or other security token)

VALIDATE_REPLY_MSG

The user exit program sends a reply message to the Connect:Direct VALIDATE_MSG. The VALIDATE_REPLY_MSG contains:

- Status value of zero for success or non zero for failure

- Status text message (if status value is failure, status text message is included in the error message)
- ID used for security context
- Security token to use in conjunction with ID for security context

User Exit Stop Message

Connect:Direct sends the stop message, STOP_MSG, when all useful work for the user exit is complete and to notify the user exit to terminate. A user exit should terminate only when a stop message is received or if one of the above listed user exit functions returns an error code.

Copy Control Block

The copy control block structure contains the fields, which control how Connect:Direct Processes the copy statement Process file.

Exit Log Files

If user exit programs are specified in the initparm.cfg, Connect:Direct creates exit logs. Exit log files are provided specifically for the user exit programs and are used for debug and trace type messages. The user exit program is started with the log file already opened on STDOUT and STDERR. The exit log files are:

- stat_exit.log
- file_exit.log
- security_exit.log

Note: You can access the log files through the normal **printf()** and **fprintf** (stderr,...) functions.

The log files are located in the installed (d_dir) working directory:

```
.../d_dir/work/cd_node
```

Glossary

A

Application Programming Interface (API)

An application that enables End User Applications (EUAs) and the Connect:Direct Command Line Interface (CLI) to interact with Connect:Direct.

C

Client

The program that executes commands sent by the CLI/API and sends the results back to the CLI/API. Carries out the Connect:Direct authentication procedure in conjunction with API to determine access to Connect:Direct. Interacts with the PMGR when executing commands.

Command Line Interface (CLI)

A program available to submit Connect:Direct Processes and commands from a command line environment.

Command Manager (CMGR)

The program that executes commands sent by the API and sends the results back to the API. In conjunction with the API, the CMGR carries out the Connect:Direct authentication procedure, which determines if the user name and password are authorized to access the system. CMGR interacts with the PMGR when required by command execution.

Connect Control Center

centralized management system that provides operations personnel with continuous enterprise-wide business activity monitoring capabilities for Connect:Direct z/OS, UNIX, and Windows servers. It manages multiple Connect:Direct servers to suspend, release, and delete Processes, stops Connect:Direct servers, and views detailed statistics on running or completed Processes. It monitors service levels to view Connect:Direct processing across Connect:Direct z/OS, UNIX, and Windows servers within your network and retrieve information about active and completed Processes. It receives notification of data delivery events that occur or do not occur as scheduled and defines rules that, based on processing criteria, can generate an alert, send an E-mail notification, generate a Simple Network Management Protocol (SNMP) trap to an Enterprise Management System (ESM), or run a system command. It monitors for alerts, such as a server failure or a Process not starting on time.

Connect:Direct

The family of data transfer software products that distributes information and manages production activities among multiple data centers.

Connect:Direct Browser User Interface

As an alternative to submitting Connect:Direct commands through the command line interface, you can use the Connect:Direct Browser User Interface to create, submit, and monitor Processes from an Internet browser, such as Microsoft Internet Explorer or Netscape Navigator. You can also use the Connect:Direct Browser to perform Connect:Direct system administration tasks, such as viewing and changing the network map or initialization parameters, if you have the appropriate Connect:Direct authority.

Connect:Direct for UNIX

The UNIX implementation of the Connect:Direct product.

Connect:Direct Node

Any computer/workstation running Connect:Direct.

Connect:Direct Process

A series of statements, which can be predefined and stored in a directory, submitted through the API to initiate Connect:Direct for UNIX activity. Examples of Process functions are copying files and running jobs.

D

daemon

The long-running process that provides a service to a client. The PMGR is the Connect:Direct for UNIX daemon.

Diagnostic Commands

Connect:Direct commands that assist in the diagnosis of Connect:Direct software problems.

E

End User Application (EUA)

An application program developed by an end user to accomplish a particular task.

Execution Queue

A logical queue in the TCQ. A Process in the Execution Queue can be transferring data to or from a remote Connect:Direct node or it can be waiting for a connection to the remote Connect:Direct node before it can perform its tasks.

F

File Agent

An application program and component of Connect:Direct. It scans specified directories searching for the presence of a file. When a file appears in a watched directory, Connect:Direct either submits a Process or performs the action specified by the rules for the file.

H

Hold Queue

A logical queue in the TCQ. Processes in the Hold Queue are waiting for operator intervention before they move to the Wait Queue for scheduling.

M

Monitoring Commands

Connect:Direct commands that allow you to display information from the statistics file and the TCQ about Connect:Direct Process execution results.

O

Operational Control Commands

Connect:Direct commands that allow you to submit a Process, change specific characteristics of a Process in the TCQ, remove executing and nonexecuting Processes from the TCQ, and stop Connect:Direct.

P

Process Manager (PMGR)

The long-running Connect:Direct server that initializes the Connect:Direct software, accepts connection requests from Connect:Direct APIs and remote Connect:Direct nodes, creates Command Managers and Session Managers, accepts requests from Command Managers and Session Managers where centralized Connect:Direct functions are required, and terminates Connect:Direct software execution.

PNODE (Primary Node)

The Connect:Direct node on which the Process is being executed. The primary node is also called the controlling or source node, but is not always the sending node because PNODE can be the receiver. Every Process has one PNODE and one SNODE. The submitter of a Process is always the PNODE. If defined in the **netmap.cfg** file, the PNODE name can be 1–16 characters long. If not defined in the **netmap.cfg** file, the PNODE name can be 1–256 alphanumeric characters long.

S

Session

A connection between two Connect:Direct nodes.

Session Manager (SMGR)

Responsible for creating or completing a connection with a remote Connect:Direct node and carrying out the Connect:Direct work to be performed.

SNODE (Secondary Node)

The Connect:Direct node that interacts with the Primary node (PNODE) during Process execution. The Secondary node (SNODE) also can be referred to as the participating, target, or destination node. Every Process has one PNODE and one SNODE. The secondary node is the node participating in Process execution initiated by another node (PNODE). If defined in the **netmap.cfg** file, the SNODE name can be 1–16 characters long. If not defined in the **netmap.cfg** file, the SNODE name can be 1–256 alphanumeric characters long.

T

TCQ Status Value

A two-letter code assigned to a Process by Connect:Direct when the Process is placed on the TCQ. The status of a Process can be examined with a **select process** command.

TCQ (Transmission Control Queue)

A queue that holds all Processes that are submitted to Connect:Direct for UNIX. TCQ contains the following four logical queues:

- EXECUTION
- WAIT
- TIMER
- HOLD

Timer Queue

A logical queue in the TCQ. Processes on the Timer Queue are waiting for a start time before they move to the Wait Queue for scheduling.

W

Wait Queue

A logical queue in the TCQ. Processes on the Wait Queue are waiting on a connection to or from the remote Connect:Direct node.

Index