# Compaq NonStop™ Pathway/iTS TCP and Terminal Programming Guide

## Abstract

This manual is a guide for programmers who are writing SCREEN COBOL requesters to be used in Pathway applications.

## Product Version

Pathway/iTS 1.0

| Part Number | Published |
|---|---|
| 426751-001 | October 2000 |

## Document History

| Part Number | Product Version | Published |
|---|---|---|
| 110075 | Pathway/TS D30+ | July 1995 |
| 121308 | Pathway/TS D40 | December 1995 |
| 426751-001 | Pathway/iTS 1.0 | October 2000 |

# Compaq NonStop™ Pathway/iTS TCP and Terminal Programming Guide

Index | Examples | Figures | Tables

## Figures

## Tables

# What's New in This Manual

## Manual Information

### Abstract

This manual is a guide for programmers who are writing SCREEN COBOL requesters to be used in Pathway applications.

### Product Version

Pathway/iTS 1.0

| Part Number | Published |
|---|---|
| 426751-001 | October 2000 |

### Document History

| Part Number | Product Version | Published |
|---|---|---|
| 110075 | Pathway/TS D30+ | July 1995 |
| 121308 | Pathway/TS D40 | December 1995 |
| 426751-001 | Pathway/iTS 1.0 | October 2000 |

## New and Changed Information

The Compaq *NonStop*™ Pathway/iTS product was formerly called Pathway/TS. For the Pathway/iTS 1.0 independent product release, the product was renamed to conform to current Compaq product naming standards and to reflect the new internet (web client) capabilities of the product. After the first reference to the product name in each section of this manual, subsequent references use the shortened form of the name, Pathway/iTS.

### Product Changes

This manual refers to the new capability of Pathway/iTS to convert SCREEN COBOL object files to web clients. However, this manual focuses on programming for terminal and IDS requesters; for details about web client programming, the reader is referred to the new *Compaq NonStop™ Pathway/iTS Web Client Programming Manual*.

### Corrections and Enhancements to the Manual

The following corrections and enhancements have been made to this manual:

- References have been added to context-sensitive Pathsend requesters, a feature that was added to the NonStop™ TS/MP and Extended General Device Support (GDSX) products since the last edition of this manual.

- The discussion of User-Written User Conversion Procedures on page 4-2 has been corrected to reflect the use of pTAL and the `nld` utility.
- A repeated syntax error in the programming examples in Section 7, Processing Unsolicited Messages has been corrected.
- References to Compaq trademarks have been updated.
- References to obsolete products have been removed.
- Miscellaneous terminology changes and editorial corrections have been made.

# About This Manual

This manual is a guide for programmers who are writing SCREEN COBOL requesters to be used in Pathway applications. It describes how to use the major features and capabilities available with SCREEN COBOL, such as transaction management and intelligent device support.

This manual is intended to be used in conjunction with the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*, which contains detailed reference information about the SCREEN COBOL programming language.

## Who Should Read This Manual

Readers of this manual should be experienced programmers familiar with the *Guardian* operating environment on Compaq *NonStop™ Himalaya* systems and with the SCREEN COBOL programming language.

## Related Documentation

This manual is one in a set of Compaq manuals for the *NonStop™* TS/MP and Pathway/iTS products.

The following manuals may be useful:

| | |
|---|---|
| *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* | Describes the SCREEN COBOL programming language which is used for writing programs that define and control terminal displays or intelligent devices for online transaction processing applications running in a PATHMON environment. |
| *Compaq NonStop™ Pathway/iTS SCUP Reference Manual* | Describes managing a SCREEN COBOL library with the SCREEN COBOL Utility Program (SCUP). |
| *Compaq NonStop™ Pathway/iTS Web Client Programming Manual* | Describes how to convert SCREEN COBOL requesters to web clients, explains how to build and deploy those clients, and also provides the information Java developers and web designers need to to modify and enhance the Java and HTML portions of the converted clients. |
| *Compaq NonStop™ Pathway/iTS System Management Manual* | Describes the interactive management interface to the Pathway/iTS product and describes how to configure and manage Pathway/iTS objects. |
| *Compaq NonStop™ Pathway/iTS Management Programming Manual* | Describes the management programming interface for Pathway/iTS objects in the PATHMON environment. |
| *Compaq NonStop™ Pathway Products Glossary* | Defines technical terms used in this manual and in other manuals for the Pathway products: Pathway/iTS, NonStop™ TS/MP, and Pathway/XM. |

| | |
|---|---|
| *Operator Messages Manual* | Describes all messages that are distributed by the Event Management Service (EMS), including those generated by NonStop™ TS/MP and Pathway/iTS processes. |
| *Guardian Procedure Errors and Messages Manual* | Describes the Guardian messages for NonStop™ Himalaya systems. The manual covers various types of error codes and error lists associated with Guardian procedure calls and also the interprocess messages sent to application programs by the operating system and the command interpreter. |
| *NonStop™ TM/MP Application Programmer's Guide* | Provides additional information, beyond what is covered in this manual, about programming for the Transaction Management Facility (TMF) subsystem. |

## Your Comments Invited

After using this manual, please take a moment to send us your comments. You can do this by returning a Reader Comment Card or by sending an Internet mail message.

A Reader Comment Card is located at the back of printed manuals and as a separate file on the Compaq CD Read disc. You can either FAX or mail the card to us. The FAX number and mailing address are provided on the card.

Also provided on the Reader Comment Card is an Internet mail address. When you send an Internet mail message to us, we immediately acknowledge receipt of your message. A detailed response to your message is sent as soon as possible. Be sure to include your name, company name, address, and phone number in your message. If your comments are specific to a particular manual, also include the part number and title of the manual.

Many of the improvements you see in Compaq manuals are a result of suggestions from our customers. Please take this opportunity to help us improve future manuals.

# Notation Conventions

## General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

**UPPERCASE LETTERS.** Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

**lowercase italic letters.** Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

*`file-name`*

**Punctuation.** Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

*`error`* `:= NEXTFILENAME (` *`file-name`* `) ;`

`LISTOPENS SU $`*`process-name`*`.#`*`su-name`*

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

`"["` *`repetition-constant-list`* `"]"`

**Item Spacing.** Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

`CALL STEPMOM (` *`process-id`* `) ;`

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

`$`*`process-name`*`.#`*`su-name`*

## Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

**Nonitalic text.** Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

**lowercase italic letters.** Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

*`p-register`*

*`process-name`*

# 1 Introduction to TCP and Terminal Application Programming

This section introduces Pathway transaction processing applications, which you write and run with the assistance of the NonStop™ Transaction Services/MP (NonStop™ TS/MP) and Compaq *NonStop*™ Pathway/iTS software. The emphasis of this section is on applications that include SCREEN COBOL requesters for use with terminals or intelligent devices.

**Table 1-1. Task and Manual Correspondences**

| If Your Application Includes… | You Need… | To Perform the Following… |
|---|---|---|
| SCREEN COBOL requesters | Section 2 | Design an application including SCREEN COBOL requesters |
| | Section 3 | Handle programming for specific terminals, terminal emulators, intelligent devices, or simulated devices |
| | Section 4 | Write user conversion procedures that make custom validation checks or data conversions |
| | Section 5 | Manage transactions with the Compaq Transaction Management Facility (*TMF*) |
| | Section 6 | Use the Pathway/iTS intelligent device support (IDS) facility |
| | Section 7 | Accept and reply to unsolicited messages from *Guardian* operating environment processes outside the Pathway environment |
| | Section 8 | Process double-byte character sets |
| | Section 9 | Set device-dependent functions with SETMODE calls or perform device-dependent I/O operations with CONTROL calls |
| | Section 10 | Handle errors returned to a SCREEN COBOL requester program |
| | Appendix A | Use the MAKEUL macro for creating the native user library and facilitating pTAL compilation |

If you are writing SCREEN COBOL requesters that communicate with Compaq NonStop™ TUXEDO servers, refer also to the manuals for the NonStop™ TUXEDO system, particularly the *Compaq NonStop™ TUXEDO System Application Development Guide* and the *NonStop™ TUXEDO System Pathway Translation Servers Manual*.

If you are writing Pathsend requesters or Pathway servers, refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

If you are writing web clients created from SCREEN COBOL requesters, refer to the *Compaq NonStop™ Pathway/iTS Web Client Programming Manual*.

## Advantages of the Pathway Environment

NonStop™ TS/MP and Pathway/iTS provide ease of development, manageability, and the fundamental strengths and benefits of Compaq NonStop™ Himalaya systems. The strengths and benefits of NonStop™ Himalaya systems include data integrity, fault tolerance, high performance and low cost, system security, scalability, and distributed processing. The following paragraphs describe how NonStop™ TS/MP, Pathway/iTS, and related products—known together as the Pathway environment—benefit the application designer and programmer. The *Introduction to NonStop™ Transaction Processing* provides a fuller description of how all the fundamentals of NonStop™ Himalaya systems apply to transaction processing.

### Ease of Development

Development costs are one of the highest expenses associated with online transaction processing (OLTP) systems. The more sophisticated the features and safeguards that are built into your OLTP application—for example, multiprocessing, fault tolerance, and data integrity—the greater the costs. When you use NonStop™ TS/MP, Pathway/iTS, and related Compaq transaction processing products for NonStop™ Himalaya systems to create your OLTP applications; development time and efforts, and therefore costs, can be measurably reduced.

This cost reduction occurs because:

- NonStop™ TS/MP, Pathway/iTS, and related products provide the most complex components of an OLTP application:
  - NonStop™ TS/MP includes the transaction monitor (PATHMON), the command interpreter for management (PATHCOM), and the means for interprocess communication.
  - Pathway/iTS provides a multithreaded terminal control process (TCP) for communication with terminals, including fault tolerance and transaction protection.
  - The NonStop™ Transaction Manager/MP (NonStop™ TM/MP) product provides transaction management.
- Compaq makes valuable application development tools and utilities available for the Pathway environment. These development tools and utilities can significantly reduce the amount of programming time and effort required to generate a working Pathway application.

The Compaq NonStop™ Remote Server Call/MP (RSC/MP) product facilitates client/server computing, allowing workstation applications to access Pathway servers. A number of packaged tools and utilities are commercially available for use with RSC/MP.

- The Pathway environment helps you standardize program code. You can repeat and reuse code; you do not have to write the same requester and server programs over and over again. This ability to reuse code saves development time.
- The Pathway environment allows you to isolate and test your requester and server programs before adding them to a running application. This capability is important because coding errors are difficult, time-consuming, and expensive to find after an application is put into production.
- OLTP products that are compatible with the Pathway environment are available from many third-party vendors.

In addition to making initial development faster and easier, the structured Pathway environment allows you to implement enhancements and develop new applications by simply adding new requesters, sharing existing servers, or adding new servers to the existing application. You can use code modules in the existing application as templates for new modules in the modified or new application.

## Manageability

Online transaction processing operations present a dynamic environment in which hundreds of different transactions—from disparate locations and many different I/O devices—can be entered concurrently and processed within seconds. To process hundreds of transactions, thousands to millions more application program instructions must be executed. It is critical that you be able to control and monitor such a complex processing environment.

To control and monitor your Pathway environment—as well as simplify the task of system management—NonStop™ TS/MP provides the following:

- A PATHMON process, which provides a single point of control over your OLTP applications and operations
- A choice of two different system management interfaces: the interactive PATHCOM interface and the Subsystem Programmatic Interface (SPI)
- Status and error reporting capabilities, provided through a log file and through the Event Management Service (EMS)

Because NonStop™ TS/MP provides these processes and capabilities, you do not have to spend the time and money to develop, test, and implement comparable mechanisms.

For more information about the PATHMON process, the management interfaces, and status and error reporting capabilities in the Pathway environment, refer to the *NonStop™ TS/MP System Management Manual*, the *Compaq NonStop™ Pathway/iTS System Management Manual*, the *NonStop™ TS/MP Management Programming Manual,* and the *Compaq NonStop™ Pathway/iTS Management Programming Manual.*

## Data Integrity

If your database is corrupted by a hardware or software failure, you might need weeks to isolate and then correct the problem. Because an inaccessible or inconsistent database can have a dramatic, adverse effect on business operations, the Transaction Management Facility (TMF) subsystem, provided in the NonStop™ TM/MP product, was developed as a way of ensuring database consistency. The TMF subsystem, which works with NonStop™ TS/MP, protects the entire database from catastrophic system failures by maintaining an audit trail of database changes (that is, transactions); an audit trail is also commonly known as a transaction log. You can use the audit trail to rebuild the database in the event of a hardware or software failure.

The design of Pathway servers supports the integrity of individual transactions and therefore transaction processing protection as a whole. Because the requester/server model allows a clear division of processing functions, application programmers can code each server program to handle a specific set of transaction types: for example, checking an account balance, entering a new customer, or updating the parts inventory. The server processes service their transactions by performing the same set of tasks over and over again. In this way, a valid transaction is defined as a specific set of tasks both by the requester program and within the server logic.

If for any reason a server is unable to complete all tasks involved in processing a transaction, it can abort the transaction and thereby maintain the transaction’s integrity. The server does not have to wait for the requester to abort the transaction.

## Fault Tolerance

Because OLTP systems automate core business operations and deliver key business services, companies depend on OLTP applications to stay up and running—even if a hardware or software component fails.

NonStop™ Himalaya systems, which are specifically intended for online transaction processing, are designed to remain continuously available during the hours when transactions are being entered and business is being conducted. Typically, a NonStop™ Himalaya system can continue processing despite the failure of any single software or hardware component within that system. This ability is referred to as fault tolerance.

In the Pathway enviroment, automatic fault tolerance (that is, fault tolerance that does not require any additional programming effort on your part) is provided by the use of process pairs and the actions of the PATHMON process, the TMF subsystem, and the terminal control process (TCP).

In the Guardian operating environment, the functions and tasks of an application are performed by processes, which are running programs. A process pair consists of a primary process, which does some specific function in the overall work of the application, and a secondary (backup) process, which remains ready to take over if the primary process fails. During processing, the primary process keeps the backup process informed of what it is doing (for example, sending a request) by means of special interprocess messages, in an activity called checkpointing. Through checkpointing, the backup process has enough information to take over and continue if the primary process fails.

Both the PATHMON process and the TCP can be configured as process pairs to support Pathway applications. When the PATHMON process is configured as a process pair, you are ensured the ability to control and monitor OLTP system operation even if the primary PATHMON process fails. When a TCP is configured as a process pair and the primary TCP fails, terminals controlled by the TCP can still be used.

Pathway server classes provide additional fault tolerance by allowing requests to be rerouted to surviving server processes in a server class if one server process fails.

Besides process pairs and server classes, fault tolerance in a Pathway application is ensured by the PATHMON process, the TCP, and the TMF subsystem. Using information stored in the PATHMON configuration file, the PATHMON process automatically restarts processes at their initialization level after a failure, allowing these processes to resume work immediately.

For requesters written in SCREEN COBOL that use the TMF subsystem, the TCP automatically restarts processing at the transaction boundary (for example, at the BEGIN-TRANSACTION statement) after a failure. In addition to restarting processing, the TCP directs the TMF subsystem to back out any incomplete or partial transaction and restore the database to its pre-failure state of consistency. By both restarting processing at the transaction boundary and directing the TMF subsystem to recover a transaction, the TCP ensures that the application and the database are synchronized and ready to continue processing.

## Other Fundamentals of NonStop™ Himalaya Systems

Besides data integrity and fault tolerance, the Pathway environment also provides the high performance and low cost, system security, scalability, and distributed processing of NonStop™ Himalaya systems.

### High Performance and Low Cost

The more transactions your system can process (preferably without degrading response time), the lower the cost of each transaction. The Pathway environment supports fast response time and high system throughput by allowing:

- Component processes in a Pathway application (for example, requester and server processes) to reside and execute concurrently in different processors of a multi-CPU system or even a network. This is called multiprocessing.
- More than one Pathway application to run in a NonStop™ Himalaya system.
- More than one requester program to execute in the TCP at the same time. This is called multithreading. Multithreading permits the processing of multiple and different transactions concurrently and permits multiple users to perform similar tasks—for example, order entry—simultaneously.

The Pathway environment also supports fast response time and high system throughput by allowing the replication of processes and programs and the distribution of processes. For example:

- The PATHMON process can dynamically create additional copies of server processes at times of peak demand and delete the additional servers when activity slows again.
- You can add copies of requester and server programs to your Pathway application to maintain fast response time when the number of users or terminals increases.
- You can distribute processes such as TCPs and servers close to the resources they manage, reducing interprocess communication time within a network.
- You can distribute requesters and servers to less active processors if peak activity on a particular processor is affecting throughput or response time.

## System Security

The Guardian operating environment includes basic mechanisms for controlling access to files, whether they are data files or program files. Because NonStop™ TS/MP and Pathway/iTS run in the Guardian operating environment, Guardian system security parameters also apply to Pathway users and processes. In addition, you can supplement the security features of the Guardian environment with the Safeguard product, which provides authentication, authorization, and auditing capabilities for Guardian files.

## Scalability

Your organization must be able to expand its transaction processing system as its operations evolve and its technical requirements change. NonStop™ Himalaya systems are expressly designed to support incremental, modular expansion, allowing you to increase the size and processing power of your transaction processing system by:

- Adding hardware and application resources to your existing system
- Linking individual Pathway applications into a single network or adding more Pathway applications to an existing network
- Supporting an open systems architecture in which standards-based networks as well as devices and systems from other vendors can be connected to your NonStop™ Himalaya system

## Distributed Processing

Data communications technology allows organizations to extend their online operations over long distances to form global networks and to support distributed processing. The Pathway environment, in conjunction with the Compaq NonStop™ Kernel operating system, allows you to distribute application processes within a single system. Additionally, NonStop™ TS/MP, NonStop™ TM/MP, and Pathway/iTS, in conjunction with the *Expand* networking software, allow you to spread processes, data, and transactions across a network of NonStop™ Himalaya systems. The coordination of transactions among application servers residing within an Expand network and possibly accessing different resource managers (Compaq *NonStop*™ SQL/MP and Enscribe) is known as distributed transaction processing (DTP).

## Pathway Applications

Pathway applications consist of two types of programs: requester programs and server programs**.** This design allows application logic to be distributed near the resources it manages. For example, presentation services are located near terminal devices or workstations; database logic resides in server programs near that database. Requesters and servers communicate by using the Guardian file system or the message system that is part of the NonStop™ Kernel.

Users interact with your application by using devices and processes controlled by your requester programs. Often these devices are terminals through which the users enter and retrieve transaction data. They might also, however, be intelligent devices such as personal computers, workstations, point-of-sale devices, or automatic teller machines (ATMs). Or, they might be Guardian processes that provide transaction input from a file or other batch medium.

Server processes receive requests from requester processes to access a database to add, retrieve, or modify information. Server processes process request messages and send reply messages with the results of the work on the database.

### Servers and Server Classes

You can write Pathway server programs in C, C++, COBOL85, pTAL, TAL, FORTRAN, or Pascal in the Guardian environment. Alternatively, you can write Pathway server programs in C or COBOL85 in the Compaq *NonStop*™ Kernel Open System Services (OSS) environment; you must program such servers to read the Guardian $RECEIVE file as described in the *Open System Services Programmer's Guide*. In both cases, you configure and manage the servers using the PATHCOM interactive interface or the Pathway management programming interface (based on the Subsystem Programmatic Interface, or SPI) in the Guardian environment.

The same server programs, whether developed in the Guardian environment or in the OSS environment, can be used with several different requester and client interfaces. These interfaces include SCREEN COBOL, the Pathsend procedures, and the RSC/MP interface.

The Pathway environment provides the feature of server classes. A server class is a collection of replicated Pathway server processes. All server processes in a server class provide the same set of functions; that is, they execute the same program.

#### Server Processes

Server processes provide the following benefits:

- Server processes help ensure transaction integrity and, therefore, the integrity of the database.
- Server code can be reused by many requester programs, and you can separate presentation services from database functions.
- You can control which transactions can be performed on your node. You can control the logic of the servers, database names, disk names, and so on.

- In distributed environments, server processes provide high performance by allowing you to use remote servers instead of performing multiple remote I/O operations, placing transaction processing close to system resources.

### Server Classes

Server classes provide the following benefits:

- You can minimize use of system resources—for example, processes and file opens—because server classes are shared and highly utilized.
- You can maximize performance because server classes allow multiple copies of server processe to run concurrently in multiple CPUs.
- Based on configuration settings determined by the system manager or operator, the PATHMON process can dynamically create additional server processes within the server class to maintain acceptable throughput as the workload increases.
- By temporarily freezing and stopping the server class and changing configuration parameters, the system manager or operator can adjust the number of servers that are active at any one time to suit response-time requirements.
- The system manager or operator can balance the workload over multiple processes and across multiple CPUs, which provides fault tolerance in addition to load balancing—if a CPU fails, the server class is still available.

## Requesters

The Pathway application programming environment provides two programming interfaces for requesters:

- The Pathsend application program interface (API), provided in the *NonStop*™ TS/MP product
- The SCREEN COBOL language, provided in the Pathway/iTS product

Requesters written using these two interfaces are briefly described in the following paragraphs. In addition, other Compaq products are available to assist you in writing requesters and clients that communicate with Pathway servers. These products include the RSC/MP product for workstation clients and the Extended General Device Support (GDSX) product for front-end and back-end processes.

Section 2, Designing Your Application, provides additional information about how Pathsend requesters, SCREEN COBOL requesters, RSC/MP clients, and GDSX processes can be used in Pathway applications.

### Pathsend Requesters

The Pathsend procedure calls and the LINKMON process allow Guardian processes to access Pathway server classes. The Pathsend procedures bring the benefits of Pathway server classes to a wide range of requesters, providing flexibility in application design. They also provide high performance for requesters that do not need a complex, multithreaded interface to terminals or intelligent devices. Finally, they provide support for both context-free and context-sensitive servers.

Pathsend requesters are described in the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

## SCREEN COBOL Requesters

SCREEN COBOL requesters, which are compiled by the SCREEN COBOL compiler and then interpreted and executed by the terminal control process (TCP), provide ease of programming if you need to handle large numbers of terminals or intelligent devices or if you need screen-presentation services. The TCP and the SCREEN COBOL language produce a high-quality, manageable application. The TCP provides multithreading of requesters, fault tolerance, terminal device configuration, and operations management so that you do not need to program these features in your application.

The TCP provides the following features:

- Fault tolerance (when used in combination with TMF)
- Automatic retry of I/O operations to a server process if the primary process of a server process pair fails
- Transaction protection through TMF
- Multithreading
- Interpretation of compiled pseudocode for programs written in the SCREEN COBOL language, which offers a simple single-threaded programming environment and a screen management system to drive IBM 3270 terminals and the 6530 family of terminals (652x series, 653x series, and 654x series)
- Special syntax to facilitate message assembly, disassembly, and processing (Pathway intelligent device support, or IDS)
- Access to server classes
- Unsolicited message processing (UMP) support
- Management interfaces (the PATHCOM process and the Subsystem Programmatic Interface, or SPI**)** for TCP configuration and management, terminal configuration and management, process management, error logging, and so on

You can use an Extended General Device Support (GDSX) process as a front-end process to the TCP and SCREEN COBOL requesters to communicate with devices not directly supported by the TCP. Use of the GDSX product is described in the *Extended General Device Support (GDSX) Manual*.

## Client/Server Capabilities

The RSC/MP product brings client/server capabilities to the Pathway environment by allowing you to move requester functions to a workstation. This product allows client programs residing on a workstation to access Pathway server classes in any of three different ways:

- Through a Pathsend requester provided by RSC/MP, which works with the LINKMON process
- Through a special intelligent device support (IDS) requester supplied with RSC/MP, which works with the terminal control process (TCP)
- Through an IDS requester that you develop yourself in the SCREEN COBOL language; this requester works with the TCP

RSC/MP also allows requesters to access Guardian processes directly. To facilitate access to servers and Guardian processes, RSC/MP consists of multiple components within both the workstation and NonStop™ Himalaya system environments.

For further information about RSC/MP, refer to the *Compaq NonStop™ Remote Server Call (RSC/MP) Programming Manual*.

## Other Transaction Processing Environments

The NonStop™ TS/MP product serves as the foundation for open transaction processing on NonStop™ Himalaya systems. In addition to the Pathway environment, NonStop™ TS/MP supports the NonStop™ TUXEDO transaction processing system. The NonStop™ TUXEDO system allows you to develop TUXEDO transaction processing applications to run on NonStop™ Himalaya systems.

You can develop applications that use a combination of modules from the NonStop™ TUXEDO environment and the Pathway environment. In particular, you can write a SCREEN COBOL requester that indirectly invokes the services of a NonStop™ TUXEDO server by using the Pathway to TUXEDO translation server provided with the NonStop™ TUXEDO product. For more information about this translation server, refer to the *NonStop™ TUXEDO System Pathway Translation Servers Manual*.

## Development Tools and Utilities

When you are writing requester and server programs for your Pathway application, a variety of program development tools and utilities are available to you. These tools and utilities allow you to shorten the amount of time it takes to code, debug, and test your programs.

### Programming Languages and Related Tools

You can write application programs for the TNS environment on NonStop™ Himalaya systems in C, C++, COBOL85, SCREEN COBOL, Transaction Application Language (TAL), FORTRAN, and Pascal. You use the Binder product to bind TNS object files for creating executable object files.

You can write application programs for the TNS/R native environment in C, C++, and portable Transaction Application Language (pTAL). You use the native `nld` utility for linking TNS/R native object files and for creating executable object files.

For D40 and later releases, you must use the pTAL compiler and `nld` utility for compiling and linking your TCP user-conversion procedures.

### The *Inspect* Symbolic Debugger

The Compaq Inspect product is the symbolic program debugging tool for NonStop™ Himalaya systems. You can use it interactively to examine and modify the execution of Guardian processes (for example, Pathsend requesters and Pathway servers) as well as SCREEN COBOL requesters. An online help facility is available for all Inspect commands and topics.

Using the Inspect product in a Pathway environment requires the use of two terminals or a terminal emulator with windowing capability. One terminal or window acts as the application terminal, while the second terminal or window acts as a command or Inspect terminal.

Because SCREEN COBOL programs are interpreted by the TCP and therefore are not running directly as Guardian processes, you use the TCP for the Inspect session. To use the TCP for an Inspect session, you set a TCP configuration parameter to support the Inspect product and then issue a PATHCOM command to initiate the Inspect session.

### The SCREEN COBOL Utility Program (SCUP)

The SCREEN COBOL Utility Program (SCUP) provides a means for maintaining libraries of SCREEN COBOL pseudocode. SCUP also provides a means for obtaining information about certain aspects of the code, such as versions and compile dates and times, data sizes, screen sizes, programs called, and code sizes.

SCUP allows you to operate on the SCREEN COBOL object library files without recompiling the source programs. In addition, SCUP allows you to:

- Display information about the library files or about programs in a SCREEN COBOL library

- Control access by a TCP to programs in a SCREEN COBOL library
- Copy programs from one SCREEN COBOL library to another
- Delete programs from a SCREEN COBOL library
- Reclaim file space by compressing SCREEN COBOL library files
- Convert a group of programs in a SCREEN COBOL library into a web client

SCUP is described in the *Compaq NonStop™ Pathway/iTS SCUP Reference Manual*.

## The Pathmaker Application Generator

The Pathmaker product helps you create Pathway applications consisting of requester programs written in SCREEN COBOL and server programs written in C or COBOL85. To create applications with the Pathmaker product, you:

- Enter information about your application into a series of screen-based entry forms, which the Pathmaker product then stores in a catalog
- Use the text editor TEDIT to create source files containing C or COBOL85 service code

At your command, the Pathmaker product uses the information from the catalog and the TEDIT file to generate SCREEN COBOL requester code, C or COBOL85 server code, and command files to configure and start the finished Pathway environment for testing.

The Pathmaker product simplifies the creation of Pathway applications by:

- Generating application code in a uniform structure for all requesters and servers, to help simplify maintenance and modification
- Producing program statements for tasks that are specific to Pathway
- Automatically generating TMF statements in your requester programs when you indicate that you want your programs to have TMF protection
- Providing a central location for most application information
- Creating error-handling code for the most commonly encountered errors
- Letting you simulate application screens and navigate from one application screen to another before you write a single line of code

Applications developed with the Pathmaker product can access data from databases managed by either the NonStop™ SQL/MP relational database management system or the Enscribe database record manager. If you are using Pathsend requesters, or clients that use RSC/MP, you can use the Pathmaker tool to create prototype servers.

## The Enable Product

The Enable product, used with the Enscribe database record manager, is a tool that allows you to develop simple data management applications without using a conventional programming language. You can use the Enable product to generate SCREEN COBOL requester programs that use a generic server program provided with the Enable product to record, maintain, or retrieve information stored within a single database file or multiple database files. The Enable application performs these database operations on a record-by-record basis and on one database file at a time. Finally, you can use the Enable product to produce a PATHCOM command file to execute the Enable application in the Pathway environment.

The Enable product reduces the amount of time needed to develop a simple application, thereby decreasing application development costs. The Enable product, although not as powerful as the Pathmaker product, allows you to:

- Control the format of the screen displayed by the application
- Limit the types of operations (delete, insert, read, or update) that the application can perform on a database file
- Define a method that the application uses to ensure the integrity of a database file

Although applications generated by the Enable product may lack the sophistication of custom-designed application programs (Enable applications cannot perform either mathematical calculations or ensure database consistency), you can quickly generate Enable applications to meet immediate processing needs. Among its many uses, you can use an Enable application as a prototype for a more complex application, a data entry program, or a tool to maintain a small database. Enable applications can be readily integrated into the Pathway environment.

## Client/Server Development Tools

As mentioned earlier, the RSC/MP product facilitates client/server computing, allowing workstation applications to access Pathway servers. A number of packaged tools and utilities are commercially available for use with RSC/MP.

## Transaction Processing Scenarios

This subsection provides two examples of how transactions from SCREEN COBOL requesters are processed. The two scenarios illustrate the following:

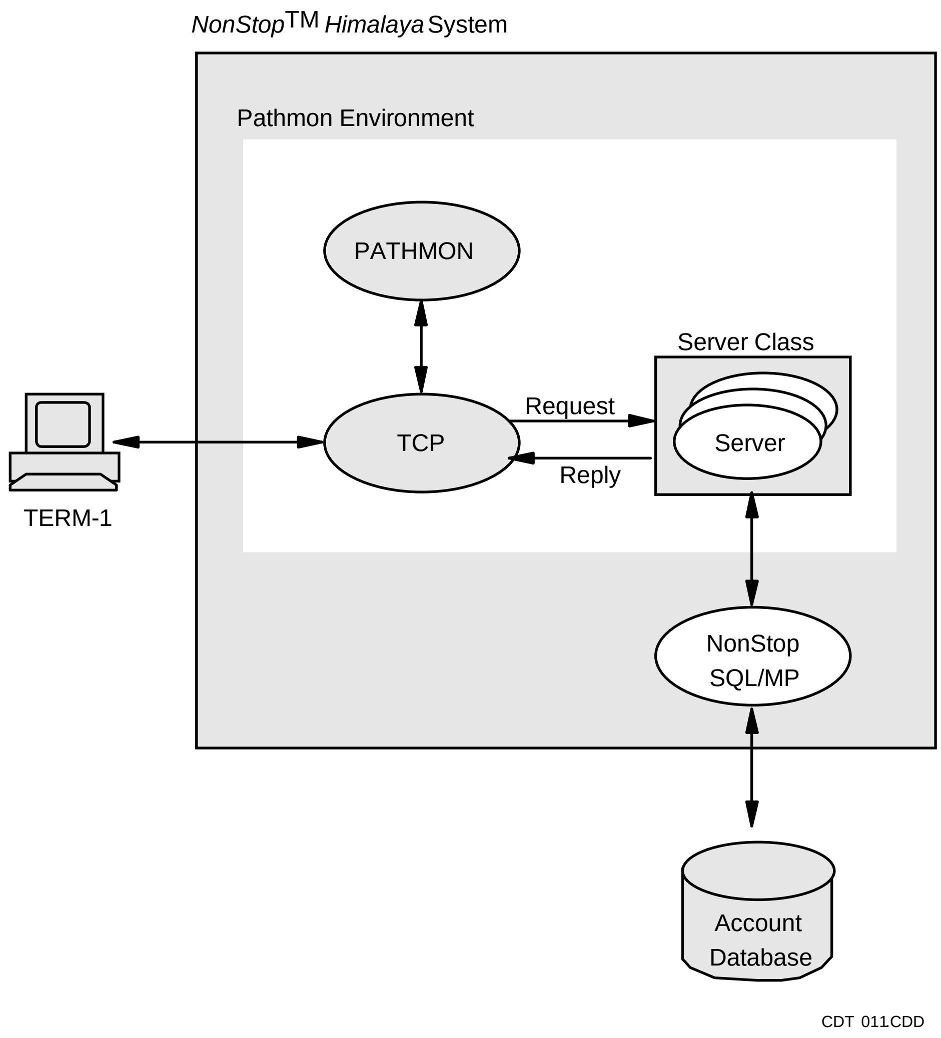
- A transaction from a Pathway/iTS terminal to a Pathway server
- A transaction from an intelligent device to a Pathway server

## Transaction From a Terminal

Figure 1-1 illustrates the path of a transaction from a Pathway/iTS terminal, which is controlled by a SCREEN COBOL program executed by the TCP, to a Pathway server. For this example, consider a clerk at an order entry office who must update customer information for account number 1234567.

1. The clerk displays the order screen, a data entry screen, on a terminal. To the Pathway application, the terminal is defined as object TERM-1.
2. The clerk enters the account number in the appropriate field and requests an update to customer information by pressing a function key.
3. The requester, which is the TCP interpreting a SCREEN COBOL program, checks the input data, confirming that the account number has no more than seven characters. The requester then displays a new screen showing the customer information as it is currently recorded in the database.
4. The clerk enters the new information (for example, a new customer address) in the appropriate field and requests that the information be updated by pressing a function key.
5. The requester checks the input data for validity and confirms that there are no input errors.
6. The SCREEN COBOL program formats a request message containing the name of the server class and the data needed by the server to complete its work. The TMF transaction begins.
7. The SCREEN COBOL program executes a SEND statement, directing the request message to be sent to the specified server class.
8. If the TCP does not have a link to the specified server class, the TCP asks the PATHMON process for a link to a server process in the server class. The PATHMON process replies that a server process is available. If the TCP already has a link to the server class, this step is not performed.
9. The TCP forwards the request to the server process by using the interproces communication mechanism of the NonStop™ Kernel operating system.
10. The server process receives and reads the request message.
11. Executing NonStop™ SQL/MP statements in its program, the server process accesses the database, using the account number as the key, and updates the specified customer information.
12. The server process formats a reply message verifying the database update and replies to the TCP using the interproces communication mechanism of the NonStop™ Kernel operating system.
13. The TCP receives, interprets, and then forwards the reply message to TERM-1. The TMF transaction ends.The SCREEN COBOL requester program displays a message on the terminal screen verifying that the specified information has been updated.

**Figure 1-1. Example of a Pathway/iTS Terminal Requester**



## Transaction From an Intelligent Device

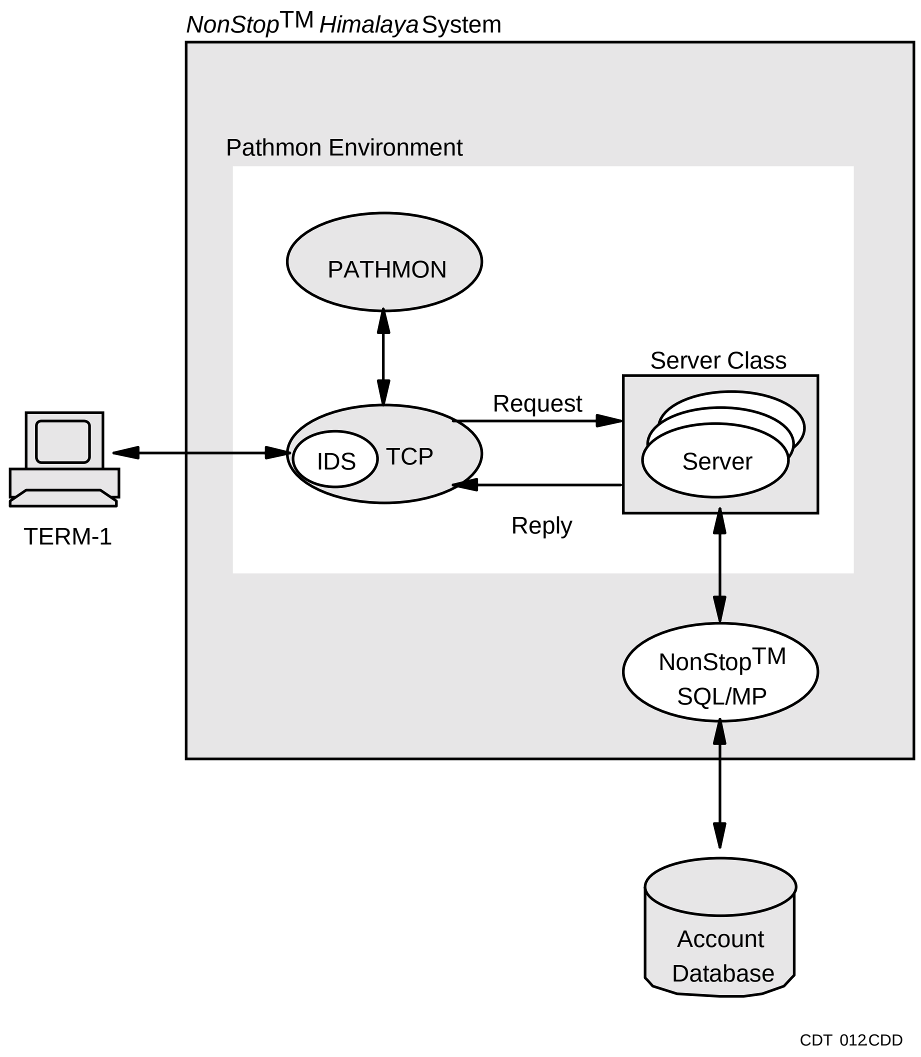
Figure 1-2 illustrates the path of a transaction from an intelligent device, communicating with a SCREEN COBOL program that uses the IDS facility of the TCP, to a Pathway server. For this example, consider again a clerk at an order entry office. Using a workstation, the clerk must update customer information for account number 2345678.

1. The clerk displays the order screen, a data entry screen, on a workstation. To the Pathway application, the terminal is defined as object TERM-1.

2. The clerk enters the account number in the appropriate field and requests an update to customer information by pressing a function key.

3. The workstation application checks the input data, confirming that the account number entered has no more than seven characters. The workstation application then displays a new screen showing the customer information as it is currently recorded in the database.
4. The clerk enters the new information (for example, a new customer telephone number) in the appropriate field and requests that the information be updated by pressing a function key.
5. The requester checks the input data for validity and confirms that there are no input errors.
6. The workstation application formats a request message containing the name of the server class and the data needed by the server to complete its work. The workstation application—with the aid of user-developed conversion procedures—converts its data to SCREEN COBOL format, a representation acceptable to the TCP.
7. The workstation application executes a SEND statement, directing the request message to be sent to the specified server class. A user-developed communications subsystem forwards the request to the TCP using a supported protocol. The IDS requester, which is the TCP interpreting a SCREEN COBOL program, receives the request message as part of an IDS SEND MESSAGE statement. The TMF transaction begins.
8. If the TCP does not have a link to the specified server class, the TCP asks the PATHMON process for a link to a server process in the server class. The PATHMON process replies that a server process is available. If the TCP already has a link to the server class, this step is not performed.
9. The TCP forwards the request to the server process by using the interproces communication mechanism of the NonStop™ Kernel operating system.
10. The server process receives and reads the request message.
11. Executing NonStop™ SQL/MP statements in its program, the server process accesses the database, using the account number as the key, and updates the specified customer information.
12. The server process formats a reply message verifying the database update and replies to the TCP using the interproces communication mechanism of the NonStop™ Kernel operating system.
13. The TCP receives, interprets, and then forwards the reply message to TERM-1 using the IDS SEND MESSAGE statement. The TMF transaction ends.
14. The workstation application displays a message on the workstation screen verifying that the specified information has been updated.

**Figure 1-2. Example of an IDS Requester**



# 2 Designing Your Application

To develop a functioning Pathway application, you must identify the individual transactions in your business operations, design and build the application database, and design and code requester programs and server programs.  This section describes the design of transactions and databases for Pathway applications and the design of requester and server programs.

To explain these application design tasks, this section uses as an example an application that processes sales orders for a distributorship.  The example shows how the Pathway environment can be used to create an OLTP application that supports the distributorship's order-processing operations.

The distributorship in the example has three offices linked by telecommunications:

- \CORP is a network node at corporate headquarters where the purchasing, accounts receivable, and accounts payable functions are managed.
- \WHS is a network node in a warehouse where the inventory, shipping, and receiving functions are performed.
- \REG is a network node in a sales office that is responsible for processing all customer orders in a particular geographic region.  Order-processing functions consist of entering orders as input and maintaining records of each order.  To perform these two functions, the order processing group:
    - Checks with inventory control to determine if items to be ordered are in stock
    - Sends inventory control shipping and ordered-items information about each order
    - Gets customer credit information from accounts receivable
    - Sends billing information to accounts receivable
    - Answers customer inquiries about order status
    - Records complete information about each order in the database

## Designing Transactions

The first step in developing a Pathway application is to identify and define the transactions that your application will process.  To do this, you isolate the business tasks you plan to automate, analyze the flow of information within those tasks, list the transactions that result from the analysis, and then identify the various components of the transactions.  After these tasks are performed, you protect each transaction, and therefore the integrity and consistency of the database, with the Transaction Management Facility (*TMF*) subsystem.

## Analyzing Data Flow

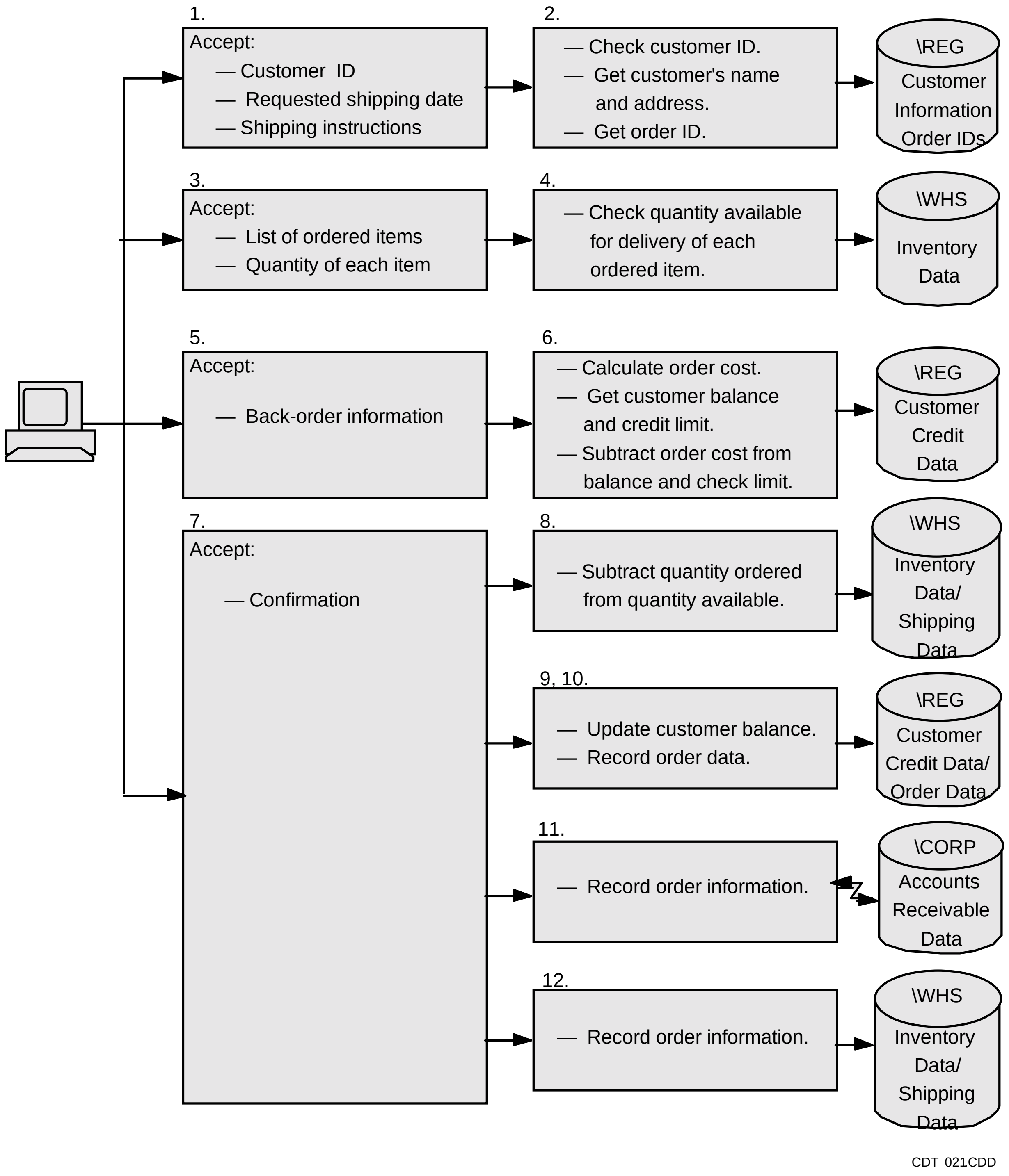
Analyzing the flow of data involves identifying what information is required for a business task, determining the order in which that information is required, and specifying how the information is to be handled. To automate the order-processing tasks of the previously described distributorship, for example, you could analyze the flow of information as follows:

1. Accept the customer’s identification number, a requested delivery date for the order, and shipping instructions such as the delivery address.
2. Check the customer’s identification number to ensure that the customer is defined in the \REG database; get the customer’s name and address from the \REG database; and get a new order identification from the \REG database.
3. Accept a list of order items along with the requested quantity for each order item.
4. Check the current quantity available, in the database on \WHS, of each ordered item to ensure that sufficient quantity exists to fill the order.
5. Accept any special instructions, such as back-ordering out-of-stock items, required to process the order.
6. Calculate the total order cost; get the current customer balance and credit limit from the \REG database; add the total order cost to current customer balance; and ensure that the new balance does not exceed the customer’s credit limit.
7. Ask the customer to confirm the order.
8. After the customer has confirmed the order, subtract the quantity ordered from the current quantity available, in the \WHS database, for each ordered item.
9. Add the total order cost to the customer’s current balance in the \REG database.
10. Record the order information in the \REG database.
11. Transmit the order information in the accounts receivable files to the \CORP database and record the information in the database.
12. Record the order shipping information in inventory files on the \WHS database.

Assume that your analysis of the previous flow of information shows that only two transactions need to be created to support order processing: an Add New Customers transaction and an Enter Sales transaction. The Enter Sales transaction, which accepts and records all the information associated with a customer order, is the example used in the rest of this section.

The data flow outlined in the previous steps is illustrated in Figure 2-1.

**Figure 2-1. Data Flow for a Business Task**


1.
Accept:
— Customer ID
— Requested shipping date
— Shipping instructions
2.
— Check customer ID.
— Get customer's name and address.
— Get order ID.
\REG
Customer Information Order IDs
3.
Accept:
— List of ordered items
— Quantity of each item
4.
— Check quantity available for delivery of each ordered item.
\WHS
Inventory Data
5.
Accept:
— Back-order information
6.
— Calculate order cost.
— Get customer balance and credit limit.
— Subtract order cost from balance and check limit.
\REG
Customer Credit Data
7.
Accept:
— Confirmation
8.
— Subtract quantity ordered from quantity available.
\WHS
Inventory Data/ Shipping Data
9, 10.
— Update customer balance.
— Record order data.
\REG
Customer Credit Data/ Order Data
11.
— Record order information.
\CORP
Accounts Receivable Data
12.
— Record order information.
\WHS
Inventory Data/ Shipping Data
CDT 021CDD

## Identifying Transaction Components

After you have identified the Enter Sales transaction for the order-processing application, you list the functions performed by the transaction and group them either into data collection and validation operations or into database update operations. For example, the key functions performed by the Enter Sales transaction during data collection and validation are:

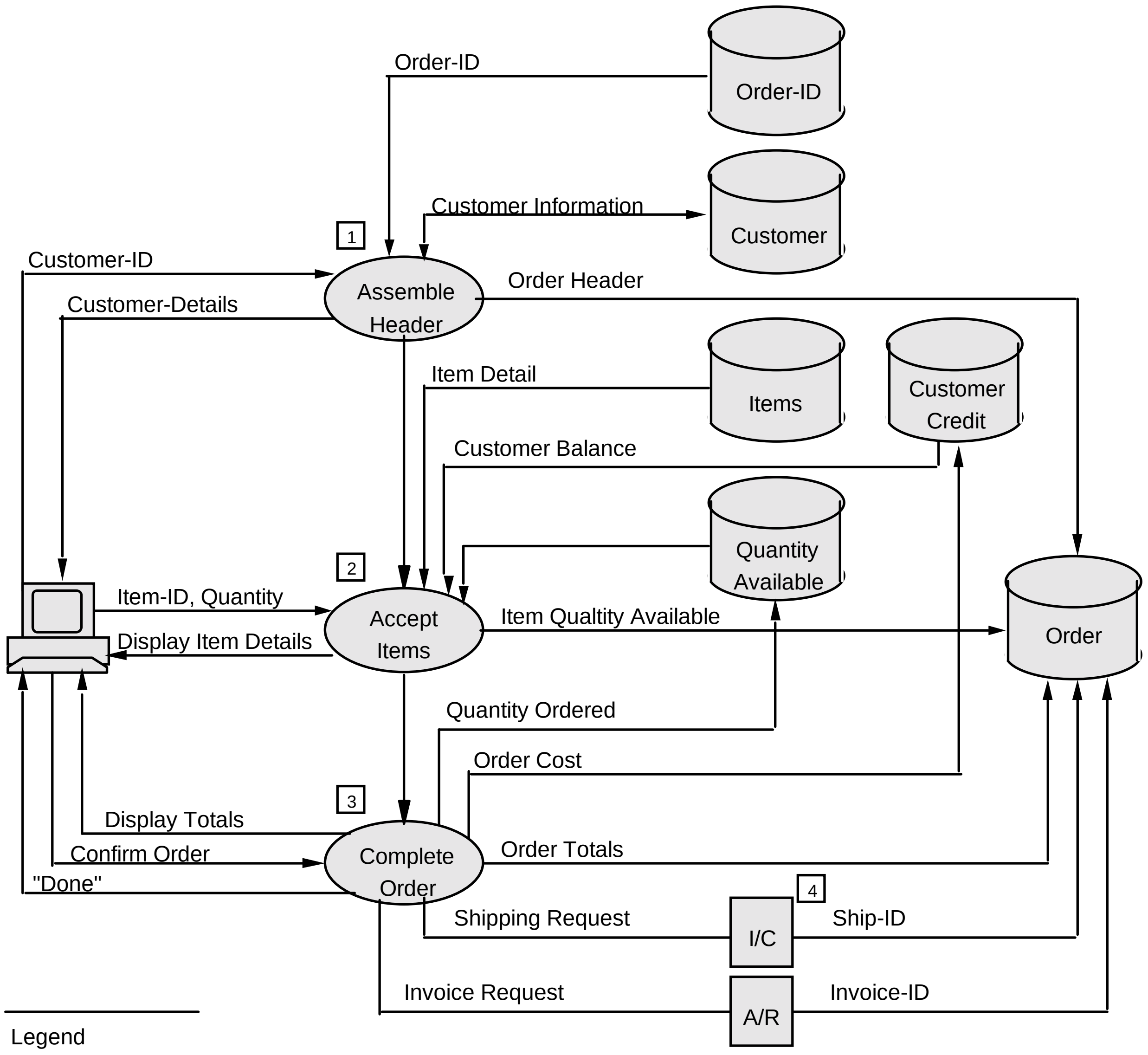
- Assembling information for the order header, including:
    - Obtaining the order-ID
    - Accepting the customer-ID
    - Accepting the requested delivery date
    - Accepting shipping instructions
    - Checking the customer-ID
    - Obtaining the customer’s name and address from the database
- Assembling the order, including:
    - Accepting the list of order items and the quantity of each item
    - Checking the current quantity available for each item ordered
    - Accepting special instructions
    - Calculating total order cost
    - Obtaining the customer’s balance and credit limit from the database
    - Adding the total cost to the customer’s balance and ensuring that it does not exceed the credit limit

The key function performed by the Enter Sales transaction during database update operations is order completion. The order completion function includes:

- Subtracting the quantity ordered from the current quantity available for each ordered item
- Adding the total order cost to the customer’s current account balance
- Recording the order in the database
- Recording the order invoice in the accounts receivable files
- Recording order shipping information in the inventory files

The relationships of the various functions for the Enter Sales transaction are illustrated in Figure 2-2. The dark arrows in the figure show the sequence of actions from Step 1 through Step 3. The lighter arrows show the flow of information.

**Figure 2-2. Relationships Between Transaction Functions**



Legend

1 Assemble information for order header; display at terminal and add to database; optionally change customer information.

2 Accept items in order, check item availability and customer credit; display item details at terminal and add to database.

3 Display totals at terminal and get confirmation; update item quantity and customer balance; add totals to database; inform related applications about order.

4 Later, when order is shipped and customer billed, add shipping and invoice numbers to database.

CDT 022CDD

## Protecting Transactions

After listing and grouping the components of the Enter Sales transaction, you protect the integrity of each transaction, and ultimately the consistency of the database, with the TMF subsystem. The following pages outline how to integrate the TMF subsystem with your business transactions.

For details about TMF programming in SCREEN COBOL requesters, see Section 5, Managing Transactions With the TMF Subsystem For information about the overall features of the TMF subsystem, including database file recovery and audit trails, refer to the *Introduction to NonStop™ Transaction Processing*.

### Defining TMF Transactions

From a systems perspective, a transaction includes all the steps necessary to transform a database from one consistent state to another. A TMF transaction must be constructed as a logical unit of work: that is, all parts of a transaction, which usually consists of multiple operations, must be handled as a single entity. If any parts of a TMF transaction are not successfully completed or applied to a database, then none of the transaction parts are applied to the database. By forcing all components of a transaction to be handled as a single unit of work, the TMF subsystem prevents inaccurate or partial updates to the database and protects database consistency.

At the application level, a TMF transaction is defined by special procedure calls or statements that specify the beginning and end of a transaction. For example, in SCREEN COBOL, a transaction begins with a BEGIN-TRANSACTION statement and ends with an END-TRANSACTION or ABORT-TRANSACTION statement. The procedure calls that define TMF transactions act as brackets; that is, the statements are placed before and after the add record, update record, and delete record procedures in your requester program.

### Database Consistency and Concurrency

Potentially, all operations that alter the database are candidates for TMF protection. But before you can apply TMF protection to your transactions, you need to determine:

- When to begin a TMF transaction
- Whether all of the database update operations have to happen together in the same TMF transaction or whether they can be parts of different transactions

To answer these issues, you have to establish your criteria for database consistency and decide how much processing concurrency you can achieve in the application. For example, the Enter Sales transaction affects several pieces of information: order data, inventory data, shipping data, customer credit, and receivables. Upon examination of this transaction, you will see that it is possible to make one general assertion about order processing and about the Enter Sales transaction in particular: An order is not complete until every piece of information associated with the order is recorded in the \REG, \CORP, and \WHS databases.

To illustrate this assertion, consider a situation where a transaction fails after it changes the customer's balance, records the order information, and records the order invoice, but before it records the shipping information. In this scenario, the customer is going to be billed for an order never received. Consequently, your basic criterion for database consistency is as follows: all database updates that are related to the order must be part of one TMF transaction.

Any record modified or inserted by a database operation that is protected by the TMF subsystem is locked and unavailable to other transactions until the initial transaction ends successfully. This type of locking protocol means that you always have a design tradeoff—consistency versus concurrency—with respect to locking records that are actively accessed by the application. If records are locked too early, other transactions cannot access them and the application’s concurrency (its ability to process many transactions at the same time) suffers.

As the Enter Sales transaction demonstrates, all of the data collection and validation operations can happen before you begin the TMF transaction—although some revalidation may be done again as part of the transaction. Assembling the order header and assembling the order involve reading records in the database but not changing the records. The rest of the operations change the database and should all be done within a TMF transaction.

As a general rule, you should design the application’s transactions to maintain consistency under all circumstances. After the application is installed and running successfully, you can look for ways to improve its concurrency.

## Aborting Transactions

If the requester or the server program detects a problem during the processing of a TMF transaction, the requester or server causes the transaction to be aborted with a special statement or procedure call (for example, an ABORT-TRANSACTION statement in a SCREEN COBOL program). For requesters, the statement that aborts a transaction is executed in lieu of the statement that ends a transaction; for example, in a SCREEN COBOL program the requester either completes the transaction with an END-TRANSACTION statement or causes it to be backed out, because of an error, with an ABORT-TRANSACTION statement.

In the past, program designs typically assigned the task of aborting transactions to requesters. Current program design often assigns that task to servers. Servers abort transactions and inform the requesters of those actions, thus ensuring protection of data. The aborting of transactions by servers is described further under “Designing Server Programs” later in this section.

The TMF subsystem backs out aborted transactions by using information contained in the TMF audit-trail files. For more information about transaction backout and audit-trail files, refer to the *NonStop™ TM/MP Application Programmer’s Guide*.

## Designing the Database

The next step in developing a Pathway application is to design the database that will be accessed and updated by the application. Designing the database, which is a highly specialized activity typically performed by experienced database administrators, involves:

- Precisely identifying the meaning and use of the data as it exists in your business and specifying the database files and records that will store this data. This step is referred to as logical design.
- Choosing file types and keys for the records. This step is referred to as physical design.

In addition to completing a logical and physical design of your database, you must also select a database manager and ensure that your server programs can interface with that database manager.

### Logical Design

During the logical design process, you determine which classes of data must be maintained by your application and identify the relationships that exist between the classes. Each class of data names something that the database will store information about. For example, in an application that processes sales orders, *`orders`* is a class of data and *`order-items`* is a relationship between a particular order and the inventory items within the order. These data classes and relationships generally become records in files accessed by the application.

After specifying data classes, you list the attributes (data items) for each class of data. For example, some of the attributes are *`order-ID`*, *`cust-ID`*, and *`order-total`*. These attributes become fields in the records of the database. After specifying attributes for data classes, you diagram the relationships between each of the files in the database and then normalize your database files. To normalize files is to ensure, at a minimum, that:

- There are no repeating fields.
- Data is dependent on the entire key (a unique element) of a field.
- Data is dependent on nothing but the key.

## Physical Design

You undertake the physical design of your database by selecting the appropriate file types and record keys for each of the files in the database. Whether you are using the Compaq *NonStop*™ Structured Query Language/MP (SQL/MP) software or the Enscribe software as your database management system (DBMS), these file types can be classified as key-sequenced, relative, entry-sequenced, or unstructured:

| | |
|---|---|
| Key-Sequenced | Each record in the file has a primary key and up to 255 alternate keys. The primary key is a field or combination of fields within the record. |
| Relative | Each record in the file has a unique record number, which is the primary key, and can have up to 255 alternate keys. The record number is a unique value that corresponds to the physical location of the record within the file. |
| Entry-Sequenced | Each record in the file has a unique record number and can have up to 255 alternate keys. The record number corresponds to the order in which a record is stored in the file. The primary key is the relative byte address of the record. |
| Unstructured | Each record in the file has a unique record number that can be used as the primary key. Alternate keys are not supported. |

Although the file type you choose depends on your application requirements, generally you should choose key-sequenced files for a database that will be accessed and maintained by a Pathway application. Key-sequenced files provide more flexibility than the other file types.

## Database Managers

Databases supporting Pathway applications can run under either the NonStop™ SQL/MP relational database management system or the Enscribe database record manager. Both of these products support the creation and use of large databases capable of operating in local or distributed systems.

The NonStop™ SQL/MP product is both a database management system (DBMS) for production environments and a relational database management system (RDBMS) for decision-making in an information-center environment. The NonStop™ SQL/MP product allows you to think about and represent files in the database as a collection of similarly structured lists. For more information about designing NonStop™ SQL/MP databases, refer to the *Compaq NonStop™ SQL/MP Reference Manual*.

The Enscribe database record manager provides a record-at-a-time interface between Pathway servers and your database. For more information about designing Enscribe databases, refer to the *Enscribe Programmer’s Guide*.

## Remote Duplicate Database Facility (RDF)

If disaster recovery of your database is important, the Remote Duplicate Database Facility (RDF) is available to maintain a copy of the database on a remote system. The RDF product monitors database updates audited by the TMF subsystem and applies those updates to the remote copy of the database. For more information about the RDF product, refer to the *Remote Duplicate Database Facility (RDF) System Management Manual*.

## Designing Requester Programs

To facilitate the accessing of Pathway server classes from different transaction sources, you can develop requester programs for a Pathway application that use any of the following access approaches:

- SCREEN COBOL and the TCP
- SCREEN COBOL and the TCP with the intelligent device support (IDS) facility
- The Pathsend procedure calls
- The Compaq *NonStop*™ Remote Server Call/MP (RSC/MP) product
- The Extended General Device Support (GDSX) product

In Table 2-1, key technical and business considerations are mapped to each way of accessing Pathway servers. More information about each approach is provided following the table.

**Table 2-1. Considerations for Requester Programs**

| Server Access Approach | Large Number of I/O Devices | Support for Intelligent Devices | Multi-Threading Capability | High Performance | Ease of Development | Fault Tolerance | TMF Support | Support for Context Sensitivity |
|---|---|---|---|---|---|---|---|---|
| TCP | X | | X | | X | X | X | |
| TCP with IDS | X | X | X | | X | X | X | |
| Pathsend | | | X | X | | | X | X |
| RSC/MP | X | X | X | X | X | | X | |
| GDSX | X | X | X | X | | X | | X |

## SCREEN COBOL Requesters

Screen programs for Pathway terminals perform a variety of front-end functions for your Pathway application and are typically written as single-threaded programs in the SCREEN COBOL language. This language offers a simple programming environment and screen-management system to drive 65xx terminals and IBM 3270 terminals. SCREEN COBOL supports both conversational mode (for either block-mode or conversational-mode terminals) and intelligent mode (for intelligent devices and communications lines).

When you write a screen program in SCREEN COBOL, you can take advantage of the features of the Compaq *NonStop*™ Pathway/iTS TCP. As supplied by Compaq, the TCP supports:

- Fault tolerance
- TMF transactions
- Multitasking of single-threaded screen programs
- Access to server processes with Pathway server classes
- Unsolicited message processing (UMP)
- System management interfaces (that is, PATHCOM or the Pathway management programming interface)

SCREEN COBOL requester programs do not perform any file I/O operations except to terminals and server classes. A file I/O operation to a server class, which is in the form of a request message, is initiated by the requester program by using the SCREEN COBOL SEND statement.

### Programming Tasks

Most Pathway applications comprise five types of screen program units that are linked by some type of calling sequence. Each program unit allows the user to perform one specific type of action. The program unit types are:

| | |
|---|---|
| Logon | Allows users to gain access to the application |
| Menu | Allows users to select applications or particular application functions |
| List-only | Allows users to select line items (for instance, an inventory item and its price) for processing |
| Data entry | Allows users to add, delete, and update specific data |
| Help | Assists users in responding to application screens |

Another type of program unit often used is a router program. This is a special type of program used to route communication (calls) between the other program units by using a hierarchy called flat-tree design. This design facilitates random screen manipulation by terminal users. A flat-tree design is typically two layers deep: the first layer is the router and the second layer consists of all the other programs.

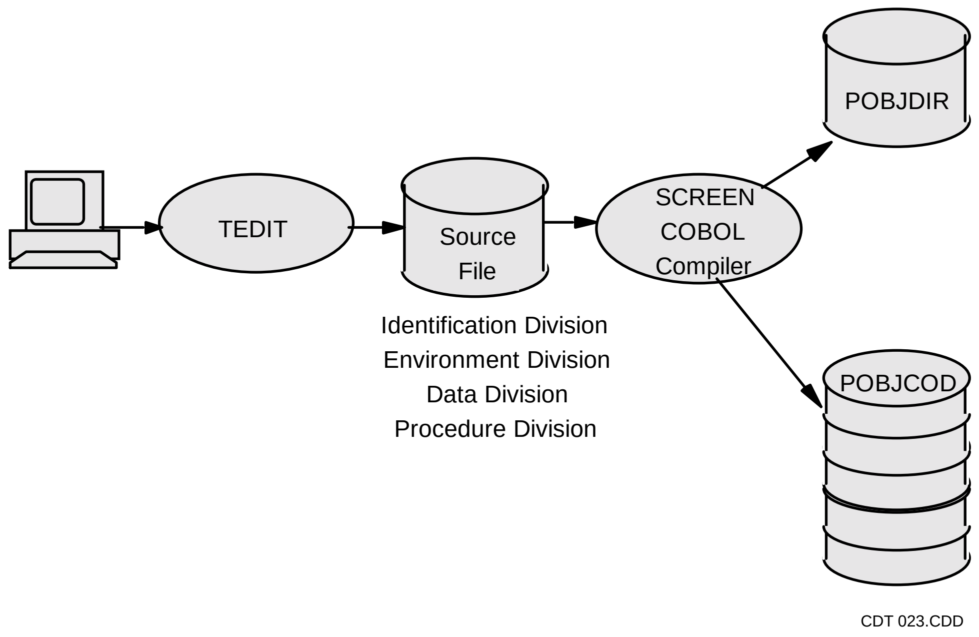
As a programmer, your task is to create each required program unit. For each program unit, you use the text editor TEDIT to create a screen program as a SCREEN COBOL source file. You then use the SCREEN COBOL compiler to read the source file and create:

- A program label entry in the SCREEN COBOL library directory (*DIR)
- Pseudocode (code that is interpreted by the TCP) in the SCREEN COBOL library code file (*COD)

Once the pseudocode resides in *COD, it is immediately usable by the TCP, which reads programs from the library file, interprets them, and executes them on behalf of the terminals logically attached to the TCP. The SCREEN COBOL library files may contain copies of many different screen programs.

Figure 2-3 illustrates the tasks and components involved in the creation of a SCREEN COBOL requester program.

**Figure 2-3. Creating SCREEN COBOL Requester Programs**



## Program Structure

The logon, menu, list-only, data entry, and help program units each consist of four required divisions. These divisions are very similar to standard COBOL divisions, except that the Data Division in a SCREEN COBOL program contains a section for screen description and entry formatting, and there are no file descriptions in a SCREEN COBOL program because the requester does not access data files.

- The Identification Division identifies the program unit to the SCREEN COBOL compiler. It contains one required paragraph and five optional paragraphs.

- The Environment Division declares the operating environment of the program unit and optionally allows modification of the TCP's error-reporting operations. It contains one required section (the Configuration Section) and one optional section (the Input-Output Section).
- The Data Division defines the program data structures by their format and usage. Both the Working-Storage Section, which describes data local to the program, and the Screen Section, which describes data displayed on and accepted from a terminal, are required.
- The Procedure Division includes all the processing steps for the program. The steps are organized into SCREEN COBOL statements and sentences and grouped into sections, paragraphs, and procedures.

The example of a SCREEN COBOL program structure in Example 2-1 illustrates a program unit containing the four required divisions previously described. This program unit, which is in outline form, would handle data entry for the order-processing application introduced at the beginning of this section.

**Note.** The program in Example 2-1 illustrates program structure only; it is not a complete program. For examples of complete, running SCREEN COBOL requester programs, refer to the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

**Example 2-1. Sample SCREEN COBOL Requester Program Structure** (page 1 of 2)

```
IDENTIFICATION DIVISION.         Declares program unit name.

  PROGRAM-ID.  ORDER

ENVIRONMENT DIVISION.            Defines type of terminals
                                    this program unit will
  OBJECT COMPUTER.                  control.
   TERMINAL IS T16-6530
  SPECIAL NAMES.                    Defines special names for:
   ENTER F1,. . .                   1.  Function key names
  BRIGHT IS BRIGHT                  2.  Video attributes
  PROTECTED IS PROTECTED            3.  Flow control data
                                        attributes

DATA DIVISION.

  WORKING-STORAGE SECTION.          Declares structure of
   01 WS-ORD-MSG                    server messages, variables
         .                          passed to called program
         .                          units, and local variables
         .                          used to manage screens and
                                    server communication.
   01 SERVER-REQ-FUNCTION-CODE
         .
         .
         .

  LINKAGE SECTION.                  Declares structure of
   01 LOGON-INFO                    variables passed from
         .                          calling program units.
         .
         .

  SCREEN SECTION.                   Declares format of all
   01 ORDER-SCREEN                  screens managed by this
         .                          program unit, including
         .                          screen fields and video
         .                          attributes of fields.
```

**Example 2-1. Sample SCREEN COBOL Requester Program Structure** (page 2 of 2)

```
PROCEDURE DIVISION.

  MAIN SECTION.
   PERFORM 0100 START
         .
         .
         .

  SCREEN MANAGER SECTION.              Displays operator screen
   DISPLAY ORDER-SCREEN                and accepts data from
   ACCEPT ORDER-SCREEN UNTIL           screen.

  SERVER MANAGER SECTION.              Sends requests to server,
   0200 MOVE ORDER TO. . .             handles server reply,
   BEGIN-TRANSACTION                   and commits transaction.
   SEND ORDER-MSG TO. . .
         .
         .
         .
   REPLY CODE 0 YIELDS. . .
   END-TRANSACTION                     Specifies END-TRANSACTION
                                       unless results of SEND
                                       require ABORT-TRANSACTION.
```

## Unsolicited Message Processing

The unsolicited-message processing (UMP) feature of Pathway/iTS makes it possible for terminals running SCREEN COBOL requesters to accept and reply to unsolicited messages sent to them by *Guardian* operating environment processes outside of the Pathway environment. These external processes can reside anywhere within a Compaq *Expand* network. Guardian processes send unsolicited messages to terminals through their controlling TCP. Such messages consist of an UMP header, which gets interpreted by the receiving TCP, and the body of the message, which gets passed to the SCREEN COBOL requester program running at the specified terminal.

Each Pathway/iTS terminal has its own unsolicited-message queue. When the TCP receives an unsolicited message addressed to one of its terminals, it places the message in the appropriate queue.

To support the processing of unsolicited messages, you code specific SCREEN COBOL clauses, statements, and registers in your requester programs. UMP works as follows:

1. A requester detects the arrival of an unsolicited message by testing the contents of its PW-UNSOLICITED-MESSAGE-QUEUED special register, by performing a RECEIVE UNSOLICITED MESSAGE statement as a waited input operation, or by including an ESCAPE ON UNSOLICITED MESSAGE clause in an ACCEPT or SEND MESSAGE statement.

2. Requesters obtain the text of an unsolicited message by performing a RECEIVE UNSOLICITED MESSAGE statement.

3. After constructing an appropriate response, requesters reply to an unsolicited message by performing a REPLY TO UNSOLICITED MESSAGE statement.

## IDS Requesters

Standard SCREEN COBOL requesters are screen oriented; they send data back and forth between the Working-Storage Section of the program and a terminal's display screen by using screen templates defined in the Screen Section. Standard SCREEN COBOL requesters use SCREEN COBOL ACCEPT and DISPLAY statements in the Procedure Division to interact with display terminals.

SCREEN COBOL requesters that employ the intelligent device support (IDS) facility within the TCP send data back and forth between the Working-Storage Section and an intelligent device (or a front-end process that controls the device) by using message templates defined in the Message Section in the Data Division. IDS requesters use SCREEN COBOL SEND MESSAGE statements and their associated REPLY clauses in the Procedure Division to interact with the intelligent devices or front-end processes.

Although the IDS facility sends and receives data through Message Section templates instead of Screen Section templates, the TCP still provides:

- Link management for access to Pathway server classes
- TMF support to ensure transaction protection and database integrity
- Fault tolerance through process pairs
- Multithreading and multitasking
- Expanded I/O editing support for data streams from intelligent devices

### Design Considerations

When using IDS to facilitate access to Pathway servers by intelligent devices, consider the following:

- IDS requester programs are written and compiled in the same way as standard SCREEN COBOL requesters.
- A controlling SCREEN COBOL program unit does not control the intelligent device or front-end process or use any information about the characteristics of the device. The programming within the device or process must start the device or process itself, accept messages from the TCP, and determine if and when to reply to the TCP.
- The intelligent device supplies the presentation services (that is, screen displays) suitable to its capabilities.
- Terminal types not supported by the TCP can use a front-end process (such as a GDSX process) in conjunction with IDS, as described under Requesters Using GDSX later in this section.

## Program Structure

The example in Example 2-2 outlines an IDS SCREEN COBOL requester program. Except for the inclusion of the Message Section and the deletion of the Screen Section, the structure of an IDS requester program is the same as that of a standard SCREEN COBOL requester program.

**Example 2-2. Sample IDS Requester Program Structure** (page 1 of 2)

```
IDENTIFICATION DIVISION.

  PROGRAM-ID.  ORDER

ENVIRONMENT DIVISION.

  OBJECT COMPUTER.
   TERMINAL IS INTELLIGENT

DATA DIVISION.

  WORKING-STORAGE SECTION.
   01 WS-ORD-MSG
          .
          .
          .

   01 SERVER-REQ-FUNCTION-CODE
          .
          .
          .

  LINKAGE SECTION.
   01 LOGON-INFO
          .
          .
          .

  MESSAGE SECTION.                       Declares structure of
   01 MSG-FORMAT                         data passed to or from
          .                              a device or process.
          .
          .
```

**Example 2-2. Sample IDS Requester Program Structure** (page 2 of 2)

```
PROCEDURE DIVISION.

  MAIN SECTION.
   PERFORM 0100 START
         .
         .
         .

  DEVICE HANDLING SECTION.
   SEND MSG MSG-FORMAT                Sends messages to and
                                      receives messages from
                                      intelligent device.

  SERVER MANAGER SECTION.
   0200 MOVE ORDER TO. . .
   BEGIN-TRANSACTION
   SEND ORDER-MSG TO. . .
   REPLY CODE 0 YIELDS. . .
   END-TRANSACTION
```

## Pathsend Requesters

As an alternative to writing SCREEN COBOL requesters, you can write Pathsend requesters in C, C++, COBOL85, Pascal, or TAL. In such requesters, you use Pathsend procedure calls to communicate with Pathway servers. The LINKMON process manages links to your server processes on behalf of Pathsend requesters.

### Design Considerations

The following considerations should help you decide whether to use Pathsend requesters in your applications:

Pathsend requesters are a good choice for your applications if you need to do the following:

- Take a high volume of transactions from a limited number of devices. In this scenario, there are relatively few requester processes, the requesters are busy, and configuration and management is minimal.
- Access servers that are shared by Pathway requesters and applications other than OLTP applications; for example, a security-checking server or a logging server. If such servers are used infrequently or if the workload varies, server processes can be automatically deleted when not needed and restarted through the PATHMON process when needed again.
- Access servers from environments containing a mix of online transaction processing and batch processing; that is, environments where the same set of servers handle both online requests and requests from batch applications such as NetBatch Plus processes.

- Write nested servers, which act as requesters by making requests to servers in other server classes, perhaps server classes managed by a different PATHMON process.
- Write context-sensitive servers (servers that retain information about the processing of previous requests).

Pathsend procedure calls give you more flexibility than WRITEREAD calls for server-to-server communication. The application gets all the advantages of server classes, including advantages not readily available with WRITEREAD; for example, load balancing, adjusting the number of servers to fit response-time requirements, and configuration and operations management. You can use the Pathsend procedure calls in C, C++, COBOL85, Pascal, and TAL programs.

The Pathsend procedures and the LINKMON process, however, do not provide multithreading, fault tolerance, device configuration, or operations management for requesters. Therefore, if you need these capabilities in a Pathsend requester, you must provide the programming for them.

In addition, Pathsend procedure calls that send messages to server classes must be protected by the TMF subsystem to ensure data integrity in your Pathway application.

The Pathsend procedures and the LINKMON process do not support the checkpointing of Guardian interprocess message synchronization IDs. This lack of checkpointing support is an important consideration when writing fault-tolerant requester programs that do not use the TMF subsystem.

For more information about designing and coding Pathsend requesters, refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

## Clients Using RSC/MP

The RSC/MP product facilitates client/server computing, allowing workstation applications to access Pathway server classes and Guardian processes. This product supports a number of different transport protocols and workstation platforms. For detailed information about the supported platforms and protocols, refer to the *Compaq NonStop™ Remote Server Call (RSC/MP) Programming Manual*.

Transactions are transmitted from the workstation application (the client) to a Pathway application running on a Compaq *NonStop™ Himalaya* system (the server) by means of a supported communications protocol, such as NETBIOS, TCP/IP, or an asynchronous connection.

RSC/MP includes a process called the Transaction Delivery Process (TDP), which resides on the NonStop™ Himalaya system. The TDP is a multithreaded process that can handle multiple workstations. It routes request messages from workstations to Pathway server classes by using either the Pathsend API and the LINKMON process or the terminal control process (TCP) provided in the Pathway/iTS product. If the TCP is used, it can route a request message to a Pathway server by using either the intelligent device support (IDS) requester supplied as part of RSC/MP or an IDS requester that you develop yourself. The TDP can also send request messages from a workstation to a Guardian process.

For information about designing and coding requesters with the RSC/MP product, refer to the *Compaq NonStop™ Remote Server Call (RSC/MP) Programming Manual*.

## Requesters Using GDSX

The Extended General Device Support (GDSX) communications subsystem product simplifies the development of front-end processes and back-end processes for communication with I/O devices. These devices can be of any type, including workstations, terminals, ATMs, point-of-sale (POS) devices, and industrial robots. GDSX supplies code that provides multitasking and other features useful for developing these front-end and back-end processes.

A GDSX process can act as a front-end process for LINKMON processes or a Pathway/iTS terminal control process (TCP).

A GDSX process contains two primary parts:

- TSCODE, supplied by Compaq
- USCODE, supplied by the application programmer

TSCODE provides generic routines and management services that help you build a multithreaded, fault-tolerant process. TSCODE provides the following functions:

- Creates new tasks and stops tasks
- Receives all system messages and I/O requests
- Dispatches (wakes up and executes) the appropriate active task to process messages and requests
- Handles errors

USCODE consists of user exits that are called by TSCODE to handle the application-specific, data communications-related functions, such as data manipulation, protocol conversion, and message routing for the I/O process. USCODE is typically written in the Transaction Application Language (TAL) and bound with TSCODE to produce a functional GDSX process.

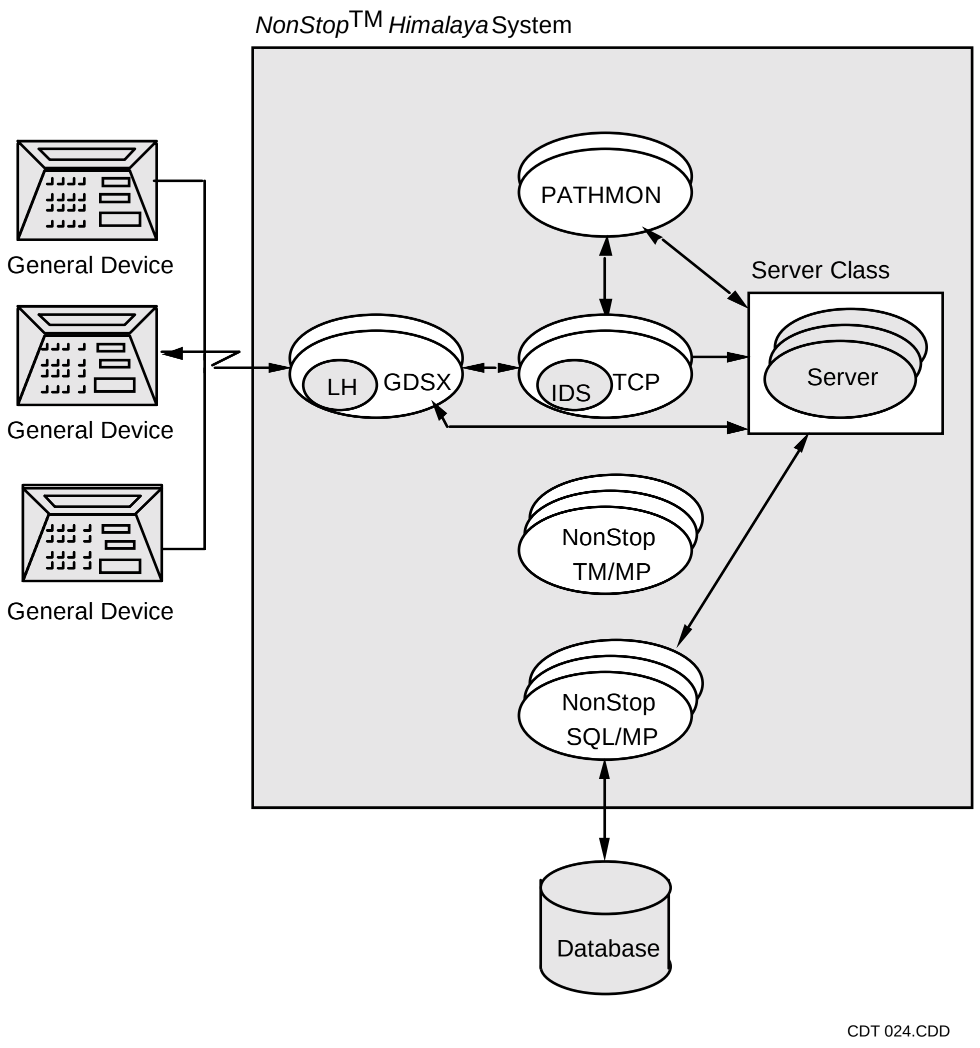
GDSX provides its own interface to Guardian procedures, NonStop™ TM/MP procedures, and Pathsend procedures. The names of the GDSX procedures typically look like their Guardian, NonStop™ TM/MP, or Pathsend equivalents, but they have a circumflex (^) character inserted before the procedure name. For example, BEGINTRANSACTION becomes ^BEGINTRANSACTION.

When a GDSX process is used as a front-end process, multiple threads of a user-coded device handler provide separate tasks to manage the input from I/O devices and provide functions such as data-stream conversion, implementation of a communications protocol, and network communications error handling. One instance of the device handler manages one I/O device.

In the Pathway environment, the GDSX process often simulates a terminal supported by the TCP; the simulated terminal is typically run by an IDS requester program. When the IDS facility is used, the GDSX product can be used to manage the line protocol controlling the connected devices. The GDSX line handler (LH) task can be used to coordinate multiple GDSX device handler tasks or the GDSX device handler task can directly communicate with a back-end line.

Figure 2-4 shows the path of a transaction from a general device to a Pathway server through a GDSX process.

**Figure 2-4. GDSX as a Front-End Process**



When developing a front-end process using GDSX, consider the following:

- A GDSX front-end process is a good choice when a specified data communications protocol is not supported by the Pathway TCP but is supported by GDSX.
- A GDSX front-end process is also a good choice when performance is critical. SCREEN COBOL may not be efficient enough to handle a large amount of application function.
- GDSX processes are managed either through the Subsystem Control Facility (SCF) interactive interface or through a management application program using the Subsystem Programmatic Interface (SPI).

For further information about designing and coding GDSX processes, refer to the *Extended General Device Support (GDSX) Manual*.

### Dividing Function Between Requester and Server

In designing a Pathway application, you must decide how to divide function between requester and server. In making this decision, you should consider the type of requester or client you are writing (SCREEN COBOL, Pathsend, RSC/MP, or GDSX), and you should also consider performance, maintainability, and other factors.

For example, what module should check entry fields for validity? If you are writing a SCREEN COBOL requester, you can easily code it so that the TCP performs these checks. However, a special edit-checking server could provide better performance. If your application includes a workstation requester that communicates with servers using RSC/MP, having the requester check the entry fields would save communications overhead.

As another example, what module should change screen field attributes such as color, blink, brightness, or reverse video for such purposes as highlighting an entry field that contains an error? The SCREEN COBOL language allows such work to be done by the requester, but it could also be done by the server.

For more considerations about dividing function among modules within an application, refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

## Designing Server Programs

Request validations, security checks, calculations, database inquiries, and database changes made in response to a request message are performed by individual units of code within Pathway server programs. As an application programmer, your task is to create a server program to perform specific tasks (for example, create a customer account).

You can write Pathway server programs in C, C++, COBOL85, pTAL, TAL, FORTRAN, or Pascal in the Guardian environment. Alternatively, you can write Pathway server programs in C or COBOL85 in the NonStop™ Kernel Open System Services (OSS) environment; you must program such servers to read the Guardian $RECEIVE file as described in the *Open System Services Programmer's Guide*. In both cases, you configure and manage the servers by using the PATHCOM interactive interface or the Pathway management programming interface (based on the Subsystem Programmatic Interface, or SPI) in the Guardian environment.

Regardless of which operating environment or programming language you use, your Pathway server programs can access database files through the NonStop™ SQL/MP relational database management system or the Enscribe database record manager. See Designing the Database on page 2-8 for information about these two database managers.

You can use the same server programs, whether developed in the Guardian environment or in the OSS environment, with several different requester and client interfaces. These interfaces include SCREEN COBOL, the Pathsend procedures, and the RSC/MP interface. Requesters or clients using different interfaces can share the same Pathway server classes if you ensure that the server program’s request and reply formats are consistent for all requesters.

After you code and compile your server program, the server object code and library code are shared among all processes of the same server class.

For information about designing and coding Pathway servers, refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

## Designing Applications for Batch Processing

If your Pathway application includes batch processing, consider the different needs of this type of processing in your design.

For example, you might code a Pathsend program that takes its input from a file rather than from a terminal, then sends requests to a server to make updates to a database. This program could be configured as a server, thus operating as a nested server. Its input file might be TMF protected, and the Pathsend program might make updates to it.

An application that does several updates to a database, with each update coded as a separate TMF transaction, could be slow when it performs these updates as a batch job rather than performing them online. For batch processing, it is usually faster to group a number of updates in a single transaction. However, if your batch jobs are very large, note that you should not try to group more than about one thousand updates in one TMF transaction.

# 3 Programming for Specific Terminals

Compaq *NonStop*™ Pathway/iTS currently supports IBM 3270 terminals, the 6530 family of terminals (652x series, 653x series, and 654x series), any device that is recognized by the file system as a conversational-mode terminal, and any entity that is identified as an intelligent-mode device. Each type of device has its unique set of requirements. This section summarizes those requirements.

## Using IBM 3270 Terminals

When communicating with IBM 3270 terminals in the Pathway environment, there are several important things to consider:

- The screen size of the terminal models
- The rules for controlling the screen mode
- The rules for positioning the screen fields
- The rules for positioning the cursor
- The use of the terminal’s function keys
- The use of extended field attributes

### Screen Size

The supported IBM 3270 terminals have a number of different physical screen sizes. Table 3-1 lists the various terminal subclasses with their maximum screen sizes (lines by columns), alternate screen sizes, and model names.

**Table 3-1. IBM 3270 Terminal Subclasses and Screen Sizes**

| Subclass | Screen Size | Model |
|---|---|---|
| 1 | 12 x 40 | IBM 3277 M1 |
| 2 | 24 x 80 | IBM 3277 M2 |
| 3 | 24 x 80, alternate 32 x 80 | IBM 3278 M3 |
| 4 | 24 x 80, alternate 43 x 80 | IBM 3278 M4 |
| 5 | 12 x 40, alternate 12 x 80 | IBM 3278 M1 |
| 6 | 24 x 80, alternate 27 x 132 | IBM 3278 M5 |

When running a SCREEN COBOL application on the IBM terminals, consider the following:

- A single screen definition can be displayed successfully on any model as long as its logical screen size is less than or equal to the maximum physical screen size capability of the terminal. If the logical screen size is greater than the maximum physical screen size, the terminal suspends operation with the following error:

  ```
  ERROR - *3990*  REFERENCED SCREEN IS ILLEGAL FOR TERMINAL
  TYPE
  ```

- A single field that wraps from one line to the next in the logical screen definition does not wrap, or wraps differently, if the physical screen width exceeds the logical screen width. The field wraps around to the next line only at the end of the physical line. This wrapping is an important consideration if a screen is intended to run on both 40-column displays and 80-column displays.

## Controlling the Screen Modes

As shown in Table 3-1, some IBM terminals have alternate screen sizes. The logical screen size specified in the SCREEN COBOL screen definition of the current base screen determines which mode the terminal operates in.

For example, terminal model IBM 3278 M5 has these two screen modes:

| | **Base Screen Size** | |
|---|---|---|
| **Mode** | **Lines** | **Columns** |
| 80-column (80-column screen with standard characters) | 1-24 | 1-80 |
| 132-column (132-column screen with compressed characters) | 1-27 | 1-132 |

When a SCREEN COBOL application is run on the IBM 3278 M5 terminal, the screen mode is determined as follows:

- If the screen size definition is equal to or less than the line limit of 24 and the column limit of 80, the 80-column mode is used.
- If the screen size definition is in the line range of 25 through 27, or in the column range of 81 through 132, the 132-column mode is used.
- If no screen size is specified, the 80-column mode is used.

**Note.** Switching screen modes can decrease performance because the terminal memory is cleared to spaces on every mode switch. For the best performance for your SCREEN COBOL application, use the same screen mode for all the screens.

## Positioning the Screen Fields

All fields must reserve a blank character position immediately before the field. For example:

- If a single-character field is to be located at line 2, column 2, on the terminal screen, then both character positions 2,1 and 2,2 must be reserved for that field. In that case, a second field could not begin at character position 2,3 on the terminal screen because the preceding character position (2,2) is already in use by the preceding field and is therefore no longer available as a blank character position.
- A field cannot be at character position 1,1 because the character position preceding 1,1 does not exist and thus cannot be reserved.

The minimum separation, in bytes, between screen elements for the IBM 3270 is indicated in Table 3-2.

**Table 3-2. Minimum Character Separation for IBM 3270 Terminals**

| | **Second Element** | | | |
|---|---|---|---|---|
| **First Element** | **Field** | **Literal** | **Overlay Area** | **End of Screen** |
| Start of base screen | 1 | 1 | 0* | N. A. |
| Start of overlay screen occupying area that does not have the same width as its base screen ** | 1 | 1 | 0 | N. A. |
| Field | 1 (3)*** | 1 | 0 (1)**** (2)*** | 0 (2)*** |
| Literal | 1 | 1 | 0 (1)**** | 0 |
| Overlay Area | 1 | 1 | 0 | 0* |

* Does not support WHEN FULL TAB.

** When an overlay screen occupies an overlay area that does not have the same width as its base screen, an overlay field cannot wrap from one line to the next.

*** Extra separation (two or three bytes) required to support WHEN FULL LOCK.

**** Extra separation required to support WHEN FULL TAB. Use one byte to separate the elements.

## Positioning the Cursor

Cursor positioning on screens with protected fields after an ACCEPT operation acts differently on an IBM 3270 than such cursor positioning does on a 6520. The IBM 3270 does not prevent cursor positioning at a protected field. The 6520, however, automatically repositions a cursor to the next unprotected field if the cursor initially positions at a protected field. If you do not want the cursor to be positioned at a protected field on an IBM 3270 before an ACCEPT operation, you can use the SET command to specify NEW-CURSOR at the desired, unprotected screen field.

## Using IBM 3270 Function Keys

The IBM terminals have a unique set of function keys that can be used by a SCREEN COBOL application. The function keys are:

- PA1 through PA3
- PA4 through PA10
- PF1 through PF24

The PA4 though PA10 keys are optional keys, and the values they transmit when pressed may vary from vendor to vendor. Assigning a unique function-key value to one of those keys' logical PA program name can be done with the user-replaceable procedures in the TCP user library. See Section 4, Writing User Conversion Procedures, for more details.

## Using Extended Field Attributes

Pathway/iTS supports extended field attributes on terminals in the IBM 3270 family. Depending on the terminal, Pathway/iTS supports the following kinds of extended field attributes:

- Color display attributes
- Highlight display attributes (REVERSE, BLINK, and UNDERSCORE)
- Outline display attributes
- Audible alarm feature

Because of differences in terminals in the IBM 3270 family, you should consider the following when you use extended field attributes:

- Some terminals do not support any color display attributes.
- Some terminals that support color display attributes support seven colors, while others support only four.
- Some terminals support highlight display attributes only.
- Some terminals support outline display attributes only.
- Some terminals support both highlight and outline display attributes.
- Some terminals support neither highlight nor outline display attributes.

When specifying the extended field attributes in a SCREEN COBOL program unit, consider:

- In the SPECIAL-NAMES paragraph, define mnemonic names to identify the system names (or combinations of system names) that correspond to one or more of the color, highlight, or outline display attributes.

- Use the mnemonic names in either the Screen Section or the Procedure Division of the program unit to refer to the color, highlight, or outline display attributes. For example:

  ```
  MNEMONIC-NAME-1 IS RED.
  ```

  When the program unit is run, the TCP determines which extended field attributes are supported by the terminal.

- The TCP uses only the extended field attributes that the terminal supports.

  If, for example, your SCREEN COBOL program unit uses color display attributes that are available on an IBM 3279 color terminal, but the program unit is run on an IBM 3278 terminal that does not support color display attributes:

  - The TCP ignores the color display attributes used in the program unit.
  - The screen appears correctly on the terminal, but without color.

  You can use the optional SET MINIMUM-COLOR and SET MINIMUM-ATTR statements in the Procedure Division if you must establish the minimum level of support for color, highlight, and outline display attributes for that program unit and all other program units called by that program unit. The TCP uses the information provided by the SET MINIMUM-COLOR and SET MINIMUM-ATTR statements to determine the level of support for color, highlight, and outline display attributes required by a program unit.

  **Note.** The minimum level of support for color display attributes does not change until a subsequent SET MINIMUM-COLOR statement is executed, even if you use a CALL statement to move between program units. Likewise, the minimum level of support for highlight or outline display attributes does not change until a subsequent SET MINIMUM-ATTR statement is executed, even if you use a CALL statement to move between program units.

  For example, if your program unit requires seven colors, or both reverse and blink highlight display attributes, you can use the following statements to establish those requirements:

  - The SET MINIMUM-COLOR statement establishes the minimum level of support for color display attributes.
  - The SET MINIMUM-ATTR statement establishes the minimum level of support for highlight and outline display attributes.

  For further information about the SET MINIMUM-ATTR and SET MINIMUM-COLOR statements and their default values, see the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual.*

  To establish the attributes required for a program unit, you issue statements such as the following before a DISPLAY BASE statement:

  ```
  MOVE 1 TO IBM-FULL-COLOR    OF WS-MINIMUM-COLOR.
  MOVE 1 TO IBM-FIELD-OUTLINE OF WS-MINIMUM-ATTRIBUTE.
  SET MINIMUM-COLOR USING WS-MINIMUM-COLOR.
  SET MINIMUM-ATTR  USING WS-MINIMUM-ATTRIBUTE.
  ```

The SET MINIMUM-ATTR and SET MINIMUM-COLOR statements cause the information in the working-storage definitions to be extracted and stored in a control block with other information related to the program unit.

- When a terminal or program unit is started, the TCP initializes the extended field attribute requirements of the program unit as follows:
  - No support for color display attributes is required.
  - No support for highlight display attributes is required.
  - No support for outline display attributes is required.
- You need not modify existing program units for terminals in the IBM 3270 family unless you want to use the color, highlight, or outline display attributes.
- When a SET MINIMUM-ATTR or SET MINIMUM-COLOR statement is executed, the TCP updates the requirements of the program unit for the appropriate extended field attributes.
- When a DISPLAY BASE statement is executed, the TCP compares the capabilities of the terminal with the requirements of the program unit:
  - If the program unit requires extended field attributes that the terminal does not support, the TCP aborts the program. Termination status 71 indicates insufficient support for the color, highlight, or outline display attributes required.
  - If the program unit requires extended field attributes that the terminal does support, the program unit runs.
- If the TCP determines that a program unit was compiled with color, highlight, or outline display attributes:
  - The TCP uses the default foreground color, green, if you do not specify a foreground color.
  - The TCP issues a READ PARTITION structured field message to the terminal. The TCP uses the reply to this message to determine the capabilities of the terminal. If the terminal does not support query reply, the TCP ignores the extended field attributes. See the *IBM 3270 Information System Data Stream Programmer's Reference Manual* for information about building and transmitting specific reply sequences.

To determine the level of support for these attributes on a given terminal, you can use the TERMINALINFO statement. This statement determines which extended field attributes a terminal supports; you can issue the statement at any place in the program. You usually issue a TERMINALINFO statement before the first DISPLAY BASE statement in a program unit. You can then determine whether a terminal supports extended field attributes and take action in the program unit accordingly.

For further information about the TERMINALINFO statement, see the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

Color, highlight, and outline display attributes associated with a mnemonic name declared in the SPECIAL-NAMES paragraph can be used in the TURN statement, which changes the display attributes of fields, as in the following example:

```
TURN ALERT           IN SCREEN-FIELD-01 SHADOWED.
TURN TEMP BLUE       IN SCREEN-FIELD-02 SHADOWED.
TURN BG-YELLOW       IN SCREEN-FIELD-03 SHADOWED.
TURN ATTRS-AND-COLOR IN SCREEN-FIELD-04 SHADOWED.
```

## Using Color Display Attributes

Terminals in the IBM 3270 family that support color and allow query reply can support some or all of the following colors:

BLUE
RED
PINK
GREEN
TURQUOISE
YELLOW
NEUTRAL

Terminals in the IBM 3270 family can be grouped in three categories according to the level of support for color display attributes.

- The following terminals in the IBM 3270 family that support color display attributes support four colors (BLUE, GREEN, RED, and NEUTRAL):

  3287-1C
  3287-2C

- The following terminals in the IBM 3270 family that support color display attributes support seven colors (BLUE, GREEN, PINK, RED, TURQUOISE, YELLOW, and NEUTRAL):

  3192
  3194
  3279

- The following terminals in the IBM 3270 family do not support color display attributes:

  3178
  3191
  3278
  3230
  3262
  3268
  3287-1
  3287-2

You cannot use more than one foreground color (that is, the color of characters displayed on the terminal screen). In other words, you can assign only one foreground color display attribute to a single mnemonic-name. For example:

```
LEGAL-MNEMONIC-6 IS RED.
```

Attempting to combine more than one foreground color display attribute in a mnemonic-name parameter results in a syntax error. In the following example, ACTION-FIELD3 and ACTION-FIELD4 are not the same, because attributes are applied in chronological order. ACTION-FIELD5 and ACTION-FIELD6, however, are exactly the same.

```
SPECIAL-NAMES.
  RED IS RED, INPUT-ERROR IS BLINK,
  DRIVE-CRAZY IS (RED, BLINK, BRIGHT),
  ACTION-FIELD1 IS (BLINK, PINK, UNDERLINE, TOPLINE),
  ACTION-FIELD2 IS (BLINK, BLUE, BOXFIELD),
  ACTION-FIELD3 IS (REVERSE, BOXFIELD, NOTOPLINE),
  ACTION-FIELD4 IS (REVERSE, NOTOPLINE, BOXFIELD),
  ACTION-FIELD5 IS (UNDERLINE, GREEN, BOXFIELD),
  ACTION-FIELD6 IS (UNDERLINE, GREEN, TOPLINE, LEFTLINE,
                    RIGHTLINE, BOTTOMLINE).

SPECIAL-NAMES.
  STOP           IS RED,
  GO             IS GREEN,
  CAUTION        IS YELLOW,
  NORMAL         IS NEUTRAL,
  ALERT          IS REVERSE,
  BLINK-VIDEO    IS BLINK,
  ALTER-VIDEO    IS BRIGHT,
  INCORRECT-DATA IS BLINK.
```

## Using Highlight Display Attributes

Pathway/iTS supports the following highlight display attributes on terminals in the IBM 3270 family:

BLINK
NOBLINK
BRIGHT
NORMAL
HIDDEN
NOTHIDDEN
MDTON
MDTOFF
NUMERIC-SHIFT
PROTECTED
UNPROTECTED
REVERSE
NOREVERSE
UNDERLINE
NOUNDERLINE

## Using Outline Display Attributes

Pathway/iTS supports the following outline display attributes on terminals in the IBM 3270 family:

TOPLINE
NOTOPLINE
LEFTLINE
NOLEFTLINE
RIGHTLINE
NORIGHTLINE
BOTTOMLINE
NOBOTTOMLINE
BOXFIELD

**Note.** BOXFIELD is equivalent to the combination of TOPLINE, LEFTLINE, RIGHTLINE, and BOTTOMLINE.

If the terminal does not support TOPLINE, BOTTOMLINE, RIGHTLINE, or LEFTLINE, the TCP marks the device as unable to do outlining.

## Using Other Extended Attributes

Pathway/iTS supports the following additional extended display attribute on terminals in the IBM 3270 family:

BELL

## Combining Extended Field Attributes

Pathway/iTS and the terminals in the IBM 3270 family support certain combinations of color, highlight, and outline display attributes. You must observe the following restrictions when combining color, highlight, and outline display attributes in a mnemonic name declared in the SPECIAL-NAMES paragraph:

- You must not combine extended field attributes that Pathway/iTS supports with those that Pathway/iTS does not support. Pathway/iTS does not support the following extended field attribute on terminals in the IBM 3270 family:

  DIM

- You can combine any of the following highlight display attributes with one another:

  BRIGHT
  HIDDEN
  MDTON
  NUMERIC-SHIFT
  PROTECTED

- You can combine one of the following highlight display attributes with one or more of the other highlight display attributes (that is, with BRIGHT, HIDDEN, MDTON, NUMERIC-SHIFT, or PROTECTED). You cannot combine the following highlight display attributes with each other:

  BLINK
  REVERSE
  UNDERLINE

- If a terminal in the IBM 3270 family supports the following outline display attributes, you can combine them with one another and with any highlight display attribute (that is, with BRIGHT, HIDDEN, MDTON, NUMERIC-SHIFT, PROTECTED, BLINK, REVERSE, or UNDERLINE):

  TOPLINE
  NOTOPLINE
  LEFTLINE
  NOLEFTLINE
  RIGHTLINE
  NORIGHTLINE
  BOTTOMLINE
  NOBOTTOMLINE
  BOXFIELD

  Examples:

  ```
  (BLINK, TOPLINE, BOTTOMLINE)

  (REVERSE, LEFTLINE, RIGHTLINE)

  (UNDERLINE, BOTTOMLINE, TOPLINE)
  ```

## Valid Language and Terminal Combinations

The keyword KANJI-KATAKANA specifies the only language that can be declared for IBM 3270 terminals.

If the language does not match the valid language choices for a terminal class, the compiler marks the statement as an error.

# Using 6520 Terminals

When communicating with 6520 terminals in the Pathway environment, you need to consider:

- The rules for controlling the screen mode
- The rules for positioning the screen fields

## Controlling the Screen Modes

The 6520 terminal has two screen modes, as shown in Table 3-3. You control which screen mode is used by a SCREEN COBOL screen definition.

**Table 3-3. Screen Modes for 6520 Terminals**

| | **Base Screen Size** | |
|---|---|---|
| **Mode** | **Lines** | **Columns** |
| 40-column (80-column screen with double-wide characters) | 1-24 | 1-40 |
| 80-column (80-column screen with standard characters) | 1-24 | 41-80 |

When running a SCREEN COBOL application on the 6520 terminal, the screen mode is determined as follows:

- If the screen definition is equal to or less than the column limit of 40, the 40-column mode is used.
- If the screen definition is in the column limit range of 41 through 80, the 80-column mode is used.
- If no screen size is specified, the 80-column mode is used.

Switching screen modes can decrease performance because the terminal memory is cleared to spaces on every mode switch. For the best performance for your SCREEN COBOL application, use the same screen mode for all the screens.

## Positioning the Screen Fields

All nonliteral fields must reserve a blank character immediately before the field. For example:

- If a field is at line 2, column 2, and is one character long, then 2,1 and 2,2 are reserved for the field. A second field cannot be at 2,3 because both fields would attempt to use location 2,2.
- A field cannot be at 1,1 because the character before 1,1 does not exist and thus cannot be reserved.

Fields cannot wrap from the bottom to the top line of the screen.

The minimum separation between screen elements for the 6520 terminal is indicated in Table 3-4.

**Table 3-4. Minimum Character Separation for 6520 Terminals**

| | Second Element | | | |
|---|---|---|---|---|
| **First Element** | **Field** | **Literal** | **Overlay Area** | **End of Screen** |
| Start of base screen | 1 | 1 | 0 | 0 |
| Start of overlay screen occupying an overlay area that does not have the same width as its base screen * | 1 | 1 | 0 | 0 |
| Field | 1 | 1 | 0 | 0 |
| Literal | 1 | 0 or 1** | 0 | 0 |
| Overlay Area | 1 | 1 | 0 | 0 |

* When an overlay screen occupies an overlay area that does not have the same width as its base screen, an overlay field cannot wrap from one line to the next.

** If two successive literals have the same attributes, no separation is necessary. Otherwise, at least one position must separate them.

## Using 6530 Terminals

The 6530 terminal has all the capabilities of the 6520 terminal plus some additional features. The considerations discussed previously for the 6520 also apply to the 6530. These include the rules for controlling the screen mode and the rules for positioning the screen fields.

The 6530 terminal is upwardly compatible with the 6520. Program units compiled for a 6520 can be run on a 6530; however, features unique to the 6530 do not function on the 6520.

The 6530 terminal enables the use of other devices to put data into screen fields. For more information about this ability, refer to the description of the RECEIVE clause in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual.*

### Return-Key Function

Pathway/iTS can enable the Return key to behave as a function key when a SCREEN COBOL program takes control of a 6530 terminal. For the RETURN-KEY function to become effective, the program's SPECIAL-NAMES paragraph must contain a RETURN-KEY phrase as the system-name parameter. The RETURN-KEY definition is local to a SCREEN COBOL program and must be defined in the program or no RETURN-KEY function exists. To use this function in a program that was previously compiled, you must recompile the program and include the RETURN-KEY phrase. If a program is defined for a 6520 terminal and run on a 6530 terminal, you cannot use the RETURN-KEY function.

## Internal Function-Key Queuing

6530 terminals have the unique capability of internally queuing a function key without a read operation being posted. When no terminal read operation is in progress and a terminal key is pressed, the function key value is stored inside the terminal. The value is read upon the next ACCEPT statement that the SCREEN COBOL program executes.

This terminal feature provides a significant convenience for most Pathway applications. You might, however, write an application that uses ESCAPE clauses in which this terminal feature is inappropriate.

In this situation, a function key is queued during the interval between the cancellation of one read and the arrival of another read so that the function key intended for the first operation is in fact applied to the second operation. The terminal operator cannot tell that the first read has been canceled; when the operator presses the function key intending to execute that original action, the key is automatically queued and executed at the next read.

Your program can avoid this situation by causing the keyboard to lock after the cancellation of a read in the following cases:

- After the ESCAPE ON UNSOLICITED MESSAGE clause
- After the ESCAPE ON TIMEOUT clause

The *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* describes the following two SCREEN COBOL special registers that control locking the keyboard when an ESCAPE operation has been performed:

- PW-QUEUE-FKEY-UMP
- PW-QUEUE-FKEY-TIMEOUT

## Using EM6530PC on a 6540 Personal Computer

The 6540 personal computer (PC) was originally equipped with the terminal emulator EM6530PC. EM6530PC emulates the capabilities of the 6530 terminal. The EM6530PC terminal emulator is upwardly compatible with the 6520 and 6530 terminals. Program units compiled for a 6520 or 6530 terminal run successfully with EM6530PC.

The following differences exist between the 6530 terminal and the EM6530PC emulator:

- The EM6530PC emulator does not support alternate input devices; the 6530 terminal does support them.
- The EM6530PC emulator has more screen modes than the 6530 terminal.

The EM6530PC emulator has four screen modes, as shown in Table 3-5. The logical screen size specified in the SCREEN COBOL screen definition of the current base screen determines which mode the terminal operates in.

**Table 3-5. Screen Modes for 6540 Personal Computers**

| | **Base Screen Size** | |
|---|---|---|
| **Mode** | **Lines** | **Columns** |
| 40-column (80-column screen with double-wide characters) | 1-24 | 1-40 |
| 66-column (132-column screen with double-wide characters) | 1-27 | 1-66 |
| 80-column (80-column screen with standard characters) | 1-24 | 67-80 |
| 132-column (132-column screen with standard characters) | 1-27 | 67-132 |

Given the screen size definition, the TCP searches a screen mode definition table to find the first appropriate base size. The table search occurs in the order that the modes are listed. Therefore, you must ensure that the screen size specified results in the desired screen mode being selected. For example, if you want the 66-column mode, you must specify a width between 41 and 66 to prevent the 40-column mode from being used.

If no screen size is specified in the base screen definition, the 80-column mode (24,80) is used.

Switching screen modes can decrease performance because the terminal memory is cleared to spaces on every mode switch. For the best performance for your SCREEN COBOL application, use the same screen mode for all screens.

## Using Conversational Terminals

A conversational terminal is any terminal that the file system recognizes as operating in conversational mode. A conversational terminal processes carriage return, line feed, and bell operations. If the data entered during ACCEPT processing exceeds the size of the I/O buffer, the terminal simply redisplays a field prompt without an advisory error message.

Some of the SCREEN COBOL statements and clauses act differently in block mode than in conversational mode. This discussion summarizes information about using conversational mode.

### Conversational-Mode Program

A SCREEN COBOL program written for conversational-mode operation can run on either a block-mode terminal or a conversational-mode terminal. When a program is specified as conversational, that program performs according to the restrictions for a conversational terminal regardless of the type of terminal on which the program runs.

A SCREEN COBOL program running in conversational mode performs as follows:

- Displays information on the terminal during an ACCEPT statement, one line at a time
- Accepts data entered from the terminal one line at a time

- Responds to a set of input-control characters when the terminal is enabled to accept data
- Recognizes only keyboard characters, carriage returns, and line feeds (not function keys)
- Restricts the display field attributes to BELL and HIDDEN

## Designating Conversational Terminals

You designate a conversational terminal by specifying the following clause in the OBJECT-COMPUTER paragraph of the Environment Division:

```
TERMINAL IS CONVERSATIONAL
```

This clause causes a terminal to operate in conversational mode regardless of the terminal type. Program units compiled for conversational mode can be run on 652x series, 653x series, 654x series, and IBM 3270 terminals or on any other device that the file system recognizes as operating as a conversational-mode terminal.

The available screen field attributes for conversational terminals are: BELL, HIDDEN, NOBELL, and NOTHIDDEN.

Error enhancement is available only for the first field found to be in error. If additional fields are specified, they are ignored. BELL is the applicable error enhancement for conversational terminals. You must explicitly specify ERROR-ENHANCEMENT IS BELL to have error enhancement.

## Input Control Characters

The Screen Section has input-control entries available for terminals in conversational mode. These clauses define the specific input-control characters to be used during execution of an ACCEPT statement. The clauses are as follows:

- ABORT-INPUT defines the characters used to terminate the processing of the current ACCEPT statement with an abort termination status.
- END-OF-INPUT defines the characters used to indicate the end of the last input field for the current ACCEPT statement. If used, the END-OF-INPUT clause must be specified at the 01 screen level. A character defined for END-OF-INPUT cannot be specified for another input-control character.
- FIELD-SEPARATOR defines the character used to separate one screen field from another during an ACCEPT statement. If a screen field description includes an OCCURS clause, each occurrence is treated as one field.
- GROUP-SEPARATOR defines the character used during the processing of an ACCEPT statement to mark the end of the last item in an OCCURS clause or the last field of a group declaration that does not contain an OCCURS.
- RESTART-INPUT defines the characters used to restart input processing during the current ACCEPT statement.

## Displaying Information

The DISPLAY BASE statement establishes the current screen.

A DISPLAY statement in conversational mode causes the TCP to write to the terminal display, which can be a screen, printer paper, and so forth.

The DISPLAY statement presents output in order by rows. A screen field value appears on the screen at the column number position specified in the screen field description. Blank lines for formatting purposes are not generated. Therefore, screen lines generally do not correspond with the line numbers specified in the Screen Section.

To display fully line-formatted screens, define at least one item for every line (row) of the screen. If a row of spacing is required, define the screen item for that row with a VALUE clause specifying blanks, for example, VALUE " ". Then, display the entire screen by specifying the screen name as the screen identifier in the DISPLAY statement.

For more information, see the descriptions of the DISPLAY BASE and DISPLAY statements in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual.*

## Accepting Information

If the terminal associated with the SCREEN COBOL program is operating in conversational mode, the ACCEPT statement performs the following:

- Displays the prompt value defined for the first screen field described with a PROMPT clause. The prompt value is always displayed in the first column of the screen line.
- Waits for a response from the terminal. If the TIMEOUT phrase is used, ACCEPT waits the length of time specified in this phrase.
- Receives input from the terminal and stores the data into the associated working-storage items of the program data area. Input can be accepted from the terminal one screen field at a time, one field per line. However, the capability referred to as type ahead enables data entry for more than one field on the same line.
- Returns only valid data to the program (checking the definitions in the Screen Section of the Data Division to determine the validity of the data). All SHADOWED fields associated with the input fields of the ACCEPT statement have their ENTERED and RETURNED bits set appropriately.

If invalid data is entered and an ADVISORY field is defined, an error message is displayed, the prompt is redisplayed for the field in error, and the data can be reentered. If an ADVISORY field is not defined for the base screen, only the prompt is redisplayed for the field in error and the data can be reentered.

For more information, see the description of the ACCEPT statement in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

## Using Intelligent-Mode Devices

A SCREEN COBOL program operates in intelligent mode when it communicates with an intelligent device. An intelligent device is any device that can receive and reply to messages sent by the SEND MESSAGE statement in a SCREEN COBOL program. An intelligent device could be a personal computer, an automatic teller machine, a point-of-sale device, a *Guardian* operating environment process, a communications line, or a 6540 terminal operating as a personal computer.

When writing a program to communicate with such a device, you should be aware of the following:

- The SCREEN COBOL program has no control over the device. It is up to the person writing code for the device to start up the device; accept any messages from the SCREEN COBOL program; send any replies back to the SCREEN COBOL program; supply the operator interface, if any; and so forth.
- The SCREEN COBOL program is responsible for synchronizing messages between the program and the intelligent device. The program must check for duplicate messages from the device.
- Pathway/iTS does not establish modem connections for intelligent devices. If you want a modem connection, you must use the RECONNECT MODEM statement.
- The TCP does not collect messages for intelligent devices in a buffer as it does for data sent to a terminal screen. Each message is sent when the SEND MESSAGE statement is executed.
- The Message Section provides some formatting of data sent to and received from an intelligent device. When data is passed directly between the intelligent device and data areas in the Working-Storage or Linkage Sections, it is not formatted.
- Programs that communicate with intelligent devices cannot use the following SCREEN COBOL statements:

| | |
|---|---|
| ACCEPT | RESET |
| CLEAR INPUT | SCROLL |
| DISPLAY | SET NEW-CURSOR AT |
| DISPLAY BASE | TURN |
| DISPLAY OVERLAY | USE FOR SCREEN RECOVERY |
| DISPLAY RECOVERY | USE FOR TERMINAL-ERRORS |
| PRINT SCREEN | |

- The following special registers have no meaning for programs that communicate with intelligent devices:

  REDISPLAY

  PW-QUEUE-FKEY-UMP

  PW-QUEUE-FKEY-TIMEOUT

  PW-TERMINAL-ERROR-OCCURRED

  TELL-ALLOWED

- In order for a SCREEN COBOL program unit to communicate with an intelligent device, the OBJECT-COMPUTER paragraph of the Environment Division must specify that the terminal type is INTELLIGENT. In addition, the PATHMON environment must be configured for intelligent devices; this is accomplished by setting the terminal type in a SET TERM or SET PROGRAM command. Refer to the *Compaq NonStop™ Pathway/iTS System Management Manual* for details.

## Using Simulated Devices

Compaq provides the means for you to write nonprivileged programs that function as a terminal, printer, tape driver, or other device. In the terminology of the Compaq *NonStop*™ Kernel operating system, these programs are called subtype 30 processes.

Specifying device subtype 30 tells the system that the terminal-simulation process will supply device information in response to a request for the device-type information. Hence device subtype 30 must be specified for a terminal-simulation process; otherwise, the file system will reply to the DEVICEINFO request.

All the devices that Pathway/iTS interfaces with can be emulated with a subtype 30 process. The device list includes:

- Terminals

  RUN PROGRAM terminals have a DEVICEINFO time limit of five minutes. START TERM terminals do not have a DEVICEINFO time limit.

- Log files

  Log files have a DEVICEINFO time limit of two minutes if PATHMON is starting. However, log files have a time limit of five minutes if PATHMON is running.

- Associative servers

  Associative servers have a DEVICEINFO time limit of five minutes.

When a DEVICEINFO error occurs, an error is not logged. Instead, the following error message is sent to the operator:

```
ERROR - *1040*  UNABLE TO DETERMINE DEVICE TYPE (nnn)
```

where *nnn* is an operating system error.

## Using Dial-in Terminals

When dial-in terminals are in use, the terminal control process (TCP) issues a CONTROL 11 operation (wait for modem connect) immediately after the terminal file is opened. At terminal startup time, no program unit or data area is attached to the terminal; therefore, the terminal is using a minimum of TCP resources while waiting for modem connect. When the terminal is stopped, the terminal file is closed. The close causes the modem to disconnect if no other process has the terminal file open.

# 4 Writing User Conversion Procedures

If you include a USER CONVERSION clause in a screen description entry, a message description entry, or a SEND MESSAGE statement, you must provide a corresponding user conversion procedure. The user conversion procedure lets you make your own validation checks or conversions of data passed between a SCREEN COBOL program and a terminal screen or intelligent device.

This section presents information about the following topics:

- User conversion procedures
- User-written user conversion procedures
- Screen input procedures
- Screen output procedures
- 3270 key mapping
- Intelligent device input procedures
- Intelligent device output procedures

## User Conversion Procedures

Compaq provides nine user conversion procedures with the Compaq *NonStop*™ Pathway/iTS software: four to convert terminal screen data, four to convert intelligent device message data, and one to support IBM 3270 attention keys. The data conversion procedures consist of input and output procedures. The input procedures provide conversion and data validation; the output procedures provide output conversion. The key-mapping procedure supports the program attention keys (PA4 through PA10) on an IBM 3270 (or analogous) terminal. In summary, the user conversion procedures are as follows:

| | **Conversion Procedures** | **Data** |
|---|---|---|
| Terminals | Screen input | Numeric data<br>Alphanumeric data |
| | Screen output | Numeric data<br>Alphanumeric data |
| | 3270 key mapping | Attention keys |
| Intelligent Devices | Device input | Numeric data<br>Alphanumeric data |
| | Device output | Numeric data<br>Alphanumeric data |

The five terminal procedures and four intelligent-device procedures exist in the TCP object library. The TCP calls the specified procedure whenever you include a USER

CONVERSION clause as part of either a field definition or a SEND MESSAGE statement.

## User-Written User Conversion Procedures

You can write your own user conversion procedures in the Portable Transaction Application Language (pTAL) and use the `nld` utility to link your procedures in the native TCP user library object file, PATHTCPL.

In releases prior to D40, user conversion procedures were written in TAL. In D40 and later releases (including all G-series releases), user conversion procedures must be written in pTAL. pTAL is based on TAL. The pTAL language excludes architecture-specific TAL constructs and includes new constructs that replace the architecture-specific constructs. You can write user conversion procedures that can be compiled by both the TAL and the pTAL compilers, thus enabling you to use the same source code for different releases of Pathway/iTS.

If you are converting existing user-written user conversion routines to pTAL for use with a D40 or later version of Pathway/iTS, refer to the *pTAL Conversion Guide* and the *pTAL Reference Manual* for further information. Many user conversion routines are simple enough that no changes will be needed. However, there is an interface change in the four user conversion procedures for intelligent devices, as shown in Figure 4-7 and Figure 4-8 later in this section.

User conversion routines and alternative advisory message routines (described in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*) are the only routines that must be compiled with the pTAL compiler.

### Coding the User Conversion Procedures and Creating the User Library

There are four files provided in the installation subvolume:

| | |
|---|---|
| SLIB | An auxiliary source file that is passed to the pTAL compiler for compiling the user conversion procedures |
| ILIB | An interface source file containing all the definitions and data structures used by the user conversion procedures |
| TLIB | A source file containing the stubs of the user conversion procedures |
| TCPLIB | An object file that is required to build the native user library |

You should code the user conversion procedures only in the TLIB file.

Use the following four steps to code your user conversion procedures:

1. Copy the files SLIB, ILIB, and TLIB from the installation subvolume to your subvolume: for example, $MY.USERCNV. Make changes to the user conversion procedures that you want to use.

2. Compile the source using pTAL as follows:

```
PTAL/IN $MY.USERCNV.SLIB/user-conversion-object
```

*user-conversion-object*

is a pTAL object file.

3. Remove all compilation errors.
4. Build the user library using the `nld` utility as follows:

```
NLD user-conversion-object $volume.ZPATHWAY.TCPLIB
-UL-O native-user-library
```

*user-conversion-object*

is a pTAL object file.

$*volume*

is the volume where the installation subvolume ZPATHWAY resides.

*native-user-library*

is the native user library object file used by the TCP.

## Using the MAKEUL Macro

A Compaq Tandem Advanced Command Language (TACL) macro called MAKEUL is provided to facilitate the process of creating the user library. If you are using MAKEUL, perform the following two steps instead of the four steps described in the previous subsection:

1. Copy the files SLIB, ILIB, and TLIB from the installation subvolume to your subvolume, for example $MY.USERCNV. Make changes to the user conversion procedures that you want to use.
2. Compile and build the native user library as follows:

```
MAKEUL -SRC $MY.USERCNV.SLIB -LIB native-user-library
-LOC $volume.ZPATHWAY
```

*native-user-library*

is the native user library source file used by the TCP.

$*volume*

is the volume where the installation subvolume ZPATHWAY resides.

See Appendix A, The MAKEUL Macro, for further information on the MAKEUL macro and related examples.

## Restrictions on User Conversion Procedures

Do not code your user conversion procedures to perform any I/O operations. Such I/O operations could interfere with the operation of the TCP.

# Screen Input Procedures

Two procedures provide conversion during screen input, one procedure for input of numeric data items and the other for input of alphanumeric items. When the USER CONVERSION clause is declared for the field, the appropriate procedure is called:

- Before value checks are applied
- After the input has been stripped of fill characters
- After standard conversion is attempted

The procedure is called even if an error occurs during the standard conversion attempt; if a length error occurs, however, the procedure is not called.

**Note.** If an alphanumeric field is declared with the UPSHIFT and USER CONVERSION clauses, the TCP upshifts the field both before and after the user conversion procedure is called.

Most of the parameters of the two procedures are the same; they differ only for the internal data item. Declarations for the numeric and alphanumeric screen input conversion procedures are shown in Figure 4-1 and Figure 4-2.

**Figure 4-1. Screen Numeric Input Procedure Declaration**

```
PROC USER^NUMERIC^INPUT^CONVERSION ( USERCODE, ERROR,
            INPUT, INPUT^LEN, INTERNAL, INTERNAL^SCALE );

INT       USERCODE;
INT       .ERROR;
STRING    .INPUT;
INT       INPUT^LEN;
FIXED     .INTERNAL;
INT       INTERNAL^SCALE;
```

**Figure 4-2. Screen Alphanumeric Input Procedure Declaration**

```
PROC USER^ALPHA^INPUT^CONVERSION ( USERCODE, ERROR,
            INPUT, INPUT^LEN, INTERNAL, INTERNAL^LEN );

INT       USERCODE;
INT       .ERROR;
STRING    .INPUT;
INT       INPUT^LEN;
STRING    .INTERNAL;
INT       INTERNAL^LEN;
```

`USERCODE`

is the value given in the USER CONVERSION field characteristic clause. This parameter can be used to select a particular type of conversion.

`ERROR`

is both an input and an output parameter. When the procedure is called, the parameter contains either 0 (indicating no error) or the number of a conversion error detected during the attempted standard conversion. Refer to the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* for a listing of the possible error codes.

The value of the ERROR parameter after the call determines whether an error for the field is reported back to the terminal. If the value is nonzero, that value is used to select the error message to be displayed. Processing depends on the purpose of the procedure as follows:

- If the user conversion procedure simply performs additional checking on the input it has received, then the procedure should return immediately if the ERROR is nonzero: that is, skip the additional checking. This is because having a nonzero ERROR indicates that an error has already been encountered and hence it is not necessary to do any further checks. However if the ERROR is zero, then the procedure should proceed with its own checking and set the ERROR accordingly.
- If the user conversion procedure is performing some conversion, then the ERROR may have no meaning because you may have entered some values which will cause the ERROR to be set, but programmatically you would like to convert the entered value to some value which would be understood by your application. Hence in these cases, you should ignore the value present in the ERROR, perform the conversion, and set the ERROR parameter accordingly.

`INPUT`

contains the string of characters input from the terminal. Alphanumeric input is stripped of fill characters from the right; numeric input is stripped of fill characters from both the right and the left.

`INPUT^LEN`

gives the number of bytes in the input string after the string is stripped of fill characters. The byte before and the byte after the input string are set to null values.

`INTERNAL`

contains the result of the standard conversion (if no error occurred) and should contain the result of the user conversion (unless ERROR is nonzero upon return).

- For the numeric procedure, INTERNAL is a FIXED parameter; if necessary, this value is later converted to the final data type by the TCP. The INTERNAL^SCALE parameter gives the scale that INTERNAL should have.

- For the alphanumeric procedure, INTERNAL is a STRING parameter. INTERNAL^LEN represents the maximum number of bytes that the result of the conversion can occupy; therefore, the conversion routine should use INTERNAL^LEN to control the maximum amount of data moved to INTERNAL.

## Screen Output Procedures

Two procedures provide conversion during screen output. One procedure is for output of numeric data items, and the other is for alphanumeric items. When the USER CONVERSION clause is declared for the field, the appropriate procedure is called after standard conversion has completed.

**Note.** If an alphanumeric field is declared with the UPSHIFT and USER CONVERSION clauses, the TCP upshifts the field both before and after the user conversion procedure is called.

Most of the parameters of the two procedures are the same; they differ only for the internal data item. Declarations for the numeric and alphanumeric screen output conversion procedures are shown in Figure 4-3 and Figure 4-4.

**Figure 4-3. Screen Numeric Output Procedure Declaration**

```
PROC USER^NUMERIC^OUTPUT^CONVERSION ( USERCODE, OUTPUT,
            OUTPUT^LEN, MAX^OUTPUT^LEN, INTERNAL,
            INTERNAL^SCALE );

INT       USERCODE;
STRING    .OUTPUT;
INT       .OUTPUT^LEN;
INT       MAX^OUTPUT^LEN;
FIXED     .INTERNAL;
INT       INTERNAL^SCALE;
```

**Figure 4-4. Screen Alphanumeric Output Procedure Declaration**

```
PROC USER^ALPHA^OUTPUT^CONVERSION ( USERCODE, OUTPUT,
     OUTPUT^LEN, MAX^OUTPUT^LEN, INTERNAL, INTERNAL^LEN );

INT        USERCODE;
STRING     .OUTPUT;
INT        .OUTPUT^LEN;
INT        MAX^OUTPUT^LEN;
STRING     .INTERNAL;
INT        INTERNAL^LEN;
```

`USERCODE`

contains the value given in the USER CONVERSION field characteristic clause. This parameter can be used to select a particular type of conversion.

`OUTPUT`

indicates where the string of characters for output to the terminal is to be placed. When the procedure is called, the location designated by this parameter contains the result of the standard conversion.

`OUTPUT^LEN`

contains the length of the output string. If the procedure changes the output string, the procedure should set OUTPUT^LEN to the associated length; in no case should OUTPUT^LEN be greater than MAX^OUTPUT^LEN. If OUTPUT^LEN is less than the field length, the fill character is used to pad the field.

`MAX^OUTPUT^LEN`

represents the maximum possible length of the particular converted output field. This value should be used to control the maximum amount of data moved to OUTPUT.

`INTERNAL`

contains the data to be converted.

- For the numeric procedure, INTERNAL is a FIXED parameter. The INTERNAL^SCALE parameter contains the number of decimal places.
- For the alphanumeric procedure, INTERNAL is a STRING parameter. The INTERNAL^LEN parameter contains the number of bytes in the string.

---

**Note.** Do not expect the string represented by the INTERNAL parameter to be bounded by nulls.

---

## 3270 Key Mapping

The user-replaceable procedure USER^3270^KEY^MAPPING is provided to support program attention keys PA4 through PA10. These keys are used on terminals analogous to the IBM 3270 terminal.

Keys PA4 through PA10 transmit a code called an attention-ID (AID) byte. The actual codes vary among terminals of different vendors. To use these keys, you need to write your own procedure that associates the AID byte transmitted by your terminal with the appropriate Pathway/iTS key number.

Declarations for the key-mapping procedure are shown in Figure 4-5.

**Figure 4-5. 3270 Key-Mapping Procedure Declaration**

```
PROC USER^3270^KEY^MAPPING ( AID, KEYNUM );

INT AID;           !  3270 AID BYTE -- AID byte from terminal
INT .KEYNUM;       !  ON THE CALL   -- Associated with 3270 AID
                   !                   byte (Table 4-1) or -1 if
                   !                   key is undefined
                   !  ON THE RETURN -- Pathway/iTS key number (Figure 4-1)
                   !                   or -1 if undefined
```

`AID`

contains the code transmitted by the attention key (PA4 through PA10). A value for AID is passed to the procedure.

`KEYNUM`

contains the Pathway/iTS key number for the SCREEN COBOL program. A value for KEYNUM is passed on the call and return. KEYNUM is set to -1 if the Pathway/iTS key number is undefined.

The procedure could include logic to test for the AID byte values represented by keys PA4 through PA10 and return the appropriate KEYNUM. The logic could test for KEYNUM value of -1; this value means that the key number is undefined. If KEYNUM is -1, you should then test the AID byte value:

- If the AID byte value indicates a key PA4 through PA10, change KEYNUM to the appropriate Pathway/iTS key number (31 through 37).
- If the AID byte value indicates something other than key PA4 through PA10, merely return, leaving KEYNUM as -1.

Table 4-1 shows the defined relationships between AID byte values and Pathway/iTS key numbers.

**Table 4-1. Mapping of Internal 3270 Key Number to Pathway/iTS Key Number** (page 1 of 3)

| SCREEN COBOL Special-Name | 3270 AID Byte | Pathway/iTS Key Number |
|---|---|---|
| ENTER | %047 | 0 |
| PA1 | %045 | 1 |
| PA2 | %076 | 2 |
| PA3 | %054 | 3 |
| CLEAR | %137 | 4 |
| PF1 | %061 | 5 |
| PF2 | %062 | 6 |

* The AID byte value for this key varies from terminal vendor to terminal vendor. Refer to the manual that came with your terminal for the value.

**Table 4-1. Mapping of Internal 3270 Key Number to Pathway/iTS Key Number** (page 2 of 3)

| | | |
|---|---|---|
| PF3 | %063 | 7 |
| PF4 | %064 | 8 |
| PF5 | %065 | 9 |
| PF6 | %066 | 10 |
| PF7 | %067 | 11 |
| PF8 | %070 | 12 |
| PF9 | %071 | 13 |
| PF10 | %072 | 14 |
| PF11 | %043 | 15 |
| PF12 | %100 | 16 |
| PF13 | %101 | 17 |
| PF14 | %102 | 18 |
| PF15 | %103 | 19 |
| PF16 | %104 | 20 |
| PF17 | %105 | 21 |
| PF18 | %106 | 22 |
| PF19 | %107 | 23 |
| PF20 | %110 | 24 |
| PF21 | %111 | 25 |
| PF22 | %133 | 26 |
| PF23 | %056 | 27 |
| PF24 | %074 | 28 |
| Undefined | %060 | 29 (Test Request) |
| Undefined | %127 | 30 (Op ID Card Reader) |
| PA4* | Undefined | 31 |
| PA5* | Undefined | 32 |
| PA6* | Undefined | 33 |
| PA7* | Undefined | 34 |
| PA8* | Undefined | 35 |
| PA9* | Undefined | 36 |
| PA10* | Undefined | 37 |

* The AID byte value for this key varies from terminal vendor to terminal vendor. Refer to the manual that came with your terminal for the value.

**Table 4-1. Mapping of Internal 3270 Key Number to Pathway/iTS Key Number** (page 3 of 3)

| | | |
|---|---|---|
| Undefined | %075 | -1 (Selector Pen Attn) |
| Undefined | %055 | -1 (No AID--Display) |
| Undefined | %131 | -1 (No AID--Printer) |
| Other | -- | -1 |

* The AID byte value for this key varies from terminal vendor to terminal vendor. Refer to the manual that came with your terminal for the value.

## Intelligent Device Input Procedures

Two procedures can be called to convert input received from an intelligent device: one procedure to convert numeric input and the other to convert alphanumeric input.

You can use the input procedures to:

- Receive and format an input message
- Specify the actual scale of a numeric item or the actual length of a nonnumeric item
- Convert any fill characters in the message
- Right justify alphanumeric data in the message
- Report whether data was actually sent and, if sent, whether the data is nonblank

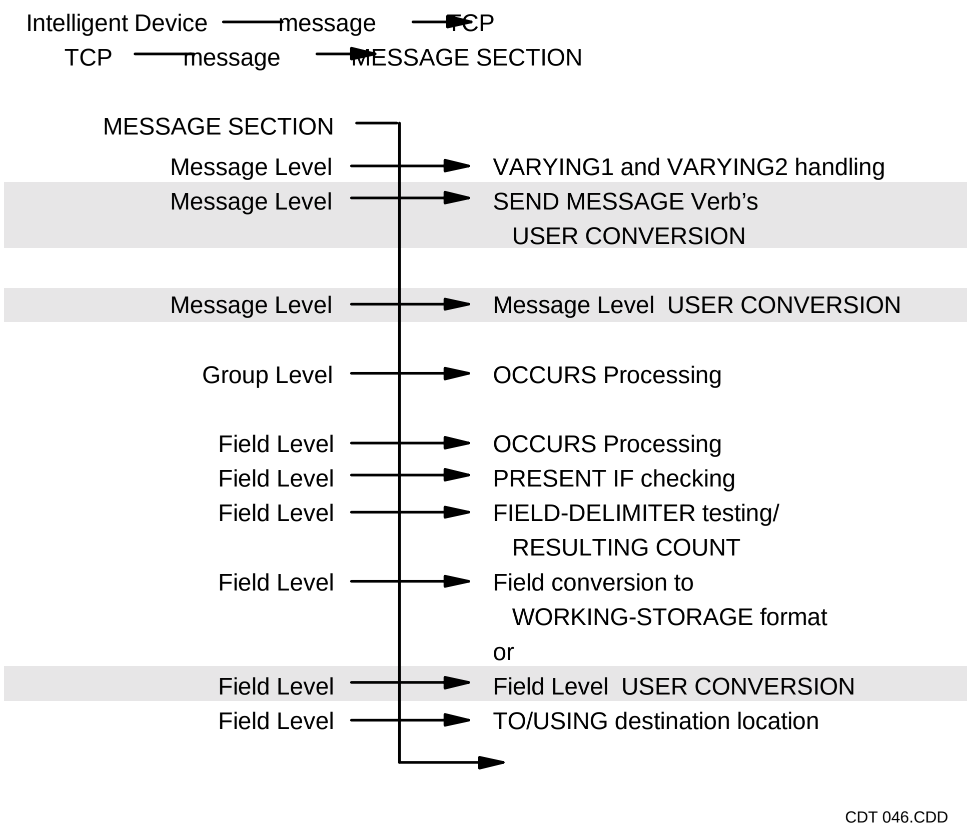
When USER CONVERSION is specified for a message field, the TCP calls the numeric procedure if the data is to be moved to a numeric working-storage field; it calls the alphanumeric procedure if the data is to be moved to an alphanumeric field.

When USER CONVERSION is specified in a SEND MESSAGE statement, the TCP calls the alphanumeric procedure regardless of the message data type. The input procedures are performed before the standard conversion or formatting of the send-message data.

If a USER CONVERSION clause is specified for both a Message Section message field and the SEND MESSAGE statement that sends the message, the TCP first calls the procedure for the SEND MESSAGE statement, then calls the procedure for the message, and finally calls the procedure for the message field.

Figure 4-6 illustrates the data flow of the processing of Message Section items on input. The sequence of events performed for each elementary data field input from an intelligent device is shown; processing steps relevant to user conversion routines are highlighted.

**Figure 4-6. Message Input From an Intelligent Device**



Most of the parameters for the two input procedures are the same; they differ in their internal data representation and in that alphanumeric items can be justified but numeric items cannot. Declarations for the numeric and alphanumeric intelligent device input conversion procedures are shown in Figure 4-7 and Figure 4-8.

**Figure 4-7. Device Numeric Input Procedure Declaration**

```
PROC USER^NUMERIC^INPUT^MSG^CONV ( USERCODE, ERROR, INPUT,
   INPUT^LEN, INTERNAL, INTERNAL^SCALE, FILL^CHAR,
   FILL^OFF, FIELD^RETURNED, FIELD^PRESENT );

INT     USERCODE;        !  Supplied by TCP
INT     .ERROR;          !  Generated by user procedure
STRING  .EXT INPUT;      !  Supplied by TCP
INT     .INPUT^LEN;      !  Supplied by TCP; modifiable by user
FIXED   .INTERNAL;       !  Generated by user procedure
INT     INTERNAL^SCALE;  !  Supplied by TCP
STRING  .FILL^CHAR;      !  Supplied by TCP
INT     FILL^OFF;        !  Supplied by TCP
INT     .FIELD^RETURNED; !  Generated by user procedure
INT     .FIELD^PRESENT;  !  Generated by user procedure
```

**Figure 4-8. Device Alphanumeric Input Procedure Declaration**

```
PROC USER^ALPHA^INPUT^MSG^CONV ( USERCODE, ERROR, INPUT,
   INPUT^LEN, INTERNAL, INTERNAL^LEN, FILL^CHAR,
   FILL^OFF, RIGHT^JUSTIFIED, FIELD^RETURNED,
   FIELD^PRESENT );

INT     USERCODE;          ! Supplied by TCP
INT     .ERROR;            ! Generated by user procedure
STRING  .EXT INPUT;        ! Supplied by TCP
INT     .INPUT^LEN;        ! Supplied by TCP; modifiable by user
STRING   .EXT INTERNAL;    ! Generated by user procedure
INT     INTERNAL^LEN;      ! Supplied by TCP
STRING  .FILL^CHAR;        ! Supplied by TCP
INT     FILL^OFF;          ! Supplied by TCP
INT     .FIELD^RETURNED;   ! Generated by user procedure
INT     .FIELD^PRESENT;    ! Generated by user procedure
```

`USERCODE`

contains the numeric literal specified in the USER CONVERSION clause. The procedure uses the supplied value of this parameter to determine which subset of code to execute.

`ERROR`

contains 0 when the procedure is called. If an error indication is to be returned, the procedure should set ERROR to a nonzero value. If ERROR is nonzero after the call, the TCP reports an error.

`INPUT`

contains the string of characters input from the intelligent device. This string is the raw data for the conversion.

`INPUT^LEN`

contains the length, in bytes, of the data item.

- At the message level, the length is the number of characters received from the intelligent device, that is, the I/O transfer count.
- At the field level, length is relevant only if the item is alphabetic or alphanumeric. Length is the effective length of the field; that is, either the declared length of the field for fixed-format messages or the actual length of the field for delimited (FIELD DELIMITERS ON) messages. Additionally, the TCP adjusts the length (INPUT^LEN) by subtracting the number of trailing blanks found in a field. This adjustment is performed for both fixed-format and delimited-format fields.

`INTERNAL`

is where the TCP expects to find the results of the conversion routine; that is, it is the destination of the data from the INPUT parameter—of length INPUT^LEN—that the conversion routine has processed.

`INTERNAL^SCALE`

is a parameter in the numeric procedure only. The procedure should set it to the number of decimal places of the value stored in INTERNAL when it is a FIXED field.

`INTERNAL^LEN`

is the size of the destination buffer INTERNAL. It is set to the length of the maximum expected reply message for the current SEND MESSAGE operation. The conversion routine should use INTERNAL^LEN to control the maximum amount of data moved into INTERNAL, the destination of the conversion operation.

`FILL^CHAR`

is always set to either 0 (for numeric conversions) or blank (for alphanumeric conversions) by the TCP.

`FILL^OFF`

is always set to -1 by the TCP.

`RIGHT^JUSTIFIED`

contains -1 (TRUE) if an alphanumeric value is to be right justified and contains 0 (FALSE) if not.

`FIELD^RETURNED`

should be set to -1 (TRUE) if any data was sent from the device or be set to 0 (FALSE) if there was no data.

`FIELD^PRESENT`

should be set to -1 (TRUE) if the data sent from the device is nonblank or be set to 0 (FALSE) if the data is blanks.

## Intelligent Device Output Procedures

Two procedures can be called to convert output being sent to an intelligent device: one procedure to convert numeric output and the other to convert alphanumeric output.

You can use the output procedures to:

- Add fill characters to the message, increasing the length up to the maximum message length allowed
- Right justify alphanumeric data in the message

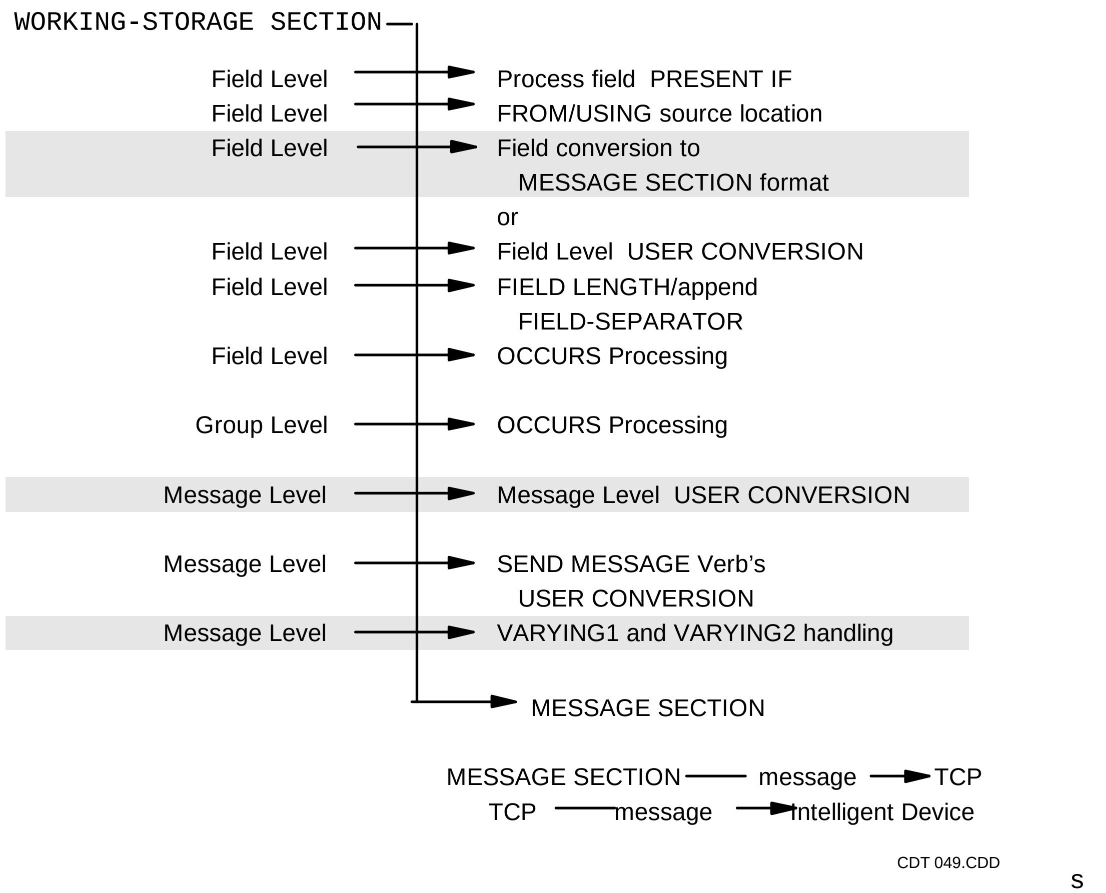
When USER CONVERSION is specified in a message entry, the TCP calls the numeric procedure for data moved from a numeric data field in working storage; it calls the alphanumeric procedure for data moved from an alphanumeric data field in working storage.

When USER CONVERSION is specified in a SEND MESSAGE statement, the TCP calls the alphanumeric procedure regardless of the message data type. The user procedures are performed after any standard conversion of the data.

If a USER CONVERSION clause is specified for both a Message Section message field and the SEND MESSAGE statement that sends the message, the TCP first calls the procedure for the message field, then calls the procedure for the message, and finally calls the procedure for the SEND MESSAGE statement. This calling order is the inverse of the order for intelligent device input procedures.

Figure 4-9 illustrates the data flow of the processing of Message Section items on output. The sequence of events performed for data output to an intelligent device is shown; processing steps relevant to user conversion routines are highlighted.

**Figure 4-9. Message Output to an Intelligent Device**



Most of the parameters for the two output procedures are the same; they differ in their internal data representation and in that alphanumeric items can be justified and numeric items cannot. Declarations for the numeric and alphanumeric device output procedures are shown in Figure 4-10 and Figure 4-11.

**Figure 4-10. Device Numeric Output Procedure Declaration**

```
PROC USER^NUMERIC^OUTPUT^MSG^CONV ( USERCODE, OUTPUT,
   OUTPUT^LEN, MAX^OUTPUT^LEN, INTERNAL, INTERNAL^SCALE,
   FILL^CHAR, FILL^OFF );

INT     USERCODE;         !  Supplied by TCP
STRING  .EXT OUTPUT;      !  Generated by user procedure
INT     .OUTPUT^LEN;      !  Supplied by TCP; modifiable by user
INT     MAX^OUTPUT^LEN    !  Supplied by TCP
FIXED   .INTERNAL;        !  Supplied by TCP
INT     INTERNAL^SCALE;   !  Supplied by TCP
STRING  .FILL^CHAR;       !  Supplied by TCP
INT     FILL^OFF;         !  Supplied by TCP
```

**Figure 4-11. Device Alphanumeric Output Procedure Declaration**

```
PROC USER^ALPHA^OUTPUT^MSG^CONV ( USERCODE, OUTPUT,
   OUTPUT^LEN, MAX^OUTPUT^LEN, INTERNAL, INTERNAL^LEN,
   FILL^CHAR, FILL^OFF, RIGHT^JUSTIFIED );

INT     USERCODE;          !   Supplied by TCP
STRING  .EXT OUTPUT;       !   Generated by user procedure
INT     .OUTPUT^LEN;       !   Supplied by TCP; modifiable by
user
INT     MAX^OUTPUT^LEN     !   Supplied by TCP
STRING  .EXT INTERNAL      !   Supplied by TCP
INT     INTERNAL^LEN;      !   Supplied by TCP
STRING  .FILL^CHAR;        !   Supplied by TCP
INT     FILL^OFF;          !   Supplied by TCP
```

`USERCODE`

contains the numeric literal specified in the USER CONVERSION clause. The procedure uses the supplied value of this parameter to determine which subset of code to execute.

`OUTPUT`

contains the string of characters to be sent to the intelligent device. The TCP expects OUTPUT to contain the value from INTERNAL as converted by this procedure.

`OUTPUT^LEN`

contains the number of characters in the output message (the data pointed to by INTERNAL). If the user conversion procedure changes this length, it should set

OUTPUT^LEN to the new length. In no case should OUTPUT^LEN be greater than MAX^OUTPUT^LEN. OUTPUT^LEN can vary from the user-defined length in cases of delimited-format messages.

`MAX^OUTPUT^LEN`

represents the maximum possible length of the particular converted output field. This value should be used to control the maximum amount of data moved to OUTPUT.

`INTERNAL`

points to the data to be converted. The procedure converts this data and stores it in OUTPUT.

`INTERNAL^SCALE`

is a parameter in the numeric procedure only. It contains the number of decimal places in INTERNAL when INTERNAL is a FIXED field.

`INTERNAL^LEN`

is a parameter in the alphanumeric procedure only. It contains the number of characters in INTERNAL when INTERNAL is a STRING field.

`FILL^CHAR`

is always set to either 0 (for numeric conversions) or blank (for alphanumeric conversions) by the TCP.

`FILL^OFF`

is always set to -1 by the TCP.

`RIGHT^JUSTIFIED`

contains -1 (TRUE) if an alphanumeric value is to be right justified and contains 0 (FALSE) if the data is not to be justified.

# 5 Managing Transactions With the TMF Subsystem

This section provides information about the Compaq Transaction Management Facility (TMF) subsystem for SCREEN COBOL programmers. It also briefly discusses PATHCOM and SPI parameter options for Pathway applications that use SCREEN COBOL requesters and the TMF subsystem.

The general environment for Pathway applications that use SCREEN COBOL requesters and the TMF subsystem is a requester/server environment where SCREEN COBOL programs accept input from terminal operators and transform the input into requests to servers for database services. The servers, in turn, satisfy the requests by reading, locking, and changing (or adding or deleting) records in audited database files. Servers can be written in C, C++, COBOL85, pTAL, TAL, FORTRAN, or Pascal, and they must follow the record-locking rules imposed by the TMF subsystem.

To write application requesters that use the TMF subsystem, you should be familiar with the following information, which is provided in this section:

- The recommended structure for applications that use the TMF subsystem
- How to use the SCREEN COBOL statements that support the TMF subsystem

For information about writing servers that use the TMF subsystem, you should refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

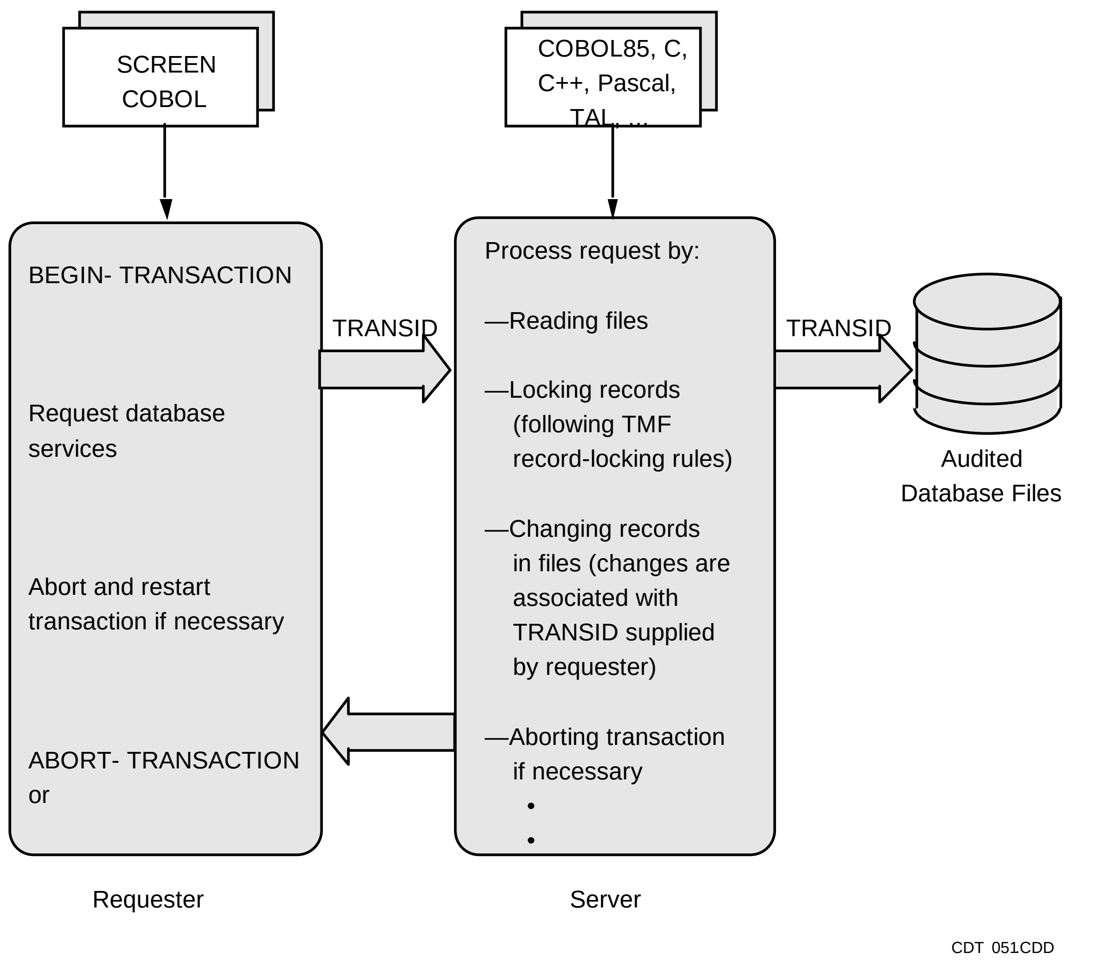
In addition, you should refer to the *NonStop™ TM/MP Application Programmer's Guide* for discussions of the following topics related to the TMF subsystem:

- How to access audited database files
- General guidelines for coding servers
- Locking rules that must be followed by processes that change records in audited database files
- How to cope with transaction deadlock
- Anomalies that can occur during transaction backout

## Task Overview

Figure 5-1 illustrates the basic tasks involved in programming Pathway applications that use SCREEN COBOL requesters and the TMF subsystem.

**Figure 5-1. Pathway Application Programming for the TMF Subsystem**


SCREEN COBOL
COBOL85, C, C++, Pascal, TAL, ...
BEGIN- TRANSACTION
Request database services
Abort and restart transaction if necessary
ABORT- TRANSACTION or
TRANSID
Process request by:
—Reading files
—Locking records (following TMF record-locking rules)
—Changing records in files (changes are associated with TRANSID supplied by requester)
—Aborting transaction if necessary
•
•
TRANSID
Audited Database Files
Requester
Server
CDT 051CDD

## TMF Application Structure

This subsection describes the recommended structure for applications that use the TMF subsystem.

One process (usually the SCREEN COBOL requester) coordinates all of the work required to do a single TMF transaction; this process identifies the beginning and ending points of each transaction. Additionally, if the server replies to a request message by indicating that it failed to complete all of the changes, this process can either abort and abandon the transaction or abort and retry the transaction according to the SCREEN COBOL application.

The communication between requesters and servers is by standard interprocess I/O. The SCREEN COBOL requester does the SEND operation, and the server does the READUPDATE call for $RECEIVE and the REPLY call. Each request message and the server's reply to the message is for a single transaction.

Any disk I/O request is for a single transaction. The TMF subsystem appends the process's current transaction identifier to each disk-request message so that the audit trails (transaction logs) can include the identity of the transaction responsible for each database change.

Servers should not reply to request messages until all work for the request has been completed. The contents of the reply message indicate the outcome of the request, which is one of the following:

- All the work for the request was completed successfully.
- None of the work for the request was completed.
- The work for the request was only partially completed.

In the first case, the requester can commit the transaction. In the second case, the requester can abort the transaction and then retry it. In both these cases, the information in the server's reply is sufficient to ensure the integrity of the transaction.

However, if the transaction work was only partially completed, the application needs to ensure that the transaction is not committed so that the incomplete work can be backed out. To ensure transaction backout, the server should call the ABORTTRANSACTION procedure after reading the request and before sending its reply. A call to ABORTTRANSACTION by the server does not end the transaction—only the requester can end it—but such a call imposes the requirement that the requester use the ABORT-TRANSACTION statement, rather than the END-TRANSACTION statement, after the requester's reply.

The remainder of this section contains detailed information related to writing SCREEN COBOL requesters that use the TMF subsystem.

## TMF Programming in SCREEN COBOL

The SCREEN COBOL language provides the following features to support TMF transactions:

- SCREEN COBOL statements that begin and end a transaction, abort a transaction, and restart a transaction
- Special registers TRANSACTION-ID, TERMINATION-STATUS, and RESTART-COUNTER

### Transaction Mode Use

A terminal program unit (that is, a SCREEN COBOL program executing on behalf of a terminal) configured for use with the TMF subsystem enters transaction mode when the BEGIN-TRANSACTION statement is executed and leaves transaction mode when the END-TRANSACTION or ABORT-TRANSACTION statement is executed.

When BEGIN-TRANSACTION is executed, the transaction is assigned a unique transaction identifier that distinguishes one transaction from all other transactions. If the program unit is configured with TMF OFF, the TCP does not allow that program unit to enter transaction mode, but causes BEGIN-TRANSACTION to issue a null transaction identifier.

When END-TRANSACTION or ABORT-TRANSACTION is executed, the transaction identifier is discarded and can no longer be used.

For the PATHCOM or SPI SUSPEND, STOP, or FREEZE commands, the effect of operating in transaction mode is like setting the STOP-MODE special register to a nonzero value; none of these commands can take effect until the terminal leaves transaction mode and the terminal STOP-MODE register is 0.

The SUSPEND! and FREEZE! commands take effect immediately and cause transaction backout.

The ABORT command takes effect immediately. If the terminal is in transaction mode when this command is executed, the transaction is aborted.

For details regarding SUSPEND, FREEZE, STOP, and ABORT, refer to the *NonStop™ TS/MP System Management Manual* and the *Compaq NonStop™ Pathway/iTS System Management Manual*.

**Note.** To work appropriately with the ORDERLY option of the PATHCOM SHUTDOWN2 command, SEND requests must be coded so that a terminal can be stopped after the last I/O operation in the logical transaction completes. In other words, the requester must end or abort the current transaction after the last SEND request in the transaction, based on the server's reply to the SEND request. For details regarding the SHUTDOWN2 command, refer to the *NonStop™ TS/MP System Management Manual*.

## SCREEN COBOL Verbs for the TMF Subsystem

In a SCREEN COBOL requester, you invoke the functions of the TMF subsystem by using the following transaction-control statements:

- ABORT-TRANSACTION aborts and backs out a transaction.
- BEGIN-TRANSACTION begins a transaction.
- END-TRANSACTION ends a transaction.
- RESTART-TRANSACTION backs out a transaction and then starts it from the BEGIN-TRANSACTION point with a new transaction identifier.

When you use these transaction statements in your SCREEN COBOL programs, Pathway/iTS handles a number of failure cases itself by automatically aborting the transaction and restarting it at the BEGIN-TRANSACTION point. The TCP does the following:

- Takes care of all details involved in handling concurrent active transactions
- Keeps track of the transaction identifiers for multiple transactions
- Checkpoints the transaction identifier
- Operates as a fault-tolerant process pair
- Handles the TMF programming involved when the backup process takes over

You should include an ON ERROR clause in each BEGIN-TRANSACTION statement and provide coding to check for file-system errors that could occur on any of the transaction statements. Failure to perform these checks could cause important parts of your application to fail. For a list of the file-system errors that can be returned by the TMF procedure calls corresponding to these transaction statements, refer to the *NonStop™ TM/MP Application Programmer’s Guide*.

## ABORT-TRANSACTION Use

Generally the ABORT-TRANSACTION statement is used when the SCREEN COBOL program detects an unrecoverable error and abandons the transaction. When this statement is executed, the transaction is aborted; all updates made by the transaction to audited data files are backed out. The aborted transaction is not restarted automatically.

The form of the ABORT-TRANSACTION statement is:

```
ABORT-TRANSACTION
```

Execution of the ABORT-TRANSACTION statement causes the terminal to leave transaction mode and sets the special register TRANSACTION-ID to SPACES.

If the terminal is not in transaction mode when ABORT-TRANSACTION is executed, it is suspended; in such a case terminal execution cannot be restarted with a PATHCOM or SPI RESUME command.

If a fatal error occurs while the transaction is being aborted and the current BEGIN-TRANSACTION statement does not have an ON ERROR clause, the terminal is suspended; in such a case the current transaction is backed out and terminal execution cannot be resumed with a RESUME command. If the BEGIN-TRANSACTION statement has an ON ERROR clause, that clause is executed and the terminal is not suspended.

## BEGIN-TRANSACTION Use

The BEGIN-TRANSACTION statement begins a new transaction; this statement identifies the beginning of a sequence of operations that are treated by the TMF subsystem as a single transaction. When this statement is executed, the following occurs:

- The terminal enters transaction mode.
- The TMF subsystem is requested to begin a new transaction.
- The transaction identifier for the new transaction is assigned to the TRANSACTION-ID special register.
- RESTART-COUNTER and TERMINATION-STATUS special registers are reset to 0 for the first occurrence of the transaction.

The form of the BEGIN-TRANSACTION statement is:

```
BEGIN-TRANSACTION [ ON ERROR imperative-statement ]
```

The BEGIN-TRANSACTION statement indicates the restarting point to be used if a failure occurs while the terminal is in transaction mode. If the transaction fails for any reason, its database changes are backed out. Except when the SCREEN COBOL program issues an ABORT-TRANSACTION, execution of the SCREEN COBOL program can be restarted at that point if these conditions are met:

- If the ON ERROR clause is omitted, the TCP compares the number of times that the transaction has been restarted with the global transaction-restart limit specified with the MAXTMFRESTARTS option of the SET PATHWAY command in PATHCOM. If the number of restarts is less than that limit, the transaction is restarted with a new transaction identifier, the RESTART-COUNTER special register is incremented by 1, and the TERMINATION-STATUS special register remains set to 1. If the number of restarts equals the transaction-restart limit, the terminal is suspended but its execution can be resumed manually.
- If ON ERROR is present and if the requester restarts the transaction, RESTART-COUNTER is incremented by 1, TERMINATION-STATUS remains set to 1, and the ON ERROR branch is executed. The ON ERROR branch of the SCREEN COBOL program can then include a check to determine whether or not the transaction should be restarted; for example, the program can compare RESTART-COUNTER to a local restart limit established within the program.

If the terminal is already in transaction mode when BEGIN-TRANSACTION is issued, it is suspended; in such a case the current transaction is backed out and terminal execution cannot be resumed with a PATHCOM or SPI RESUME command.

The following code sequence accepts input data from the operator and starts a new transaction. In the event of an error, the ON ERROR code tests the RESTART-COUNTER to determine if the particular transaction has been restarted more than two times. If the transaction has been started more than twice, it is aborted and the operator is asked to enter the data again. If the transaction has not been restarted more than two times, the TCP makes another attempt to process the transaction.

```
enter-data
    .
    .
ACCEPT screen...
BEGIN-TRANSACTION
    ON ERROR PERFORM check-error.
IF abort-flag NOT = 0
    GO TO enter-data.
    .
    .
SEND ...
END-TRANSACTION.
.
.
stop-trans.
    GO TO enter-data.
    .
    .
check-error.
 MOVE 0 TO abort-flag.
IF TERMINATION-STATUS = 1
```

```
    IF RESTART-COUNTER > 2
       ABORT-TRANSACTION
       DISPLAY "No" IN MSG
       MOVE 1 TO abort-flag.
```

### END-TRANSACTION Use

The END-TRANSACTION statement indicates that the transaction is complete. When this statement is successfully executed, the database updates made by the transaction become permanent, the terminal leaves transaction mode, and the special register TRANSACTION-ID is set to SPACES.

If the TMF subsystem rejects END-TRANSACTION, the SCREEN COBOL program is restarted at the previous BEGIN-TRANSACTION point.

The form of the END-TRANSACTION statement is:

```
END-TRANSACTION
```

If the terminal is not in transaction mode when END-TRANSACTION is executed, it is suspended; in such a case terminal execution cannot be resumed with a RESUME command.

### RESTART-TRANSACTION Use

The RESTART-TRANSACTION statement is used when the SCREEN COBOL program detects an error that might be temporary, abandons the current attempt, and retries the transaction. When this statement is executed, the following occurs:

- The current execution of the transaction is backed out.
- The transaction is restarted at the previous BEGIN-TRANSACTION point with a new transaction identifier.
- The special register RESTART-COUNTER is incremented by 1.

The form of the RESTART-TRANSACTION statement is:

```
RESTART-TRANSACTION
```

The restart due to executing RESTART-TRANSACTION counts as a restart for purposes of the global transaction-restart limit.

If the terminal is not in transaction mode when RESTART-TRANSACTION is executed, the terminal is suspended; in such a case terminal execution cannot be resumed with a PATHCOM or SPI RESUME command.

## SCREEN COBOL Special Registers for the TMF Subsystem

Special registers are data items defined automatically by the SCREEN COBOL compiler, not by the programmer. Three special registers have been provided for TMF subsystem users: TRANSACTION-ID, TERMINATION-STATUS, and RESTART-COUNTER.

- TRANSACTION-ID

  Executing BEGIN-TRANSACTION sets TRANSACTION-ID to the value of the transaction identifier. Executing END-TRANSACTION or ABORT-TRANSACTION sets this register to SPACES.

  TRANSACTION-ID has this implicit declaration:

  ```
  01 TRANSACTION-ID     PIC X(8).
  ```

- TERMINATION-STATUS

  Executing BEGIN-TRANSACTION sets the value of TERMINATION-STATUS to indicate the outcome of BEGIN-TRANSACTION. The following values are possible:

  | | |
  |---|---|
  | 1 | The transaction is started or restarted. |
  | 2 | The TMF subsystem is not installed. If there is no ON ERROR clause, the default system action is to suspend the terminal for the pending abort. |
  | 3 | The TMF subsystem is not started. If there is no ON ERROR clause, the default system action is to suspend the terminal, but the terminal can be restarted by the PATHCOM or SPI RESUME command. |
  | 4 | A fatal error occurred. If there is no ON ERROR clause, the default system action is to suspend the terminal for the pending abort. |

  TERMINATION-STATUS has this implicit declaration:

  ```
  01 TERMINATION-STATUS     PIC 9999 COMP.
  ```

- Executing BEGIN-TRANSACTION sets RESTART-COUNTER to the number of times the transaction has been restarted. RESTART-COUNTER is reset to 0 when BEGIN-TRANSACTION is first executed for a particular transaction.

  RESTART-COUNTER has this implicit declaration:

  ```
  01 RESTART-COUNTER     PIC 9999 COMP.
  ```

  See BEGIN-TRANSACTION Use on page 5-5 for an example of how to use RESTART-COUNTER to limit selectively the number of times a transaction is retried.

## Interaction Between the PATHMON Environment and the TMF Subsystem

When you are configuring and controlling Pathway applications that include SCREEN COBOL requesters and use the TMF subsystem, you need information about three basic questions related to the interaction between the PATHMON environment and the TMF subsystem:

- How do the settings you specify for the TMF parameter of the PATHCOM SET SERVER, SET TERM, and SET PROGRAM commands affect SCREEN COBOL SEND statements?

- How is TCP checkpointing strategy affected by the settings you specify for the TMF parameter of the SET SERVER command?
- What problems are caused by using the TMF OFF option of the SET TERM or SET PROGRAM commands as a switch to turn TMF subsystem operation off for a SCREEN COBOL requester that is communicating with servers running under the TMF subsystem?

Understanding the answers to these questions ensures the consistency of the database and helps you to improve the reliability and performance of the applications that use the database.

## SET SERVER Command and the TMF Subsystem

The SET SERVER command contains a TMF parameter with an ON or OFF option. By setting this parameter you control how a TCP allows access to a server class, that is, the types of operations a server class can perform.

- TMF ON means the TCP allows a SEND operation to members of this server class whether or not the SCREEN COBOL program is in transaction mode.
- TMF OFF means the TCP allows a SEND operation to the members of this server class only if the SCREEN COBOL program is not in transaction mode. OFF is the default setting.

In addition, the TCP makes checkpointing decisions based in part upon the option specified for the TMF parameter. You must match the TMF parameter setting to the application environment. For further information, see TCP Checkpointing Strategy on page 5-12.

## SET TERM and SET PROGRAM Commands and the TMF Subsystem

The SET TERM and SET PROGRAM commands each contain a TMF parameter with an ON or OFF option.

- TMF ON causes the TCP to invoke the corresponding *Guardian* procedure call for any TMF statement issued from a SCREEN COBOL program. This setting allows your SCREEN COBOL program to perform SEND operations to a server for which the TMF parameter is set to ON. ON is the default setting whether or not the TMF subsystem is running.
- TMF OFF does not cause the TCP to invoke a corresponding Guardian procedure call for any TMF statement issued from a SCREEN COBOL program. Instead, the TMF statement appears (to the SCREEN COBOL program) to complete successfully, and the program can continue to execute.

For most Pathway applications, you should use the default parameter settings, whether or not the TMF subsystem is running.

## Effect of TMF Parameters on SCREEN COBOL SEND Operations

Table 5-1 illustrates how the various combinations of settings of the TMF parameter in the PATHCOM SET TERM, SET PROGRAM, and SET SERVER commands affect a SCREEN COBOL SEND statement when the PATHMON process and the TMF subsystem are both running on the system. Depending on the type of file access attempted, the TCP either allows the SEND statement to execute or issues the appropriate error message.

**Table 5-1. SEND Operations With the TMF Subsystem** (page 1 of 2)

| PATHCOM Commands | | Audited Files | |
|---|---|---|---|
| | | **Transaction Mode** | **Nontransaction Mode** |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF ON<br>TMF ON<br>TMF ON | SEND statement executes[1] | SEND statement executes. possible file-system error 75 in server[2] |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF ON<br>TMF OFF<br>TMF OFF | SEND statement executes. possible file-system error 75 in server[2] | SEND statement executes. possible file-system error 75 in server[2] |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF OFF<br>TMF ON<br>TMF ON | SEND error 13[3] | SEND statement executes. possible file-system error 75 in server[2] |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF OFF<br>TMF OFF<br>TMF OFF | SEND error 13[3] | SEND statement executes. possible file-system error 75 in server[2] |

| PATHCOM Commands | | Non-Audited Files | |
|---|---|---|---|
| | | **Transaction Mode** | **Nontransaction Mode** |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF ON<br>TMF ON<br>TMF ON | SEND statement executes[4] | SEND statement executes[5] |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF ON<br>TMF OFF<br>TMF OFF | SEND statement executes[4] | SEND statement executes[5] |

**Table 5-1. SEND Operations With the TMF Subsystem** (page 2 of 2)

| PATHCOM Commands | | | Audited Files |
|---|---|---|---|
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF OFF<br>TMF ON<br>TMF ON | SEND error 13[3] | SEND statement executes[5] |
| SET SERVER<br>SET TERM<br>SET PROGRAM | TMF OFF<br>TMF OFF<br>TMF OFF | SEND error 13[3] | SEND statement executes[5] |

**LEGEND**

**Transaction Mode.** The SEND statement is executed after the SCREEN COBOL program has issued a BEGIN-TRANSACTION statement, but before the program has issued an END-TRANSACTION or an ABORT-TRANSACTION statement. Note that a program is considered to be in transaction mode if it executes a BEGIN-TRANSACTION statement even if the TMF parameter is set to OFF for the terminal or program.

**Note 1.** These are PATHCOM command parameter settings for normal TMF and Pathway operations; SET SERVER TMF ON must be set within PATHCOM or SPI.

**Note 2.** Although no transaction identifier was present, Pathway/iTS has allowed the SEND operation. If the server attempts a file lock or update operation on an audited file, Guardian file-system error 75 is returned to the server. How this information is returned by the server to the SCREEN COBOL requester is application dependent.

**Note 3.** TMF mode violation: the error is returned in the SCREEN COBOL TERMINATION-STATUS special register.

**Note 4.** These are PATHCOM command parameter settings for special program testing. These settings provide a convenient way to partially test or debug a SCREEN COBOL program on a system that does not yet have the TMF subsystem configured. The program will execute, but all SEND requests to audited files will receive Guardian file-system error 75 replies.

**Note 5.** These are PATHCOM command parameter settings for normal Pathway operations without the TMF subsystem.

In a PATHMON environment that normally runs with the TMF subsystem, do not use the following commands to turn off TMF subsystem operations temporarily:

```
SET SERVER  TMF ON
SET TERM  TMF OFF
SET PROGRAM  TMF OFF
```

The condition resulting from these commands appears to allow normal operation because the BEGIN-TRANSACTION statement that would have failed if the TMF subsystem were stopped now appears to work; the TCP allows a SEND operation to a server that can access and update only nonaudited files. Files updated by servers are not protected by the TMF subsystem, and the TCP does not perform checkpoints before or after SEND statements.

## Timeouts on SEND Operations to Servers

Although the syntax of the SCREEN COBOL SEND statement does not include a TIMEOUT clause, you can effectively supply one with the PATHCOM SET SERVER command. When you include a TIMEOUT clause in the SET SERVER command, all SENDs to that server class are timed by the TCP. If the specified number of seconds elapses after a SEND operation is initiated and before a reply is received, the TCP issues a Guardian CANCEL procedure call against the outstanding I/O to the server. If the SEND operation was performed while the requester program was in transaction mode, the transaction is automatically aborted by the file system. In such a case the requester program discovers that the transaction was aborted when it subsequently attempts to update the database (with another SEND statement) or issues an END-TRANSACTION statement.

## TCP Checkpointing Strategy

In a PATHMON environment with the TMF subsystem running, the TCP uses the following checkpointing strategy:

- At the BEGIN-TRANSACTION statement, a full copy of the task's context is made to a secondary area (slot 1) in the extended data segment, and a checkpoint to the backup is performed.
- At the END-TRANSACTION statement, a full-context checkpoint is performed.
- At the SEND statement with the SET SERVER TMF parameter defined as OFF, a checkpoint is performed before and after the SEND statement when the SCREEN COBOL program is outside of transaction mode.

  Any time a SEND operation is performed outside of a transaction boundary and the server attempts to lock or update a record in an audited file, the operation fails with a Guardian file-system error 75.
- At the SEND statement with the SET SERVER TMF parameter defined as ON, no checkpoints are performed, whether or not the SCREEN COBOL program is in transaction mode. Therefore, SEND requests to TMF protected servers that operate on audited files require fewer checkpoints than SEND requests to servers that do not operate under TMF protection.

TCP checkpointing requirements can be reduced significantly if Pathway applications that use the TMF subsystem have the servers read outside of transaction mode before updating the database.

△ **Caution.** If a SEND request outside of transaction mode is sent to a TMF protected server that operates on nonaudited files, data might be lost because the TMF subsystem is not invoked, and the TCP performs fewer checkpoints.

You can improve the performance of a Pathway application by taking advantage of the TCP checkpointing strategy for TMF protected servers, as follows:

- Do not use transaction mode for a server with read-only access to a database if the requester displays the data before any attempt is made to change the data. In the event of a failure, the requester can retry the read operations and fault-tolerant operation is maintained.
- Do not use transaction mode for a server that writes to an entry-sequenced logging file in which duplicates are acceptable. In the event of a failure, the requester can retry the write operations, so there is no need to back out the write. In contrast, a key-sequenced file requires a backout; otherwise, the transaction fails when the second write is attempted at the same location.

## Precautions for Using TMF Parameters

If a TMF error occurs and makes normal operation impossible, you should not try to solve the problem by setting the PATHCOM TMF parameter options to OFF. Setting these options to OFF can have the following results:

- A server intended for operation with TMF protection probably does not send the checkpoint messages necessary to function as a fault-tolerant server when the TMF subsystem is not invoked.
- A SCREEN COBOL program that uses ABORT-TRANSACTION or RESTART-TRANSACTION statements to handle exceptions to normal program operation only appears to execute; the ABORT-TRANSACTION or RESTART-TRANSACTION statements have no effect.
- With the SET SERVER TMF parameter defined as ON and the SET TERM or SET PROGRAM TMF parameters defined as OFF, the TCP sends checkpoint messages, performs retries, and sets the sync depth as if the TMF subsystem were running. For example, the TCP performs fewer checkpoints and opens servers with a sync depth of 0 instead of 1. In this case, the TCP does not take full advantage of fault tolerance, and a single CPU failure can cause the application to fail.

Refer to the *NonStop™ TM/MP Operations and Recovery Guide* to determine how to address the TMF error.

# 6 Programming for Intelligent Devices

Standard SCREEN COBOL requesters interact with a limited set of video display terminals. Standard requesters are screen-oriented; they send data from working storage to the display screen of a terminal by using screen templates defined in the Screen Section of the Data Division. Similarly, they receive data from the terminal into working storage by using Screen Section templates. Standard requesters use ACCEPT and DISPLAY statements in the Procedure Division to interact with the display terminals.

Intelligent device support (IDS) SCREEN COBOL requesters interact either directly or indirectly with intelligent devices such as automated teller machines, airline reservation terminals, and personal computers. IDS requesters are message-oriented**;** they send data from working storage to the device (or to a front-end process that controls the device) and receive data from the device or process into working storage by using Message Section templates. IDS requesters use SEND MESSAGE statements and their REPLY clauses in the Procedure Division to interact with the intelligent devices or front-end processes.

Because of the wide variety of message formats that IDS devices must be able to process, the IDS extensions to Compaq *NonStop*™ Pathway/iTS have been designed to provide the following capabilities:

- The ability to use field delimiters, message delimiters, and RESULTING COUNT clauses to process compacted messages containing variable-length fields
- The ability to use a TRANSFORM statement to move data elements from source data structures in working storage to target data structures and, in the process of doing so, reorder and convert the data by using one or more templates in the Message Section
- The ability to use fields within Message Section templates to determine, on either input or output, which fields that follow in the template are present in an actual message

In addition, there are other capabilities (such as data-item editing and conversion, scatter reads, and gather writes) used in the standard requester environment that also apply to the IDS environment.

This section describes both the IDS extensions and the standard capabilities that apply to IDS requesters. It presents information about the following topics:

- Use of the SEND MESSAGE statement
- Use of delimiters and RESULTING COUNT clauses
- Use of the TRANSFORM statement
- Use of PRESENT IF clauses
- IDS error processing and debugging techniques

## The SEND MESSAGE Statement

The SEND MESSAGE statement moves data from working storage to an external process (outside the PATHMON environment). The associated REPLY statement accepts data from the external process and moves it into working storage.

Besides moving the data, both the SEND MESSAGE statement and its REPLY statement edit and convert the individual data fields according to the pertinent PICTURE clauses in the Working-Storage Section and the Message Section data declarations.

The SEND MESSAGE statement also allows you to use the gather-write capability (whereby individual items in the outgoing message are extracted from noncontiguous locations in working storage). The associated REPLY statement allows you to use the scatter-read capability (whereby individual items in the incoming message are stored into noncontiguous locations in working storage).

When you send data to the process or device, you can use only a single message template. When you receive data from the process or device, however, you can do so through any of several input message templates. You use the REPLY CODE and YIELDS clauses with the SEND MESSAGE statement to determine which input message template the response has been mapped through.

You code SEND MESSAGE statements in a manner similar to the way in which you code SEND statements (by using REPLY CODE and YIELDS clauses and the TERMINATION-STATUS register). An example of the use of the SEND MESSAGE statement is as follows:

```
SEND MESSAGE MSG-3-OUT
 REPLY CODE FIELD IS WS-MSG-IN-CODE
    CODE 1 YIELDS MSG-3-IN
    CODE 2 YIELDS MSG-4-IN
    CODE 3 YIELDS MSG-3-IN
    CODE 4 YIELDS MSG-4-IN
 ON ERROR PERFORM IDS-SERVER-SEND-ERROR.


PERFORM ONE OF
    PROC-MSG-3-IN
    PROC-MSG-4-IN
    PROC-MSG-3-IN
    PROC-MSG-4-IN
       DEPENDING ON TERMINATION-STATUS.
```

## Using Delimiters and the RESULTING COUNT Clause

Among the IDS extensions to the SCREEN COBOL programming language is one that provides the ability to send and receive messages that contain message-delimiter and field-delimiter characters.

The use of delimiters makes it possible for your requester and the external device or front-end process to exchange compact variable-length messages efficiently.

The presence and use of delimiters is not an optional choice for you but rather something that is dictated by the design characteristics of whatever entity your requester is communicating with. When the external device or process uses a message delimiter, your message template must also declare the same message delimiter. When the external device or process uses a field delimiter, your message template must also declare the same field delimiter.

### Declaring Delimiters

You declare a message format to be delimited by including a MESSAGE FORMAT IS DELIMITED clause or a MESSAGE FORMAT IS FIXED-DELIMITED clause at the 01 level of the particular message template in the Message Section. The only difference between the two formats is that DELIMITED messages contain entirely variable-length fields while FIXED-DELIMITED messages contain entirely fixed-length fields.

Having declared the message format to be delimited, you then declare (again at the 01 level):

1. Whether the message includes a message delimiter and, if so, what characters it consists of (The default message delimiter is //.)
2. Whether the message includes field delimiters and, if so, what characters they consist of (The default field delimiter is , [comma].)

### Sample Declarations

In the examples that appear later in this section, assume that the following data structures have been declared in the Working-Storage Section and the Message Section:

```
DATA DIVISION.

  WORKING-STORAGE SECTION.

     01 WS-ITEM1.
        05 WS-ITEM1-CNT    PIC 9(2) COMP.
        05 WS-ITEM1-GROUP.
           10 WS-ITEM1-DATA   PIC X(1)
                       OCCURS 1 TO 30 TIMES
                       DEPENDING ON ws-item1-cnt.
```

```
    01 WS-ITEM2.
       05 WS-ITEM2-CNT    PIC 9(2) COMP.
       05 WS-ITEM2-GROUP.
          10 WS-ITEM2-DATA   PIC X(1)
                      OCCURS 1 TO 30 TIMES
                      DEPENDING ON ws-item2-cnt.

    01 WS-ITEM3.
       05 WS-ITEM3-CNT    PIC 9(2) COMP.
       05 WS-ITEM3-GROUP.
          10 WS-ITEM3-DATA   PIC X(1)
                      OCCURS 1 TO 30 TIMES
                      DEPENDING ON ws-item3-cnt.

       01 WS-ITEM4.
          05 WS-ITEM4-CNT    PIC 9(2) COMP.
          05 WS-ITEM4-GROUP.
             10 WS-ITEM4-DATA   PIC X(1)
                         OCCURS 1 TO 30 TIMES
                         DEPENDING ON ws-item4-cnt.

       01 WS-ITEM5.
          05 WS-ITEM5-CNT    PIC 9(2) COMP.
          05 WS-ITEM5-GROUP.
             10 WS-ITEM5-DATA   PIC X(1)
                         OCCURS 1 TO 30 TIMES
                         DEPENDING ON ws-item5-cnt.


 MESSAGE SECTION.

    01 msg-format4          MESSAGE FORMAT IS DELIMITED
                            MESSAGE DELIMITER IS OFF.

       05 item1             PIC X(30)
                            USING ws-item1-group
                            RESULTING COUNT IS ws-item1-cnt.

       05 item2             PIC X(30)
                            USING ws-item2-group
                            RESULTING COUNT IS ws-item2-cnt.

       05 item3             PIC X(30)
                            USING ws-item3-group
                            RESULTING COUNT IS ws-item3-cnt.

       05 item4             PIC X(30)
                            USING ws-item4-group
                            RESULTING COUNT IS ws-item4-cnt.

       05 item5             PIC X(30)
                            USING ws-item5-group
                            RESULTING COUNT IS ws-item5-cnt.
```

The message template declares that the associated message will contain a field delimiter (a comma, by default) but no message delimiter.

Each field in the message template is fixed in length, to accommodate the maximum-size data item, whereas the corresponding fields in working storage are variable in length and rely on the content of an associated count field to determine their length.

## Processing Field Delimiters on Input

When a message is received from a Message Section template MSG-FORMAT4, the data for each field is stored in the working-storage item specified by the associated USING clause, and a count of the actual number of characters received before encountering the field delimiter is stored in the working-storage item specified by the associated RESULTING COUNT clause.

For example, suppose that the incoming message is as follows:

```
JOHN DOE,MARIA GONZALES,WILLIAM DEFOE,TONY ALLEN,SUE QUICK,
```

and that it is received from the Message Section template MSG-FORMAT4:

```
SEND MESSAGE
 YIELDS msg-format4
 ON ERROR GO TO error-exit.
```

Upon successful completion of the SEND MESSAGE statement, the various working-storage data items referred to by the MSG-FORMAT4 message-template declaration contain the following data. (The apostrophes merely illustrate the beginning and ending of each field; they do not actually occupy any space within the fields themselves.)

```
ws-item1-cnt      '8'
ws-item1-data     'JOHN DOE'

ws-item2-cnt      '14'
ws-item2-data     'MARIA GONZALES'

ws-item3-cnt      '13'
ws-item3-data     'WILLIAM DEFOE'

ws-item4-cnt      '10'
ws-item4-data     'TONY ALLEN'

ws-item5-cnt      '9'
ws-item5-data     'SUE QUICK'
```

## Using Field Delimiters on Output

When sending a message to the external device or front-end process by using a Message Section template MSG-FORMAT4, you move the appropriate data values and the byte counts into the working-storage structure and then issue a SEND MESSAGE statement specifying the MSG-FORMAT4 template.

For example, suppose you want to send the following message to the external device or process:

```
BILL WINN,GIUSEPPE PINELLI,JOE BLOW,LING CHIN,SARAH HARRAH,
```

To do so, you would move the following values to the specified working-storage locations:

```
'BILL WINN'                  ws-item1-data
'9'                          ws-item1-cnt

'GIUSEPPE PINELLI'           ws-item2-data
'16'                         ws-item2-cnt

'JOE BLOW'                   ws-item3-data
'8'                          ws-item3-cnt

'LING CHIN'                  ws-item4-data
'9'                          ws-item4-cnt

'SARAH HARRAH'               ws-item5-data
'12'                         ws-item5-cnt
```

and then issue a SEND MESSAGE statement such as:

```
SEND MESSAGE msg-format4
 ON ERROR GO TO error-exit.
```

Before moving a data element from working storage to its output buffer, the TCP examines the location referenced by the associated RESULTING COUNT clause to find out how many bytes of data the field actually contains. After retrieving the specified number of bytes from the particular working-storage location, the TCP appends a field-delimiter character (in this case, a comma) to the end of the outbound field.

Thus, you can directly control the placement of field delimiters in the output stream.

## Using Message Delimiters

The use of a RESULTING COUNT clause at the message level allows you to determine, on input, how long an incoming variable-length record was, without having to do a backward search through your working-storage data structure.

If the external device or process requires the message to include a message delimiter, you must declare the message delimiter in your message template; otherwise, the TCP mistakes the message-delimiter characters for actual data.

For example, if the external device or process requires that each message be terminated by a colon (:), you must change the beginning of the preceding sample message template declaration to the following:

```
01 msg-format4
   MESSAGE IS DELIMITED
   MESSAGE DELIMITER IS ":".
```

## Using Delimited Format With Delimiters Turned Off

By declaring a message template to be delimited but turning off both the field and message delimiters, you can effectively create a new type of variable-length record format that has no prefix byte count or delimiters.

SCREEN COBOL supports two other types of variable-length message formats, known as VARYING1 and VARYING2, that include a one-byte or two-byte count field at the beginning of each message specifying the total number of bytes contained in the message. The use of delimited format with the delimiters turned off creates a variable-length message with no count field preceding it.

For all three types of messages, if the message contains variable-length fields, the structure of the message must include a count field preceding each individual field. This count-field value allows the receiving device or process to know how long the field is, because no delimiter is present.

The following Working-Storage Section and Message Section declarations define data structures and a message template suitable for sending and receiving this new type of variable-length message.

```
DATA DIVISION.

  WORKING-STORAGE SECTION.

     01 WS-RECORD1-COUNT  PIC 9(4).

     01 WS-RECORD1.

        05 WS-RECORD1-DATA   PIC X(1)
                    OCCURS 1 TO 100 TIMES
                    DEPENDING ON ws-record1-count.

  MESSAGE SECTION.

      01 msg-format1  PIC X(100)  USING ws-record1
         MESSAGE FORMAT IS DELIMITED
         FIELD-DELIMITER IS OFF
         MESSAGE-DELIMITER IS OFF
         RESULTING COUNT IS ws-record1-count.
```

The preceding message template declares that the associated message is of variable length with a maximum size of 100 bytes, that it includes no count field (such as is found in a VARYING1 or VARYING2 format record), and that the TCP is to use the working-storage data item WS-RECORD1-COUNT for storing (on input) or retrieving (on output) the appropriate record length value.

## Using TRANSFORM Statements

The TRANSFORM statement lets you move multiple data items from one place in working storage to another, converting them in the process by a single statement. You can achieve the same results without the TRANSFORM statement, but you must use a whole paragraph of MOVE statements to do so.

The data items specified in a TRANSFORM statement can be any mixture of 01-level, group, or elementary items defined in the Working-Storage Section, the Linkage Section, or the Message Section.

The two primary uses of the TRANSFORM statement are:

- To disassemble incoming messages and scatter and convert, if necessary, the data fields into diverse Working-Storage Section or Linkage Section locations according to codes nested within the message itself.
- To gather, and convert if necessary, individual data items from diverse Working-Storage Section or Linkage Section locations and assemble them into a single message to be passed to a server process by a subsequent SEND statement.

### Example 1: Disassembling Input Messages

Assume the general format of messages being passed between a front-end process and an IDS requester is as follows:

| **4 Bytes** | **2 Bytes** | **Up to 100 Bytes** |
|---|---|---|
| Transmission Header | Select Code | Data or Control Information |

Then assume the presence of the following Working-Storage Section data declarations:

```
01 PROCESSING-STATE          PIC X(4), VALUE "GO  ".

01 MSG-OUT.
  05 TRANSMISSION-HEADER.
     10 OUT-REPLY-CODE       PIC 9(4) comp.
     10 OUT-SESSION-ID       PIC 9(4) comp.
  05 MSG-OUT-DATA.
     10 OUT-SELECT-CODE      PIC 9(4) comp.
     10 OUT-DATA             PIC X(100).

01 MSG-IN.
  05 TRANSMISSION-HEADER.
     10 IN-REPLY-CODE        PIC 9(4) comp.
     10 IN-SESSION-ID        PIC 9(4) comp.
  05 MSG-IN-DATA.
     10 IN-SELECT-CODE       PIC 9(4) comp.
     10 IN-DATA              PIC X(100).

01 CONTROL-RECORD-1.
   05 CTL-FLD-1  PIC X(10).
   05 CTL-FLD-3  PIC X(10).
   05 CTL-FLD-5  PIC X(10).
   05 CTL-FLD-6  PIC X(10).
```

```
01 CONTROL-RECORD-2.
   05 CTL-FLD-2  PIC X(10).
   05 CTL-FLD-4  PIC X(10).
01 DATA-RECORD-1.
   05 DATA-FLD-A  PIC X(10).
   05 DATA-FLD-B  PIC X(10).
   05 DATA-FLD-C  PIC X(10).
   05 DATA-FLD-D  PIC X(10).
   05 DATA-FLD-E  PIC X(10).

01 DATA-RECORD-2.
   05 DATA-FLD-F  PIC X(10).
   05 DATA-FLD-G  PIC X(10).
   05 DATA-FLD-H  PIC X(10).
```

Because the type of information (either control information or data) contained in the message can vary from one transmission to another, there are two levels at which the requester must process such a message:

1. The requester must first accept the entire message and determine, by checking a reply code in the transmission header, whether or not the overall message itself was transmitted and received successfully.

```
MAIN-PARAGRAPH.

   PERFORM send-message-processing
     THRU send-message-processing-exit
     UNTIL processing-state = "STOP".

MAIN-PARAGRAPH-EXIT.

   EXIT.

SEND-MESSAGE-PROCESSING.

   SEND MESSAGE msg-out
     REPLY CODE 0 YIELDS msg-in
     ON ERROR GO TO error-exit.

   PERFORM disassemble-message.

   GO TO send-message-processing-exit.

ERROR-EXIT.

   MOVE "STOP" TO processing-state.

SEND-MESSAGE-PROCESSING-EXIT.

   EXIT.
```

2. If the reply code in the transmission header indicates successful transmission (0 in the preceding case), the requester processes the message as either control information or application data, depending upon the value of the select code within the message itself. The requester does this by using a TRANSFORM statement as follows:

```
DISASSEMBLE-MESSAGE.
  TRANSFORM msg-in-data
     CODE 1 YIELDS ctl-fld-2, ctl-fld-5, ctl-fld-1
     CODE 2 YIELDS control-record-2
     CODE 3 YIELDS data-fld-A, data-fld-F, data-fld-C
     CODE 4 YIELDS data-record-1
     ON ERROR GO TO error-exit.
```

   The TRANSFORM statement is operating upon a subset of the overall message (MSG-IN-DATA), ignoring the transmission header completely. The statement can then operate by using a select code defined within that subset of the message; in this case, by default, the code occurs in the first two bytes of MSG-IN-DATA.

   The message string following the select code varies in length and number of fields, depending upon the value of the select code.

   - If the select code is 1, the TCP extracts three 10-character fields from its input buffer and stores them in the fields named CTL-FLD-2, CTL-FLD-5, and CTL-FLD-1 of the Working-Storage data structures CONTROL-RECORD-2 and CONTROL-RECORD-1.
   - If the select code is 2, the TCP extracts two 10-character fields from its input buffer and stores them in the fields named CTL-FLD-2 and CTL-FLD-4 of the Working-Storage data structure CONTROL-RECORD-2.
   - If the select code is 3, the TCP extracts three 10-character fields from its input buffer and stores them in the fields named DATA-FLD-A, DATA-FLD-F, and DATA-FLD-C of the Working-Storage data structures DATA-RECORD-1 and DATA-RECORD-2.
   - If the select code is 4, the TCP extracts five 10-character fields from its input buffer and stores them in the fields named DATA-FLD-A, DATA-FLD-B, DATA-FLD-C, DATA-FLD-D, and DATA-FLD-E of the Working-Storage data structure DATA-RECORD-1.

   You can specify elementary, group, or 01-level items in the YIELDS lists of the TRANSFORM statement. You can also intermix these three within the same YIELDS list.

## Example 2:  Assembling Output Messages

The next example uses a TRANSFORM statement to gather a set of data values from diverse locations in working storage, to convert them from one format to another, and to assemble them into a completely different field structure to form a single record.  That record will subsequently be sent, through a SEND statement, to a server process.

Assume that the source data structures in working storage are DATA-RECORD-1, DATA-RECORD-2, and DATA-RECORD-3, and the target data structure is SERVER-RECORD-1.

```
01 DATA-RECORD-1.
   05 DATA-FLD-A  PIC A(10).
   05 DATA-FLD-B  PIC 9(10).

01 DATA-RECORD-2.
   05 DATA-FLD-C  PIC 9(3).
   05 DATA-FLD-D  PIC A(5).              54 bytes total
   05 DATA-FLD-E  PIC A(5).

01 DATA-RECORD-3.
   05 DATA-FLD-F  PIC X(10).
   05 DATA-FLD-G  PIC A(5).
   05 DATA-FLD-H  PIC X(6).

01 SERVER-RECORD-1.
   05 SRVR-FLD-1  PIC X(21).
   05 SRVR-FLD-2  PIC 9.
   05 SRVR-FLD-3  PIC 9.                 54 bytes total
   05 SRVR-FLD-4  PIC 9.
   05 SRVR-FLD-5  PIC A(10).
   05 SRVR-FLD-6  PIC X(20).
```

Then assume that the program must transmit the data from the following sequence of source fields to the server process:

```
DATA-FLD-A
DATA-FLD-H
DATA-FLD-G
DATA-FLD-C
DATA-FLD-D     DATA-RECORD-2
DATA-FLD-E
DATA-FLD-B
DATA-FLD-F
```

Notice that the first three and last two fields are from the structures DATA-RECORD-1 and DATA-RECORD-3.  Not only are they separated from one another in the target record, but also they appear in a different order from that defined for them in the DATA-RECORD-1 and DATA-RECORD-2 Working-Storage Section definitions.  You, therefore, must refer to those fields by their elementary data-item names when specifying where the data to be transformed is coming from.

In contrast, because the fourth, fifth, and sixth fields are all from the data structure DATA-RECORD-2 and appear in the same order as the one defined for that structure in working storage, you can refer to them collectively by their shared 01-level name.

When it is time to gather all of the specified data values from their diverse locations in working storage, convert them from alphabetic or numeric to alphanumeric format if necessary, and store them in the proper order into the data structure SERVER-RECORD-1, the requester can use a TRANSFORM statement such as the following:

```
TRANSFORM data-fld-A, data-fld-H, data-fld-G,
          data-record-2, data-fld-B, data-fld-F
   YIELDS server-record-1
   ON ERROR GO TO error-exit.
```

You can specify a mixture of 01-level, group, and elementary items in either the source or target list of the TRANSFORM statement.

When the TRANSFORM statement is executed, the TCP uses the source list to construct a buffer filled with alphanumeric, alphabetic, and numeric bytes. The TCP then disperses the bytes from that buffer to the target data structure on a byte-for-byte basis, converting the data as necessary. When the target list comprises working-storage items, the total number of bytes in the source list must exactly match the total number of bytes in the target list, or a run-time error occurs.

When moving bytes from the buffer to the target structure, all of the standard MOVE statement rules apply, as described in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*. For example, a byte that originates as part of a numeric data item in the source list cannot be moved to an alphabetic data item in the target list; it can, however, be moved to either a numeric or alphanumeric data item. The transfer of data bytes from source to target data items in this particular example can be illustrated as follows:

```
10 bytes     DATA-FLD-A
 6 bytes     DATA-FLD-H          SRVR-FLD-1     21 bytes
 5 bytes     DATA-FLD-G

                                 SRVR-FLD-2      1 byte
 3 bytes     DATA-FLD-C          SRVR-FLD-3      1 byte
                                 SRVR-FLD-4      1 byte

 5 bytes     DATA-FLD-D          SRVR-FLD-5     10 bytes
 5 bytes     DATA-FLD-E

10 bytes     DATA-FLD-B
```

## Using PRESENT IF Clauses

The SCREEN COBOL programming language PRESENT IF clause lets you declare that certain fields in Message Section data structures are present only if a particular preceding field is nonzero (if numeric) or nonblank (if nonnumeric).

The following sample code illustrates two ways that you might use this capability. In the first example (which uses the Message Section template MSG-IN-FLAVOR1), the presence of the alias address, city, state, and ZIP fields all depend on the presence of the alias name field. In the second example (which uses the Message Section template

MSG-IN-FLAVOR2), the presence of the alias name, address, city, state, and ZIP fields is determined by bit-mask values contained earlier in the message.

The Working-Storage Section declarations for MSG-IN-FLAVOR1 are as follows:

```
WORKING-STORAGE SECTION.

   01 WS-MSG-IN.
      05 NAME         PIC X(20).
      05 ADDRESS      PIC X(20).
      05 CITY         PIC X(15).
      05 STATE        PIC X(3).
      05 ZIP          PIC 9(5).

      05 ALIAS-NAME         PIC X(20).
      05 ALIAS-ADDRESS      PIC X(20).
      05 ALIAS-CITY         PIC X(15).
      05 ALIAS-STATE        PIC X(3).
      05 ALIAS-ZIP          PIC 9(5).

   01 FIELD-STATUS.
      05 FS-NAME.
         10 FS-NAME-SHADOW       PIC 9(4) COMP.
         10 FS-NAME-ERROR        PIC 9(4) COMP.
      05 FS-ADDRESS.
         10 FS-ADDRESS-SHADOW    PIC 9(4) COMP.
         10 FS-ADDRESS-ERROR     PIC 9(4) COMP.
      05 FS-CITY.
         10 FS-CITY-SHADOW       PIC 9(4) COMP.
         10 FS-CITY-ERROR        PIC 9(4) COMP.
      05 FS-STATE.
         10 FS-STATE-SHADOW      PIC 9(4) COMP.
         10 FS-STATE-ERROR       PIC 9(4) COMP.
      05 FS-ZIP.
         10 FS-ZIP-SHADOW        PIC 9(4) COMP.
         10 FS-ZIP-ERROR         PIC 9(4) COMP.

   01 WS-MSG-IN-BIT-MASK.
      05 ALIAS-NAME-PRESENT     PIC 9.
      05 ALIAS-ADDRESS-PRESENT  PIC 9.
      05 ALIAS-CITY-PRESENT     PIC 9.
      05 ALIAS-STATE-PRESENT    PIC 9.
      05 ALIAS-ZIP-PRESENT      PIC 9.
```

The Message Section declarations for the MSG-IN-FLAVOR1 template are as follows:

```
MESSAGE SECTION.

   01 MSG-IN-FLAVOR1.
      05 MS-NAME            PIC X(20) TO NAME.
      05 MS-ADDRESS         PIC X(20) TO ADDRESS.
      05 MS-CITY            PIC X(15) TO CITY.
      05 MS-STATE           PIC X(3) TO STATE.
      05 MS-ZIP             PIC 9(5) TO ZIP.
      05 MS-ALIAS-NAME      PIC X(20) TO ALIAS-NAME.
      05 MS-ALIAS-ADDRESS   PIC X(20) TO ALIAS-ADDRESS
```

```
                                        PRESENT IF MS-ALIAS-
                                        NAME FIELD-STATUS IS
                                        FS-ADDRESS.
      05 MS-ALIAS-CITY       PIC X(15)  TO ALIAS-CITY
                                        PRESENT IF MS-ALIAS-
                                        NAME FIELD-STATUS IS
                                        FS-CITY.
      05 MS-ALIAS-STATE      PIC 9(3)   TO ALIAS-STATE
                                        PRESENT IF MS-ALIAS-
                                        NAME FIELD-STATUS IS
                                        FS-STATE.
      05 MS-ALIAS-ZIP        PIC 9(5)   TO ALIAS-ZIP
                                        PRESENT IF MS-ALIAS-
                                        NAME FIELD-STATUS IS
                                        FS-ZIP.
```

When the MSG-IN-FLAVOR1 template is used for receiving incoming messages, the fields NAME through ALIAS-NAME are always physically present. The overall content, blank or nonblank, of the field ALIAS-NAME determines whether the fields ALIAS-ADDRESS through ALIAS-ZIP are present. If ALIAS-NAME is entirely blank, the remaining ALIAS fields are not present. If ALIAS-NAME contains any nonblank characters, the remaining ALIAS fields are present.

The Message Section declarations for the MSG-IN-FLAVOR2 template are as follows:

```
MESSAGE SECTION.

01 MSG-IN-FLAVOR2.
   05 PRESENCE-MASK.
      10 ALIAS-NAME-PM      PIC 1 TO ALIAS-NAME-PRESENT.
      10 ALIAS-ADDRESS-PM   PIC 1 TO ALIAS-ADDRESS-PRESENT.
      10 ALIAS-CITY-PM      PIC 1 TO ALIAS-CITY-PRESENT.
      10 ALIAS-STATE-PM     PIC 1 TO ALIAS-STATE-PRESENT.
      10 ALIAS-ZIP-PM       PIC 1 TO ALIAS-ZIP-PRESENT.
      10 FILLER             PIC 1(3).
   05 MS-NAME            PIC X(20) TO NAME.
   05 MS-ADDRESS         PIC X(20) TO ADDRESS.
   05 MS-CITY            PIC X(15) TO CITY.
   05 MS-STATE           PIC X(3) TO STATE.
   05 MS-ZIP             PIC 9(5) TO ZIP.
   05 MS-ALIAS-NAME      PIC X(20) TO ALIAS-NAME.
                                 PRESENT IF ALIAS-NAME-PM
                                 FIELD-STATUS IS FS-NAME.
   05 MS-ALIAS-ADDRESS   PIC X(20) TO ALIAS-ADDRESS
                                 PRESENT IF ALIAS-ADDRESS-PM
                                 FIELD-STATUS IS FS-ADDRESS.
   05 MS-ALIAS-CITY      PIC X(15) TO ALIAS-CITY
                                 PRESENT IF ALIAS-CITY-PM
                                 FIELD-STATUS IS FS-CITY.

   05 MS-ALIAS-STATE     PIC 9(3)  TO ALIAS-STATE
                                 PRESENT IF ALIAS-STATE-PM
                                 FIELD-STATUS IS FS-STATE.
   05 MS-ALIAS-ZIP       PIC 9(5)  TO ALIAS-ZIP
                                 PRESENT IF ALIAS-ZIP-PM
                                 FIELD-STATUS IS FS-ZIP.
```

When the MSG-IN-FLAVOR2 template is used for receiving incoming messages, the fields PRESENCE-MASK through ZIP are always physically present. Each single-bit elementary item within PRESENCE-MASK determines whether one of the ALIAS fields (following ZIP) is present. For example, a value of 1 in the ALIAS-STATE-PM field indicates that the ALIAS-STATE field is present, a value of 0 in the ALIAS-NAME-PM field indicates that the ALIAS-NAME field is not present, and so forth.

## Error Processing and Debugging Techniques

Use the ON ERROR clause to detect errors that occur on input or output of the message to or from working storage. Use the FIELD STATUS clause to test for edit errors.

### ON ERROR Processing

If an error is detected in either phase of the SEND MESSAGE operation, the ON ERROR path is taken. The processing of the ON ERROR clause for the SEND MESSAGE statement is the same as that for the CALL and SEND statements. TERMINATION-STATUS values for the SEND MESSAGE statement are summarized in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

As part of the ON ERROR processing you need to check for a TERMINATION-STATUS of 5 or 15; 5 indicates input phase and 15 indicates output phase. If TERMINATION-STATUS is 5 or 15, you can then process the FIELD STATUS data item to detect an edit error.

### FIELD STATUS Processing

The FIELD STATUS clause identifies a working-storage data group or item that receives status information about the field during SEND MESSAGE operations.

The FIELD STATUS clause is used to obtain information on editing errors in fields where editing is specified.

The method of deciding which message template the FIELD STATUS data item belongs to differs for input and output messages.

#### Message Template Input Case

For input messages, you need to know which of the message templates of the YIELDS list you were processing when the errors occurred.

The relative position of the YIELDS list is returned in TERMINATION-SUBSTATUS.

The position is returned in TERMINATION-SUBSTATUS instead of TERMINATION-STATUS because this is the ON ERROR case. At this point TERMINATION-STATUS holds the error number. If this were the normal case, rather than ON ERROR, TERMINATION-STATUS would be used to define the relative position in the YIELDS list.

The following SEND MESSAGE statement shows the YIELDS clauses associated with the input messages:

```
SEND MESSAGE MSG-3-OUT-M-1
    REPLY CODE FIELD IS WS-MSG-4-IN-FROM-MSG-4-IN-CODE
       CODE 1 YIELDS MSG-3-IN
       CODE 2 YIELDS MSG-4-IN
       CODE 3 YIELDS MSG-3-IN
       CODE 4 YIELDS MSG-4-IN
    ON ERROR PERFORM IDS-SERVER-SEND-ERROR.
```

The following PERFORM statement shows how to use TERMINATION- SUBSTATUS to decide which input message template to process:

```
PERFORM ONE OF
   PROC-MSG-3-IN-EDIT-STATUS
   PROC-MSG-4-IN-EDIT-STATUS
   PROC-MSG-3-IN-EDIT-STATUS
   PROC-MSG-4-IN-EDIT-STATUS
       DEPENDING ON TERMINATION-SUBSTATUS.
```

The following table shows the input message that correlates with TERMINATION-SUBSTATUS:

| Message Template | TERMINATION-SUBSTATUS |
|---|---|
| YIELDS MSG-3-IN | 1 |
| YIELDS MSG-4-IN | 2 |
| YIELDS MSG-3-IN | 3 |
| YIELDS MSG-4-IN | 4 |

## Message Template Output Case

Because only one message template can be specified on output, you know which set of FIELD STATUS data items to test.

## Recommended Format of FIELD STATUS Item

The recommended format of the FIELD STATUS item is:

```
02   FIELD-STATUS-AREA.
     03   SHADOW-INFO          PIC 9(4) COMP.
     03   FIELD-ERROR          PIC 9(4) COMP.
```

This format, although not required by the compiler, allows for easy processing of the FIELD STATUS information. The SHADOW information is the same as that within the Screen Section. The FIELD-ERROR area allows the TCP to report specific errors that relate to this individual Message Section field.

# 7 Processing Unsolicited Messages

The unsolicited-message processing (UMP) feature of Compaq *NonStop*™ Pathway/iTS makes it possible for SCREEN COBOL requesters to accept and reply to unsolicited messages sent to them by processes that are outside the PATHMON environment. These external processes can reside anywhere within a Compaq *Expand* network.

The following clauses, statements, and registers in the SCREEN COBOL language support the processing of unsolicited messages:

- An escape condition for the ACCEPT and SEND MESSAGE statements:  ESCAPE ON UNSOLICITED MESSAGE
- The statements RECEIVE UNSOLICITED MESSAGE and REPLY TO UNSOLICITED MESSAGE
- The read-only special registers PW-UNSOLICITED-MESSAGE-QUEUED, PW-TCP-SYSTEM-NAME, PW-TCP-PROCESS-NAME, and PW-USE-NEW-CURSOR
- The read-write special register PW-QUEUE-FKEY-UMP

This section presents information about the following topics:

- Detecting the arrival of unsolicited messages
- Accepting unsolicited messages
- Replying to unsolicited messages
- The SCREEN COBOL special registers used with unsolicited messages
- TERMINATION-STATUS values and Pathway/iTS error codes related to unsolicited messages
- UMP programming examples
- Sending unsolicited messages to SCREEN COBOL requesters
- Unsolicited-message layout, reply layout, and error codes
- Configuration parameters related to the UMP feature

The *Compaq NonStop*™ *Pathway/iTS SCREEN COBOL Reference Manual* contains additional information about the unsolicited-message processing feature.

## Detecting the Arrival of Unsolicited Messages

Each requester program has its own unsolicited-message queue. In addition, each requester program has its own copy of the PW-UNSOLICITED-MESSAGE-QUEUED special register that is global to any program units called by that requester.

When the TCP receives an unsolicited message addressed to one of its requesters, it places the message in the appropriate queue and sets the value of the associated PW-UNSOLICITED-MESSAGE- QUEUED register to YES.

A requester can detect the arrival of an unsolicited message in any of the following ways:

- By testing the content of its PW-UNSOLICITED-MESSAGE-QUEUED special register

  YES signifies that one or more messages have arrived and are waiting to be processed; NO signifies that the queue is empty.

- By performing a RECEIVE UNSOLICITED MESSAGE statement as a waited input operation
- By including an ESCAPE ON UNSOLICITED MESSAGE clause in ACCEPT or SEND MESSAGE statements

## Accepting Unsolicited Messages

Requesters obtain the text of an unsolicited message by performing a RECEIVE UNSOLICITED MESSAGE statement. Messages can move either directly into working-storage or move indirectly there from a Message Section templates.

## Replying to Unsolicited Messages

When a requester has constructed an appropriate response message, it replies to an unsolicited message by performing a REPLY TO UNSOLICITED MESSAGE statement. After replying to the message, the requester is then free to accept and process another unsolicited message.

Having received one message (by performing a RECEIVE UNSOLICITED MESSAGE statement), the requester cannot perform another RECEIVE UNSOLICITED MESSAGE statement until it has replied to the first message (by using a REPLY TO UNSOLICITED MESSAGE statement).

## The PW-TCP-SYSTEM-NAME and PW-TCP-PROCESS-NAME Special Registers

The read-only special registers, PW-TCP-SYSTEM-NAME and PW-TCP-PROCESS-NAME, contain the system name and *Guardian* process name of the requester's TCP.

They are intended for use by a requester, in conjunction with the LOGICAL-TERMINAL-NAME special register, when the requester is identifying itself to a process that is a member of an active Pathway server class.

A requester can identify itself to such a process by formatting a message containing the TCP system name and process name and the requester's name and passing the message to the server process through a SEND statement. The server process can then use those values in the UMP header of unsolicited messages to communicate with the requester.

Programs attempting to modify the content of either of these special registers are flagged at compilation with the following message:

```
** ERROR 454 **  READ-ONLY SPECIAL REGISTER; MAY NOT BE ALTERED
```

These registers have the following implicit declarations. (The VALUE clauses are for illustrative purposes only.)

```
PW-TCP-SYSTEM-NAME        PIC X(8) VALUE "\STLOUIS".
PW-TCP-PROCESS-NAME       PIC X(6) VALUE "$STCP".
```

## The PW-USE-NEW-CURSOR Special Register

For all terminals supported by Pathway/iTS, except the 6510, information is displayed on the screen without altering the location of the visible cursor. Terminals other than the 6510 have a buffer pointer, independent of the visible cursor, whose value determines where information is displayed on the screen; the visible cursor position is altered separately. Thus, when PW-USE-NEW-CURSOR is set to NO, the visible cursor can be left unchanged.

The 6510 visible cursor functions as both a user next-character marker and a buffer pointer; consequently, there is no way to avoid altering the cursor position when writing to the screen on a 6510. Because of this, the PW-USE-NEW-CURSOR special register has no effect on 6510 terminal operation. The TCP performs all I/O operations on a 6510 as if PW-USE-NEW-CURSOR had the YES value

You can use basically the same source code for program units of different terminal types, changing only the terminal type in the Environment Division

For example, the following code is acceptable to either 6510 or other terminals:

For example, the following code is acceptable to any supported terminal:

```
SET NEW-CURSOR AT a-field.
MOVE "NO" TO PW-USE-NEW-CURSOR.
ACCEPT field names.
```

The ACCEPT verb in this example acts differently on a 6510 terminal than it does on other terminals. On a terminal that is not a 6510, the visible cursor position is unchanged because of the NO value in PW-USE-NEW-CURSOR; on a 6510, however, the PW-USE-NEW-CURSOR special register has no effect and the visible cursor moves to the screen location designated by *`a-field`*.

## Unsolicited-Message TERMINATION-STATUS Values

SCREEN COBOL requesters that process unsolicited messages should test for the following TERMINATION-STATUS error codes:

13 A timeout or ESCAPE ON UNSOLICITED MESSAGE caused the TCP to issue a CONTROL 26 call to the external front-end process. An I/O error occurred in conjunction with the CONTROL 26 call; however, TERMINATION-SUBSTATUS specifies the file system error code that was returned with the CONTROL 26 completion. This value is returned only for SEND MESSAGE statements.

14 A timeout or ESCAPE ON UNSOLICITED MESSAGE caused the TCP to issue a CONTROL 26 call to the external front-end process. The front-end process did not respond within the maximum allowable amount of time, however. This value is returned only for SEND MESSAGE statements.

16 A RECEIVE UNSOLICITED MESSAGE statement was issued when a REPLY TO UNSOLICITED MESSAGE was required. When processing unsolicited messages, the requester must always reply to each received message before issuing any subsequent RECEIVE UNSOLICITED MESSAGE statements.

17 A REPLY TO UNSOLICITED MESSAGE statement was issued when no corresponding RECEIVE UNSOLICITED MESSAGE statement had been previously executed.

18 A RECEIVE UNSOLICITED MESSAGE statement was issued when the PATHCOM SET TERM parameter MAXINPUTMSGS was set, or defaulted, to 0. That PATHCOM parameter specifies the maximum number of unsolicited messages that the TCP can have queued at any given time for the particular requester.

## Pathway/iTS Error Codes

The following Pathway/iTS error codes can appear in the PATHMON log as the result of unsolicited-message processing.

```
3125 - MULTIPLE UNSOLICITED MESSAGES REJECTED DUE TO TERM
STOP/SUSPEND
```

Unsolicited messages were queued for a requester when the requester was stopped, suspended, or aborted, either programmatically or by operator command. The INFO field specifies the number of unsolicited messages that were rejected.

```
3176 - ATTEMPT TO RECEIVE UNSOLICITED MESSAGE WITH ONE NOT
YET REPLIED TO
```

An attempt was made to receive an unsolicited message when a previously received one had not yet been replied to. After accepting an unsolicited message, a requester must always perform a REPLY TO UNSOLICITED MESSAGE statement.

```
3177 - NO UNSOLICITED MESSAGE TO REPLY TO
```

An attempt was made to reply to an unsolicited message when none was pending.

```
3178 - ATTEMPT TO RECEIVE UNSOLICITED MESSAGE WHEN TERM
MAXINPUTMSGS = 0
```

An attempt was made to RECEIVE UNSOLICITED MESSAGE, but the requester is not configured to accept unsolicited messages.

```
3240 - VALUE FOR MAXINPUTMSGS TOO LARGE—MAX IS 2045
```

The value specified for the PATHCOM SET TCP MAXINPUTMSGS parameter exceeded 2045. Because PATHCOM disallows a value greater than 2045, the appearance of this message in the log reflects an internal Pathway/iTS error.

```
3241 - UNSOLICITED MESSAGE REJECTED BY TCP
```

The TCP rejected an unsolicited message for the reason specified by the INFO field. This message can be generated on behalf of either an individual requester or the entire TCP, as appropriate.

```
3242 - MULTIPLE UNSOLICITED MESSAGES REJECTED
```

The TCP rejected one or more unsolicited messages and replied to their sender without delivering them to their target requester. This message is seen only if unsolicited messages are arriving and being rejected with sufficient frequency that the TCP cannot log individual error messages for each rejected unsolicited message.

For cause, effect, and recovery information for these messages and other TCP messages, refer to the *Compaq NonStop™ Pathway/iTS System Management Manual*.

## UMP Programming Examples

The subsections that follow present a series of annotated programming examples illustrating various approaches to unsolicited-message processing within SCREEN COBOL requesters.

### Polling the PW-UNSOLICITED-MESSAGE-QUEUED Register

One method of detecting the arrival of unsolicited messages is to branch periodically to a paragraph that acts upon the current state of the PW-UNSOLICITED-MESSAGE-QUEUED register. If the register contains the value YES, the paragraph processes unsolicited messages from the requester's queue. When the register finally contains the value NO, control returns to the main processing stream.

```
       .
       .
       .
 PERFORM CHECK-FOR-UNSOLICITED-MESSAGES.
       .
       .
       .

 CHECK-FOR-UNSOLICITED-MESSAGES.
   IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
      PERFORM process-unsolicited-message
         UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

 PROCESS-UNSOLICITED-MESSAGE.
   RECEIVE UNSOLICITED MESSAGE
      YIELDS ws-unsolicited-message
      ON ERROR GO TO analyze-error.
*     Do something with ws-unsolicited-message
*     and format a reply message.
   REPLY TO UNSOLICITED MESSAGE reply-message.
```

## Using Waited RECEIVE UNSOLICITED Statements

Another way of detecting the arrival of unsolicited messages is to branch to a paragraph that issues a RECEIVE UNSOLICITED MESSAGE statement; the statement may or may not include a TIMEOUT clause. If there are no unsolicited messages currently queued, the RECEIVE UNSOLICITED statement acts as a waited input request comparable to ACCEPT. If the statement does not include a TIMEOUT clause, it waits indefinitely for a message. If the statement includes a TIMEOUT clause and the specified timeout period elapses without the receipt of an unsolicited message, control passes to the ON ERROR code, if present. If there is no ON ERROR paragraph, the requester suspends.

This approach has two apparent drawbacks when compared with the preceding example: it processes only a single message, regardless of how many might currently be on the queue, and it causes the requester to halt either indefinitely or up to the specified timeout period if the queue is empty and no unsolicited message arrives. This is a valid approach, however, that might be appropriate in certain application environments.

```
        .
        .
        .
 PERFORM CHECK-FOR-UNSOLICITED-MESSAGE.
        .
        .
 CHECK-FOR-UNSOLICITED-MESSAGE.
   RECEIVE UNSOLICITED MESSAGE
      YIELDS ws-unsolicited-message
      TIMEOUT one-minute
      ON ERROR GO TO analyze-error.
*     Do something with ws-unsolicited-message
*     and format a reply message.
   REPLY TO UNSOLICITED MESSAGE reply-message.

 ANALYZE-ERROR.
   IF TERMINATION-STATUS    = 1 and
   TERMINATION-SUBSTATUS = 40
*     The statement timed out, which in this
*     case is not really an error condition;
*     at the very least you must specify some
*     kind of declarative clause here, such as
*     MOVE TIMED-OUT TO LATEST-COMPLETION, even
*     if it amounts to a no op.
   ELSE
*     Respond appropriately to other types
*     of errors.
```

If a message arrives before the RECEIVE UNSOLICITED statement times out, the message is moved to *ws-unsolicited-message* in working storage and control proceeds with the statements immediately following the RECEIVE UNSOLICITED statement.

## Using ESCAPE ON UNSOLICITED MESSAGE Clauses

One of the most common methods of detecting and reacting to unsolicited messages is the interrupt technique, whereby you include ESCAPE ON UNSOLICITED MESSAGE clauses in ACCEPT or SEND MESSAGE statements.

Example 7-1 illustrates the use of such clauses with an ACCEPT statement; Example 7-2 does the same with a SEND MESSAGE statement.

**Example 7-1. UMP and the ACCEPT Statement**

```
 GET-OPERATOR-INPUT.
    ACCEPT my-screen
       UNTIL f1-key
             sf16-key
       ESCAPE ON
             TIMEOUT one-hour
             UNSOLICITED MESSAGE.
       PERFORM ONE OF f1-key-action
             sf16-key-action
             timed-out
             unsolicited-message-arrival
       DEPENDING ON TERMINATION-STATUS.
       GO TO get-operator-input.

 F1-KEY-ACTION.
* TERMINATION-STATUS = 1; respond to function-key 1 condition.

 SF16-KEY-ACTION.
* TERMINATION-STATUS = 2; respond to shifted function-key 16
* condition.

 TIMED-OUT.
* TERMINATION-STATUS = 3; respond to time-out condition.

 UNSOLICITED-MESSAGE-ARRIVAL.
* TERMINATION-STATUS = 4; receive, process, and reply to the
* unsolicited message.
    IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
        PERFORM process-unsolicited-message
           UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

 PROCESS-UNSOLICITED-MESSAGE.
    RECEIVE UNSOLICITED MESSAGE
       YIELDS unsolicited-latest-prices
       ON ERROR GO TO analyze-error.
* Do something with unsolicited-latest-prices and format a
* reply message.
    REPLY TO UNSOLICITED MESSAGE reply-message.
    MOVE "NO" TO PW-USE-NEW-CURSOR.
*   Preserves the old cursor position.
```

**Example 7-2. UMP and the SEND MESSAGE Statement**

```
 SEND-MESSAGE-AND-RECEIVE-REPLY.
    SEND MESSAGE request-message
       REPLY CODE "AA" YIELDS aa-reply
       ESCAPE ON UNSOLICITED MESSAGE
           TIMEOUT five-minutes
           ON ERROR GO TO analyze-error.
    PERFORM ONE OF aa-reply
                   unsolicited-message-arrival
    DEPENDING ON TERMINATION-STATUS.
    GO TO send-message-and-receive-reply.

 AA-REPLY.
*   TERMINATION-STATUS = 1.  This is a normal (expected)
completion.
*   Do whatever is appropriate for an AA-type reply and then
resume
*   regular processing.

 UNSOLICITED-MESSAGE-ARRIVAL.
*   TERMINATION-STATUS = 2.  The statement was interrupted by
the
*   arrival of an unsolicited message.
*
*   If CONTROL 26 is enabled, at this point you should check
*   TERMINATION-SUBSTATUS for the values 187, 188, or 189.
*
*   If CONTROL 26 is not enabled, you should check
*   TERMINATION-SUBSTATUS for the value 0.

    IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
       PERFORM process-unsolicited-message
          UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

 PROCESS-UNSOLICITED-MESSAGE.
    RECEIVE UNSOLICITED MESSAGE
       YIELDS unsolicited-statistics-request.
*   Gather the requested statistics and format a reply message.
    REPLY TO UNSOLICITED MESSAGE requested-statistics.
```

## ESCAPE ON UNSOLICITED MESSAGE Design Considerations

When writing an application that uses ESCAPE ON UNSOLICITED MESSAGE clauses, you have several design considerations.

When using the interrupt method, if there is a chance that a UMP message might get canceled, it is recommended that a check to the PW-UNSOLICITED-MESSAGE-QUEUE be made before executing the RECEIVE UNSOLICITED MESSAGE verb. This prevents the SCREEN COBOL program from indefinitely waiting on the RECEIVE UNSOLICITED MESSAGE verb when the UMP message that caused the interrupt is canceled before the RECEIVE UNSOLICITED MESSAGE verb is executed.

## Applications for Intelligent Terminals

If your application is for intelligent terminals, consider the following:

- You can use the interrupt method of responding to unsolicited messages (ESCAPE ON UNSOLICITED MESSAGE) with SEND MESSAGE statements in IDS requesters that communicate with front-end processes whether the front-end processes support the use of CONTROL 26. If the front-end process supports the use of CONTROL 26, an ESCAPE ON UNSOLICITED MESSAGE can be performed with no loss of data. If the front-end process does not support the use of CONTROL 26, then the TCP uses only CANCEL calls (no CONTROL 26 calls) to terminate the underlying read operation. In the latter case, the amount of data that could be lost is application-dependent or device-dependent.
- If a UMP message is already queued when a SEND MESSAGE with an ESCAPE ON UNSOLICITED MESSAGE clause is executed, the TCP still issues the SEND MESSAGE call. To process the ESCAPE ON UNSOLICITED MESSAGE clause, however, the TCP immediately cancels the SEND MESSAGE WRITEREAD or issues a CONTROL 26 call against the WRITEREAD. To prevent unnecessary processing on a SEND MESSAGE, poll the PW-UNSOLICITED-MESSAGE-QUEUED register for a YES value to detect the arrival of UMP messages prior to issuing the SEND MESSAGE statement.

## Applications for Block-Mode Terminals

If your application is for block-mode terminals, consider the following:

- You can also use the interrupt technique with ACCEPT statements in standard requesters that control block-mode terminals supported by Pathway/iTS. In that context, however, the TCP uses only CANCEL calls (no CONTROL 26 calls) to terminate the underlying read operation; it does so with a minimal loss of data. (At most, the loss is a single function-key stroke.)
- See Internal Function-Key Queuing on page 3-13 for implications of the function-key queuing capability of 6530 terminals and ESCAPE ON UNSOLICITED MESSAGE clauses within ACCEPT verbs.

## Applications for Conversational Terminals

If your application is for conversational terminals, the interrupt technique of processing must be used judiciously with conversational-mode requesters. This is because the arrival of an unsolicited message causes all data entered by the terminal user for all fields being accepted by the interrupted ACCEPT verb to be discarded.

## Message Processing Requiring No Terminal Interaction

Assume that an ACCEPT statement issued by a standard requester controlling a supported block-mode terminal is interrupted by the arrival of an unsolicited message. Also assume that the message requires no terminal input-output. For example, the message asks the requester to return statistics maintained in its working storage.

The applicable UMP-code constructs for handling this situation are as follows:

```
   PROCEDURE DIVISION.
     MAIN-LOOP.
        DISPLAY BASE name-screen.

     MAIN-INPUT.
        ACCEPT name-screen
           UNTIL     f1
           ESCAPE ON
                     TIMEOUT one-hour
                     UNSOLICITED MESSAGE.
     PERFORM ONE OF
            normal-f1-completion
            operation-timed-out
            unsolicited-message-arrival
     DEPENDING ON termination-status
     GO TO main-input.


     UNSOLICITED-MESSAGE-ARRIVAL.
       IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
         PERFORM process-unsolicited-message
           UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

     PROCESS-UNSOLICITED-MESSAGE.
        RECEIVE UNSOLICITED MESSAGE YIELDS receive-msg.
*    Build the reply.
        REPLY TO UNSOLICITED MESSAGE WITH reply-msg.
*    Prevent the ACCEPT statement from moving the cursor.
        MOVE "NO" TO PW-USE-NEW-CURSOR.
```

## Message Processing Requiring Only Terminal Output

The next example illustrates the case where the arrival of an unsolicited message requires the displaying of information on the terminal whose I/O operation was interrupted. Assume that the unsolicited message contains latest price information that must be displayed to the terminal.

Following message processing, the program reissues the interrupted ACCEPT operation.

```
  PROCEDURE DIVISION.
    MAIN-LOOP.
      DISPLAY BASE normal-screen.

*   The following ACCEPT operation is reissued when
*   unsolicited-message processing is complete. At that time,
*   any data typed on the screen is preserved.  At worst,
*   the operator will have to reissue a single function-key
*   stroke.

    MAIN-INPUT.
      ACCEPT normal-screen UNTIL f5
        ESCAPE ON UNSOLICITED MESSAGE.
      PERFORM ONE OF
        normal-f5-completion
        unsolicited-message-arrival
      DEPENDING ON termination-status
      GO TO main-input.


    UNSOLICITED-MESSAGE-ARRIVAL.
     IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
       PERFORM process-unsolicited-message
         UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

    PROCESS-UNSOLICITED-MESSAGE
      RECEIVE UNSOLICITED MESSAGE YIELDS rcv-msg.
*   Reply immediately to sender because no actual information
*   is contained in the reply.
*   Shorten the wait for $RECEIVE response.
      REPLY TO UNSOLICITED MESSAGE WITH reply-msg.
*   Set up screen to request system status.
      MOVE CORRESPONDING price-info OF rcv-msg
      TO latest-prices.
*   The latest information displays on the screen; operator
*   continues screen interaction.
      DISPLAY latest-price-overlay.
*   Prevent the ACCEPT statement from moving the cursor.
      MOVE "NO " TO PW-USE-NEW-CURSOR.
```

## Message Processing Requiring Both Input and Output

The next example illustrates a case when the arrival of an unsolicited message requires that information be displayed on the terminal where I/O was interrupted and that the terminal operator enter a reply.

Following message processing, the program reissues the interrupted ACCEPT operation. Because the unsolicited message required operator input, the previous cursor position is lost and the operator has to reposition the cursor manually at the appropriate screen location.

For brevity, error handling has been omitted from the example.

```
    PROCEDURE DIVISION.
      MAIN-LOOP.
        DISPLAY BASE normal-screen.

*     The following ACCEPT operation is reissued when
*     unsolicited-message processing is complete.
*     At that time, any data typed on the screen is preserved.
*     The operator might have to reposition the cursor or
*     reissue a function-key stroke.

      MAIN-INPUT.
        ACCEPT normal-screen-data UNTIL f5
          ESCAPE ON UNSOLICITED MESSAGE.
        PERFORM ONE OF
          normal-processing
          unsolicited-message-arrival
        DEPENDING ON termination-status
        GO TO main-input.


      UNSOLICITED-MESSAGE-ARRIVAL.
       IF PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "YES"
         PERFORM process-unsolicited-message
           UNTIL PW-UNSOLICITED-MESSAGE-QUEUED IS EQUAL TO "NO".

      PROCESS-UNSOLICITED-MESSAGE.
*     Accept, process, and reply to unsolicited message.
        RECEIVE UNSOLICITED MESSAGE YIELDS rcv-msg.
*     Set up window for operator response.
        DISPLAY urgent-op-overlay.
*     Accept operator input to urgent overlay.
        ACCEPT  ws-urgent-op-reply UNTIL f1.
        MOVE ws-urgent-op-reply TO reply-msg-op-reply.
*     Reply to sender with operator's response.
        REPLY TO UNSOLICITED MESSAGE WITH reply-msg.
```

## Sending Unsolicited Messages to SCREEN COBOL Requesters

Guardian processes send unsolicited messages to SCREEN COBOL requesters by using the appropriate TCP. An application that sends multiple unsolicited messages to a TCP should open the TCP only once and close it at the end of processing.

Such messages consist of two parts:

- A UMP header that gets interpreted by the receiving TCP
- The body of the message that gets passed to the SCREEN COBOL requester program

The length of the receiving buffer within the requester program is established by the application designer. The maximum length of a message can be configured through PATHCOM. The length is that of the body of the maximum message, exclusive of the UMP header.

To send a message to a SCREEN COBOL requester program, the sending process must build a message whose header contains the Pathway/iTS terminal name; this message is then sent to the TCP controlling the terminal.

There are two methods for obtaining the information necessary to complete the header of an unsolicited message to a SCREEN COBOL requester.

- The Subsystem Programmatic Interface (SPI), a token-oriented programmatic interface provided by Compaq, provides the means whereby Guardian processes outside of a PATHMON environment can communicate with the PATHMON process to obtain the information necessary to complete the UMP header of a message intended for a known terminal.

  Given the Pathway/iTS terminal name, the management programming interface gets the TCP system name and the Guardian process name. The procedure for doing this for the terminal TERM-X is as follows:

  1. Issue an INFO TERM TERM-X request. The response from PATHMON contains the TCP name, such as TCP-X.
  2. Issue a STATUS TCP TCP-X request. The response contains both the system name and the Guardian process name of TCP-X.

  Detailed information about the management programming interface to the Pathway subsystem is presented in the *NonStop™ TS/MP Management Programming Manual* and the *Compaq NonStop™ Pathway/iTS Management Programming Manual*.

- Assuming the requester has access (direct or indirect) to the process that will be sending the unsolicited messages, the requester can initially send its information—its Pathway/iTS terminal name, its TCP system name, and its TCP process name—from the special registers LOGICAL-TERMINAL-NAME, PW-TCP-SYSTEM-NAME, and PW-TCP-PROCESS-NAME by using the SEND (to server) verb.

For requesters that are started by a PATHCOM RUN PROGRAM command, only the second method is appropriate because the logical name of the terminal is determined dynamically by the PATHMON process.

The reply returned to the sender of an unsolicited message includes error information when either of the following is true:

- The supplied terminal name is not currently active. (The terminal is either stopped or suspended.)
- The unsolicited-message queue of the target requester is full.

## Unsolicited-Message Layout, Reply Layout, and Error Codes

The subsections that follow describe the format of unsolicited messages as sent by an external process (outside the PATHMON environment) to the TCP, the format of replies sent by the TCP to the external process, and the various reply codes that the TCP can include in the reply record.

### Unsolicited-Message Layout

Figure 7-1 illustrates, in a COBOL-like representation, the format of an unsolicited message as viewed by both the TCP that receives it and the external process (outside the PATHMON environment) that sends it.

Upon receipt of a message, the TCP strips off the header and stores only the text portion in the target requester's queue.

**Figure 7-1. UMP Message Format**

```
01 UMP-MSG.
 02 TCP-UMP-HDR.
    03 PROTOCOL-ID        PIC 9(4) COMP VALUE 42.
    03 MSG-ID             PIC 9(4) COMP VALUE 1.
    03 MSG-VERSION        PIC 9(4) COMP VALUE 1.
    03 MSG-HEADER-LEN     PIC 9(4) COMP VALUE 40.
    03 DEST-NODE          PIC X(8).
    03 DEST-TCP-PROC      PIC X(6).
    03 LOGICAL-TERM-NAM   PIC X(15).
    03 FILLER             PIC X.
    03 MSG-SEQUENCE-NUM   PIC 9(4) COMP.
 02 SCOBOL-MSG.
    03 MSG-TEXT           PIC X(number-of-characters).
```

The text that follows briefly describes each field.

```
01 UMP-MSG.
```

Sent to the appropriate TCP by a Guardian process by making a WRITEREAD call with a read-count large enough to contain the reply header and reply text. The target SCREEN COBOL requester accepts the text portion by using a RECEIVE UNSOLICITED MESSAGE statement.

```
   02 TCP-UMP-HDR.
```

The header is seen only by the TCP (not by the SCREEN COBOL requester).

```
      03 PROTOCOL-ID         PIC 9(4) COMP VALUE 42.
```

Unsolicited message protocol indicator. Must be 42.

```
      03 MSG-ID              PIC 9(4) COMP VALUE 1.
```

Message identification number. Must be 1.

```
      03 MSG-VERSION         PIC 9(4) COMP VALUE 1.
```

Message format version number. Must be 1.

```
      03 MSG-HEADER-LEN      PIC 9(4) COMP VALUE 40.
```

Number of bytes in TCP-UMP-HDR. Must be 40.

```
      03 DEST-NODE           PIC X(8).
```

The symbolic name of the destination node of this message (such as \CORPHQ), left justified and blank filled.

```
      03 DEST-TCP-PROC       PIC X(6).
```

The Guardian process name of the destination TCP of this message (such as $BTCP), left justified and blank filled.

```
      03 LOGICAL-TERM-NAM    PIC X(15).
```

The name of the destination terminal as defined by the PATHCOM ADD TERM command.

```
      03 FILLER              PIC X.
```

Filler ensuring that the next field starts on word boundary.

```
      03 MSG-SEQUENCE-NUM    PIC 9(4) COMP.
```

A 16-bit binary number that uniquely identifies this particular message within the context of the program that generated the message.

```
02 SCOBOL-MSG.
```

CODE FIELD in the RECEIVE UNSOLICITED MESSAGE verb is relative to this point.

```
03 MSG-TEXT            PIC X(number-of-characters).
```

Text being sent to the SCREEN COBOL requester (can be null).

If the fields PROTOCOL-ID, DEST-NODE, and DEST-TCP-PROC are not filled out in TCP-UMP-HDR, the TCP returns a file system security-violation error (48), with no data, in response to the unsolicited message. If the unsolicited message has fewer bytes than the 40-byte TCP-UMP-HDR header, the message is also rejected with an error 48.

## Unsolicited-Message Reply Layout

Figure 7-2 illustrates, in a COBOL-like representation, the format of a reply to an unsolicited message as viewed by both the TCP that sends it and the external process (outside the PATHMON environment) that receives it.

On execution of a REPLY TO UNSOLICITED MESSAGE statement, the TCP adds the appropriate UMP header to the reply text supplied by the requester.

**Figure 7-2. UMP Reply Format**

```
01 UMP-REPLY.
 02 TCP-UMP-HDR.
    03 MSG-ID               PIC 9(4) COMP.
    03 REPLY-ID             PIC 9(4) COMP VALUE 1.
    03 REPLY-VERSION        PIC 9(4) COMP.
    03 REPLY-HEADER-LEN     PIC 9(4) COMP.
    03 ERROR-CODE           PIC 9(4) COMP.
    03 INFO1                PIC 9(4) COMP.
    03 INFO2                PIC 9(4) COMP.
    03 REPLY-SEQ-NUM        PIC 9(4) COMP.
 02 SCOBOL-REPLY.
    03 REPLY-TEXT           PIC X(number-of-characters).
```

The text that follows briefly describes each field.

```
01 UMP-REPLY.
```

Built by the TCP (with reply text supplied by the requester) and sent to the Guardian process that sent the unsolicited message. The requester passes the reply text to the TCP by using a REPLY TO UNSOLICITED MESSAGE statement.

```
02 TCP-UMP-HDR.
```

```
03 MSG-ID             PIC 9(4) COMP.
```

Copied from the MSG-ID field of the unsolicited message.

```
03 REPLY-ID          PIC 9(4) COMP VALUE 1.
```

Reply identification number. Must be 1.

```
03 REPLY-VERSION     PIC 9(4) COMP.
```

Reply format version number. This number is less than or equal to the value of the MSG-VERSION field in the corresponding unsolicited message. The TCP converts this field to:

```
$MIN ( MSG-VERSION, version the TCP recognizes )
```

This line lets the message sender, when it receives the reply back from the TCP, know what version level of the UMP protocol the TCP is using, thereby allowing message senders to interact more effectively with TCPs that are using older versions of the UMP protocol.

```
03 REPLY-HEADER-LEN   PIC 9(4) COMP.
```

The number of bytes in the header of the reply message (currently 16).

```
03 ERROR-CODE         PIC 9(4) COMP.
```

Zero or an error code returned by the TCP.

```
03 INFO1              PIC 9(4) COMP.
```

See error code 10 in "Unsolicited Message Error Codes" in this section.

```
03 INFO2              PIC 9(4) COMP.
```

Reserved for future use.

```
03 REPLY-SEQ-NUM      PIC 9(4) COMP.
```

The sequence number (MSG-SEQ-NUM) from the message that is being replied to. The COMP VALUE is a 16-bit binary number.

```
02 SCOBOL-REPLY.
```

The requester furnishes this data by using a REPLY TO UNSOLICITED MESSAGE statement.

```
03 REPLY-TEXT         PIC X(number-of-characters).
```

The text being returned.

## Unsolicited-Message Error Codes

Table 7-1 lists the error codes returned by the TCP to the sender of an unsolicited message. The codes are passed through the ERROR-CODE field of the UMP-REPLY record.

**Table 7-1. Unsolicited-Message Error Codes**

| Error Code | Meaning |
|---|---|
| 0 | No errors. |
| 1 | Guardian procedure error occurred. |
| 2 | Target terminal suspended. |
| 3 | Target terminal queue full. (The TERM MAXINPUTMSGS configuration value has been exceeded.) |
| 4 | Target TCP queue area full |
| 5 | Target queuing not enabled. (The TCP MAXINPUTMSGS configuration value is set to 0.) |
| 6 | Length field in message exceeds the configured maximum for the TCP. |
| 7 | Unrecognizable request code (refers to MSG-ID and MSG-VERSION fields). |
| 8 | Named terminal not found. |
| 9 | MAXINPUTMSGS exceeded. |
| 10 | RECEIVE UNSOLICITED MESSAGE error occurred—INFO1 gives the reason (the value obtained from the TERMINATION-STATUS register). |
| 11 | Number of bytes in reply message sent by SCREEN COBOL requester was more than had been asked for by the sender of the unsolicited message. This is merely a warning and should occur only during the application-debugging phase. |

## UMP Configuration Parameters

The following PATHCOM configuration parameters support the processing of unsolicited messages:

- SET TERM MAXINPUTMSGS *`number`*

  Specifies the total number of unsolicited messages that can be queued by the TCP for a particular requester at any one time. When that number is reached, the TCP rejects all subsequent unsolicited messages addressed to that requester until the requester replies to one of its currently queued messages. This parameter is particularly useful for ensuring that no single requester can consume a TCP's entire space for unsolicited-message processing.

- SET TCP MAXINPUTMSGS *`number`*

  Specifies the total number of unsolicited messages that a particular TCP can have queued at any one time for all its requesters. When that number is reached, the TCP rejects all subsequent unsolicited messages until one of its requesters replies to a currently queued message.

- SET TCP MAXINPUTMSGLEN *`number`*

  Specifies the maximum-size unsolicited message (in bytes) that the TCP will accept. Messages that are longer than the maximum length are rejected by the TCP with an appropriate error code. The specified length does not include the standard UMP message header (currently 40 bytes).

---

**Note.** The value that you specify for the SET TCP TERMPOOL parameter must be large enough to accommodate the largest UMP message (or the largest terminal I/O operation).

---

The *Compaq NonStop™ Pathway/iTS System Management Manual* gives details about UMP configuration parameters.

# 8 Processing Double-Byte Character Sets

As a Pathway application programmer, you can develop SCREEN COBOL program units that use double-byte character sets for selected devices.

The TCP supports these devices with the aid of translation routines in the TCP user library. The TCP uses the Shift-JIS format as its internal representation of double-byte characters. When a message is output to a device, the translation routines convert Shift JIS to double-byte characters suitable for display on the terminal or printer, inserting shift-out/shift-in (SO/SI) characters if necessary. When a message is input to a device, the process is reversed; the translation routines strip the SO/SI characters.

This section presents information about the following topics:

- Device types on which Compaq *NonStop*™ Pathway/iTS supports double-byte character sets
- How the character set is determined
- Data-item considerations
- Developing SCREEN COBOL programs for double-byte character sets
- Example of Working-Storage Section and Screen Section for double-byte character sets

## Device Types

The TCP supports double-byte character sets on the following terminals or personal computers running terminal emulators. To use SNA 3270 devices with double-byte character sets, you must have the C11 version or a later version of the SNAX/XF product installed.

- Selected Japanese versions of 6530 terminal emulators
- Selected Fujitsu 3270 terminals, terminal emulators, and printers
- Selected IBM 3270 terminals, terminal emulators, and printers

  To run SCREEN COBOL program units that use Kanji characters on IBM 3270 devices, the devices must support Start Field Extended (SFE) orders. To run applications that use Katakana characters, IBM 3270 devices need support only Start Field (SF) orders.

  Defining fields with only double-byte characters on an IBM 3270-class device or emulator requires the use of the field attribute DBC-Asia (attribute type X'43'). To enter double-byte characters on IBM 3270 terminals, the operator must be able to create shift-out/shift-in (SO/SI) characters.

In general, an IBM device must have double-byte character set (DBCS) capability, as defined in the *IBM 3270 Information System Data Stream Programmer's Reference*, to support the processing of double-byte characters.

## Determination of the Character Set

By default, your Pathway application uses the character set supported by the device. The default double-byte character set is determined as follows:

- For 3270 devices connected by the SNAX/XF subsystem, call SETMODE 144 to determine the character set.
- For 3270 devices connected by other access methods, the default character set is IBM-EBCDIC.
- For 6530-series terminals, read the device configuration to determine the double-byte character set.

For both 6530-class devices and 3270-class devices, the program unit is aborted if a SCREEN COBOL program unit is compiled with the statement CHARACTER-SET IS KANJI-KATAKANA but the run-time device does not support double-byte characters. In this case, you get the following error message:

```
ERROR - *3060* DEVICE DOESN'T SUPPORT DOUBLEBYTE CHARACTERS
```

## Data-Item Considerations

The *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* describes the SCREEN COBOL programming language that you use to write Pathway applications. In developing SCREEN COBOL program units with double-byte characters, you have particular considerations for mixed data items and subscripting.

### Mixed Data Items

The *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* discusses SCREEN COBOL language elements in general. When you develop Pathway applications for double-byte character sets, you especially need to be aware of mixed data items.

One-byte characters (Katakana and alphanumeric) and 2-byte (double-byte) characters can coexist in data items declared as PIC X. Such data items are called mixed data items. A PIC X(10) field can contain, for example, any of the following combinations in a mixed data item:

- Five 2-byte characters and no 1-byte Katakana or alphanumeric characters
- Four 2-byte characters and up to two 1-byte Katakana or alphanumeric characters
- Three 2-byte characters and up to four 1-byte Katakana or alphanumeric characters
- Two 2-byte characters and up to six 1-byte Katakana or alphanumeric characters

- One 2-byte character and up to eight 1-byte Katakana or alphanumeric characters

---

**Note.** Katakana characters are not classed as alphanumeric characters in PIC A items. PIC A items can consist only of letters of the Roman alphabet or space characters.

---

If a data item contains fewer than the maximum number of characters allowed, the appropriate number of padding space characters are added to the right. If a PIC X field contains no double-byte (2-byte) characters, it is not a mixed data item.

When manipulating a PIC X field, you must remember two points:

- Double-byte characters take two bytes of storage.
- On output to 3270-class devices, the data-conversion function converts the data stream from internal format to a format suitable for the external device. When the external device is an IBM 3270-class device, SO/SI (shift-out/shift-in) characters are inserted around each double-byte character substring. These SO/SI characters each take up one column of screen space. When the external device is a Fujitsu 3270-class device, a similar operation to that for an IBM 3270-class device is performed, but the substring framing characters do not occupy screen space.

  The use of SO/SI characters to bracket double-byte character set data is discussed later in this section.

Mixed data items are allowed in the Working-Storage Section, the Linkage Section, and the Screen Section of the Data Division of a SCREEN COBOL program. For both Working-Storage Section and Linkage Section entries, the maximum size of a data item is 16,000 double-byte characters or its equivalent (32,000 bytes). Mixed data items are not allowed in the Message Section of the Data Division of a SCREEN COBOL program.

## Subscripting Considerations

Subscripts are used to refer to elements in a table. Subscripts are needed because all table elements have the same name.

When you develop a Pathway application that uses double-byte characters, you must code the OCCURS clause to accommodate PIC X data items that might contain double-byte characters, as explained in the following paragraphs.

The left or right byte of a double-byte character in itself has no meaning. Referring to a subscripted data item, defined by using a PIC X clause containing double-byte data and redefined as PIC X(1) OCCURS *n* TIMES, might refer to only the left or right byte of a double-byte character. That half of the double-byte character by itself is undefined. For example:

```
WORKING-STORAGE SECTION.
      :
      :
01 WS-KANJI-DATA                        PIC N(05).
01 WS-UNDEFINED-DATA                    PIC X.
01 WS-NAME-1                            PIC N(05).
01 WS-GROUP-REDEF REDEFINES WS-NAME-1.
   02 WS-BYTE-DATA                      PIC X OCCURS 10 TIMES.
```

```
      :
      :
PROCEDURE DIVISION.
      :
      :
   MOVE WS-KANJI-DATA TO WS-NAME-1.
   MOVE WS-BYTE-DATA(1) TO WS-UNDEFINED-DATA.
```

The receiving data in this example (WS-UNDEFINED-DATA) is undefined because an individual byte of a double-byte character is meaningless.

Arrays defined by using a PIC N clause, rather than a PIC X clause, are referred to in units of two bytes.

## Developing SCREEN COBOL Programs for Double-Byte Character Sets

Program units written in SCREEN COBOL have four divisions: the Identification Division, the Environment Division, the Data Division, and the Procedure Division. In developing Pathway applications for double-byte character sets, you define specific attributes in the last three of these divisions.

### Environment Division

The Configuration Section of the Environment Division declares the operating environment of a SCREEN COBOL program. When you write Pathway applications for double-byte character sets, you need to consider the following syntax in the OBJECT-COMPUTER paragraph:

```
OBJECT-COMPUTER.  comment-word,

[ TERMINAL IS terminal-type [ , ] ]

[ CHARACTER-SET IS character-set-type ] .
```

#### TERMINAL IS Statement

All 3270 class devices—both IBM and Fujitsu—are identified by the keyword IBM-3270. All 6530 class devices are identified by the keyword T16-6530.

IBM 3270 devices use Start Field Extended (SFE) orders to support Kanji characters and Start Field (SF) orders to support Katakana characters. Fujitsu 3270 devices use Start Field (SF) orders for both Kanji and Katakana character sets.

## CHARACTER-SET IS Statement

The character-set-type provides support of national-use characters, that is, character sets that are not USASCII. The *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* lists the available character sets. To specify a double-byte character set, you use the keyword KANJI-KATAKANA:

```
[CHARACTER-SET IS KANJI-KATAKANA]
```

The KANJI-KATAKANA keyword indicates that the program source file can contain double-byte characters in data fields or literals. It instructs the compiler to allow the PIC N PICTURE clause described later in this section.

**Note.** The SCREEN COBOL compiler does not allow the use of double-byte characters as program names (for instance, paragraph names or variable names).

The following rules govern the use of the KANJI-KATAKANA keyword:

- The PIC N data type is supported only for program units that specify CHARACTER-SET IS KANJI-KATAKANA.
- Program units that do not use double-byte characters need not specify CHARACTER-SET IS KANJI-KATAKANA even if 1-byte Katakana characters are used.
- If a program unit specifies CHARACTER-SET IS KANJI-KATAKANA, both 1-byte and 2-byte Katakana character sets are supported; you can use whichever Katakana character set the device supports.

Depending on the terminal, support for 1-byte Katakana characters, 2-byte Katakana characters, and alphabetic characters varies:

- Applications that use only 1-byte alphabetic characters can run on any device for which the CHARACTER-SET IS KANJI-KATAKANA clause is valid.
- Applications that use 1-byte alphabetic characters as well as 1-byte Katakana characters can run only on 6530 Katakana terminals, on 6530 Kanji terminals, or on IBM 3270 terminals configured to support EBCDIC-Katakana.
- On 6530 Kanji terminals, both 1-byte and 2-byte Katakana character sets, as well as uppercase and lowercase alphabetic characters, are supported.
- IBM 3270 terminals support uppercase alphabetic characters with either lowercase alphabetic characters or 1-byte Katakana characters.
    - On IBM 3270 terminals configured to use both lowercase and uppercase alphabetic characters, a program unit cannot use 1-byte Katakana characters.
    - For IBM 3270 terminals configured to use 1-byte Katakana characters, the data stream conversion function of the TCP upshifts lowercase alphabetic characters in the outbound data stream. Such terminals can run applications with both lowercase and uppercase alphabetic characters as well as 1-byte Katakana characters. Inbound data from these devices does not contain lowercase alphabetic characters.

## Data Division

The Data Division describes the data that a SCREEN COBOL program creates, accepts as input, manipulates, or produces as input. As explained in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*, the Data Division has four sections.

In developing Pathway applications for double-byte character sets, you define specific attributes in the Working-Storage Section, the Linkage Section, and the Screen Section of the Data Division. There is no support for double-byte character sets in the Message Section. Special guidelines for writing the Screen Section of applications that use double-byte data are discussed in “Screen Section Considerations” later in this section.

### PICTURE Clause

The PICTURE clause of the Working-Storage Section, the Linkage Section, and the Screen Section of the Data Division supports the PIC N data type for double-byte character sets. The Message Section does not support PIC N.

The PICTURE clause is as follows:

```
{ PIC     }  [ IS ] character-string

{ PICTURE }
```

As well as the other character strings described in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*, you can define the character-string N to represent double-byte characters.

#### Working-Storage Section

The PICTURE clause of the Working-Storage Section defines the characteristics of an elementary item. In developing applications that use PIC N, consider the following:

- PIC N is valid only in program units that specify the KANJI-KATAKANA keyword in the CHARACTER SET clause of the OBJECT-COMPUTER paragraph of the Environment Division.
- The length of a double-byte-only data item can be 16,000 characters for a Working-Storage Section entry. Each double-byte character occupies two bytes per data item in memory, for a maximum of 32,000 bytes.
- SCREEN COBOL supports the character-string N only in PICTURE clauses that are not mixed with other character-string symbols. In other words, any data field that uses the N character-string symbol can have only contiguous Ns as a character-string symbol in the corresponding PICTURE clause. Only the following N PICTURE clauses are allowed:

```
PIC N
PIC N ... N    (maximum of 30 contiguous Ns)
PIC N(n)       (where n is 1 to 16,000)
```

Because variables defined with PIC X can contain mixed data—alphanumeric, numeric, and double-byte characters—you can combine double-byte characters with other data types in the Working-Storage Section by defining the entire data item with PIC X.

- If a VALUE clause is declared for a PIC N Working-Storage Section field, the value can consist only of characters from the Shift-JIS character set enclosed in quotation marks ("").
- You can use the THRU/THROUGH clause with 88-level data items associated with double-byte character set literals. Byte-by-byte comparisons of all items in the THRU/THROUGH clause are performed.

## Linkage Section

The Linkage Section associates the data items defined in the section with the data items defined in the Working-Storage Section of the calling program. In developing applications that use PIC N, consider the following:

- Data descriptions defined in the Linkage Section must have the same PIC clause specifications and use as the corresponding items in the Working-Storage Section.
- The length of a double-byte-only data item can be 16,000 characters for a Linkage Section entry. Each double-byte character occupies two bytes per data item in memory, for a maximum of 32,000 bytes.
- SCREEN COBOL supports the character-string N only in PICTURE clauses that are not mixed with other character-string symbols. In other words, any data field that uses the N character-string symbol can have only contiguous Ns as a character-string symbol in the corresponding PICTURE clause. Only the following N PICTURE clauses are allowed:

```
PIC N
PIC N ... N    (maximum of 30 contiguous Ns)
PIC N(n)       (where n is 1 to 16,000)
```

Because variables defined with PIC X can contain mixed data—alphanumeric, numeric, and double-byte characters—you can combine double-byte characters with other data types in the Linkage Section by defining the entire data item with PIC X.

## Screen Section

The PICTURE clause of the Screen Section defines the format in which the data appears on the terminal screen. In developing applications that use PIC N, consider the following:

- PIC N is valid only in SCREEN COBOL program units that specify the KANJI-KATAKANA keyword in the CHARACTER SET clause of the OBJECT-COMPUTER paragraph of the Environment Division.

- Data items declared as PIC N in the Screen Section must declare the PIC N attribute before all other attributes:

  ```
  SCREEN SECTION.
        :
  05 FIELD-10        AT 10, 20  PIC N
                                MUST BE ....
                                FILL    ....
                                USING WS-FIELD-10.
  ```

- The length of a double-byte-only data item can be 128 double-byte characters for a Screen Section entry. Each double-byte character occupies two bytes per data item in memory, for a maximum of 256 bytes.
- Unlike the Working-Storage Section and the Linkage Section, the Screen Section allows data items with mixed character-string symbols. A data field that uses the double-byte character-string symbol, N, can be combined with data types X, A, 9, 0, and B.

## REDEFINES Clause

In addition to the general rules that the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* outlines for the REDEFINES clause, there are special considerations for using it when you develop a Pathway application for double-byte character sets.

### Working-Storage Section

The REDEFINES clause in the Working-Storage Section of the Data Division allows the same computer storage area to be described in more than one way. The REDEFINES clause specifies that the storage area being defined is an alternate interpretation of a previously defined storage area:

```
REDEFINES  data-name-2
```

If you try to move a numeric field, a numeric edited field, a numeric noninteger, or a numeric literal to a PIC N data item, you get a syntax error. By redefining the double-byte-only PIC N field as an alphanumeric PIC X data item, you can make the move. For example:

```
WORKING-STORAGE SECTION.
    :
01 WS-KANJI-ONLY-FLD                                 PIC N(10).
01 WS-KANJI-TO-PICX-REDEF REDEFINES WS-KANJI-ONLY-FLD PIC X(20).
```

Only Screen Section data items may be defined as alphanumeric edited.

## Screen Section

In the Screen Section of the Data Division, the REDEFINES clause allows the same screen field to be described in more than one way. The REDEFINES clause specifies that the screen field being defined is an alternate interpretation of a previously defined field:

```
REDEFINES  field-name-2
```

IBM 3270 terminals have a limitation that affects the use of Screen Section redefinitions. PIC N fields on IBM devices allow only double-byte character set data; no single-byte data—including shift-out/shift-in characters—is allowed.

When you are redefining data items for Pathway applications that run on IBM 3270 devices, you must remember the following:

- A PIC N field implies that no single-byte characters are to be found in the data stream.
- A PIC X field is a mixed field. It can contain double-byte and single-byte characters. Double-byte character set substrings contained within a mixed field must be bracketed by SO/SI characters.

The data type of the field used in the operation determines the translation to be applied to the field and also the way the field is defined on the terminal. Translation is automatically done by the TCP, based on the field and the terminal. The TCP thus adds or strips shift-out/shift-in characters as needed.

Translation errors can occur if you redefine PIC N data items and then perform a DISPLAY or ACCEPT of the item by the name specified in the REDEFINES clause. On output to the terminal, this confusion can generate a terminal error by causing the TCP to insert SO/SI characters in a field initially defined as PIC N through a DISPLAY BASE operation. On input, DBCS substrings can appear with SO/SI framing characters in mixed data items, causing translation errors. A translation error is considered an editing error.

For instance, a PIC X field redefined as a PIC N results in a display error on the terminal because the PIC X clause creates a screen field that requires SO/SI insertion for display of double-byte characters, but the PIC N field used in subsequent DISPLAY writes double-byte data to the terminal, which does not contain SO/SI characters.

An ACCEPT of the PIC N redefinition causes a translation error because the terminal transmits double-byte character set data containing SO/SI characters, but the PIC N definition of the field implies that no single-byte data or SO/SI characters are in the data stream. Therefore, the translation fails and the TCP reports an editing error to the operator.

The following subsection, Screen Section Considerations, discusses the need for SO/SI characters more fully.

## Screen Section Considerations

When you develop Pathway applications for double-byte character sets, you must consider the following when you write the Screen Section of the Data Division of your SCREEN COBOL program unit.

### Permissible Character-String Symbols

In the Screen Section the character-string symbol N can be mixed with only the X, A, 9, 0, and B character-string symbols. Some examples are:

```
     SCREEN SECTION.
           :
     03 SS-KANJI-ONLY          PIC N at ...
           :
     03 SS-KANJI-CONTIG        PIC N....N at ...
   *                           (maximum of 30 Ns)
           :
     03 SS-KANJI-IMPLIED       PIC N(n) at ...
   *                           (where n is 1 to 256)
           :
     03 SS-KANJI-MIXED-ALPHA   PIC NXN at ...
           :
     05 SS-KANJI-MIXED         PIC NXNXAAXN at ...
   *                           (mixed)
           :
     03 SS-KANJI-MIXED-EDITED  PIC NXN0NBXN at ...
   *                           (mixed edited)
           :
     03 SS-PICX-DATA           PIC X(70) at ...
   *                            (Any data can be entered here.
   *                            It can consist of a
   *                            combination of single and
   *                            double-byte data.)
```

### Screen Field Limits

You need to be aware of special limitations when you design applications for IBM 3270 devices that declare screen fields containing double-byte or mixed (single-byte alphanumeric and double-byte) data and for Fujitsu terminals that include screen fields that wrap from one line to the next. Of course, no screen-field data item can be greater than 256 bytes. When one is too long, it is truncated, regardless of the device.

In creating screen fields for mixed items for IBM and Fujitsu equipment, the translation library inserts shift-out/shift-in characters into the data stream to bracket double-byte character substrings. On IBM equipment the shift characters occupy display space. On Fujitsu equipment the shift characters do not take display space and, therefore, do not affect the data displayed. Because the JET 6530 terminal and the 6530 terminal emulator use double-byte characters in Shift-JIS format, it is not necessary to insert shift characters into the data stream to be transmitted when using those terminals.

The TCP does the following when generating fields with possible double-byte characters that are to be written to a 3270 device:

1. Loads the data into the work buffer

   This operation includes all editing operations. It also ensures that the number of characters placed into the buffer does not exceed the size of the screen field.

2. Calls the appropriate translation routine

   Operations performed by the translation routines include the insertion of shift-out/shift-in characters. As a result of this insertion, the number of characters in the work buffer that must be sent to the device can increase. If the increase because of the insertion of shift characters exceeds the original buffer field size, the following occurs:

   - For an IBM device:

     The field is truncated to fit into the screen field. Because shift characters occupy display space, some user data might not appear on the screen. An error message is logged once for each program unit that truncates data. The translation routines make sure that no partial double-byte characters are included in a truncated string and also that the last double-byte string is terminated by a shift-out character.

   - For Fujitsu devices:

     The expanded field is sent to the device if no double-byte characters start in the last column of the screen. Because the load of the work buffer results in the actual number of data characters being less than or equal to the screen field, overflow does not occur. If a double-byte character starts in the last column of the screen, a space is inserted in the last column and the character that would have started at that position is started in the first column of the next line. The addition of the extra space can cause data sent to the Fujitsu to be truncated by one or two bytes.

   - For JET 6530 terminals and any PCT 6530 terminal emulator released by Compaq Japan:

     These devices do not correctly handle double-byte characters that start in column 80. They behave in the same way that a Fujitsu 3270 does when a screen field wraps from one line to the next.

As a programmer, you must be aware that an ACCEPT of a field truncated by a previous DISPLAY operation can result in the accepted data being different from what you intended.

### Additional Clause Considerations

The following conventions are checked by the compiler for these screen-field characteristics clauses. The *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* discusses each of these clauses fully. A double-byte-only field is a field that is declared by using only Ns in the PICTURE clause character string.

- The ADVISORY clause cannot be associated with a field that allows only double-byte data.
- When a FILL clause is used with a field that allows only double-byte data, the fill character must be a double-byte character.

  When a FILL clause is used with a PIC X or PIC A field, the fill literal must be a valid ASCII (single-byte) character.
- When the LENGTH clause is used with double-byte fields, the values assigned to the clause indicate the number of characters of the given type that are required for operator input. For example, the clause:

  ```
  LENGTH MUST BE 6
  ```

  means one of the following:

  - Six single-byte displayable ASCII characters must be entered for a PIC X(10) field.
  - Six single-byte alphabetic ASCII characters must be entered for a PIC A(50) field.
  - Six double-byte characters (having a total of 12 bytes) must be entered for a PIC N(30) screen field.

  If you use a mixed field, the LENGTH MUST BE clause refers to the absolute number of bytes that the operator must enter. For example, PIC A(10)N(5)X(5) with a LENGTH MUST BE 6 clause means that the operator must enter six alphabetic characters. A LENGTH MUST BE 11 clause is not possible here because the operator would have to enter ten alphabetic characters for the first ten bytes—and half of a double-byte character for the eleventh byte.

  The THRU or THROUGH variant of the functions in the same way.

  The LENGTH value for a screen field must be 256 or less.
- The THRU or THROUGH variant of the MUST BE clause is supported for fields that allow only double-byte data in the same way as it is in the Working-Storage Section.

  You can use the THRU/THROUGH with 88-level data items associated with double-byte character set literals. Byte-by-byte comparisons of all items in the THRU/THROUGH are performed.

- The UPSHIFT clause is a valid screen-field attribute for PIC N fields, but it is useful only on mixed fields—for example, PIC N(10)A(10).

  The translation routines upshift lowercase characters on output to IBM 3270 devices configured to use 1-byte Katakana characters. Upshifting of lowercase characters is not done on input for these devices because they cannot generate lowercase characters.

  Upshifting on output is not done for 3270-type devices configured to use both lowercase and uppercase 1-byte alphabetic characters. These devices cannot display or generate 1-byte Katakana characters.

- When the VALUE clause is used for a field that allows only double-byte data, the literal string provided must follow the same rules as those defined for a VALUE clause associated with a Working-Storage PIC N field.
- Programs that use a double-byte character set, identified by the CHARACTER-SET IS KANJI-KATAKANA clause, are restricted in their ability to define input control-character clauses. They can use only single-byte characters from the ASCII character set.

## Procedure Division

The Procedure Division includes all of the processing steps for the program. As the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* explains, these steps consist of SCREEN COBOL statements and sentences, grouped into paragraphs, procedures, and sections.

### IF Statement

SCREEN COBOL program units use an IF statement to evaluate a condition and then transfer control depending on whether the value of a condition is true or false. In addition to the general conventions for IF statements described in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*, when developing Pathway applications for double-byte character sets you must consider the following:

- Comparisons (using GREATER THAN, LESS THAN, EQUAL, and so on) of a PIC N data item or literal with a numeric data item (PIC 9) are not allowed.
- All other comparisons are allowed and are done on a byte-by-byte basis.

If a comparison of a numeric data item and a double-byte character set data item is attempted, the compiler issues an error message at the time of compiling.

## IF...DOUBLEBYTE Statement

The IF...DOUBLEBYTE statement tests for the existence of double-byte characters in an alphanumeric data item:

```
IF data-name [ IS ] [ NOT ] DOUBLEBYTE
```

Aligned double spaces are seen as %H2020 and are valid double-byte characters. A single space or a nonaligned space is not a double-byte character.

## MOVE Statements

SCREEN COBOL program units use MOVE and MOVE CORRESPONDING statements to transfer data from one data item to one or more other data items. In developing Pathway applications for double-byte character sets, you need to consider the conventions and restrictions for MOVE statements defined in the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*, especially the following:

- Numeric integers, numeric nonintegers, and numeric edited data items must not be moved to a data item that allows only double-byte (PIC N) data.
- A data item or literal that allows only double-byte (PIC N) data must not be moved to a numeric integer, a numeric noninteger, or a numeric edited data item.
- Violation of either of these rules causes the SCREEN COBOL compiler to issue a compilation error 453:

  ```
  ILLEGAL SENDING OR RECEIVING ITEM IN MOVE STATEMENT
  ```

- Moving any of the following figurative constants to a PIC N field is flagged as a compiler error:

  HIGH-VALUE
  HIGH-VALUES
  ZERO
  ZEROS
  ZEROES
  LOW-VALUE
  LOW-VALUES
  QUOTE
  QUOTES

  Moving the figurative constants SPACE or SPACES to a PIC N field is allowed.

  See the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual* for a list of the SCREEN COBOL compiler error messages.

- Only selected devices can be used to create SCREEN COBOL source images that contain double-byte characters. These devices must be capable of transmitting double-byte characters to the host in Shift-JIS format. Contact the Compaq Tokyo office for a list of such devices.

Table 8-1 summarizes the restrictions that apply to MOVE statements.

**Table 8-1. Restrictions on MOVE Statements**

| | Category of Receiving Data Item | | | | |
|---|---|---|---|---|---|
| **Category of Sending Data Item** | **Alphabetic** | **Alphanumeric** | **Alphanumeric Edited** | **Numeric Integer, Numeric Noninteger, Numeric Edited** | **Double-Byte Character** |
| Alphabetic | Yes | Yes | No | No | Yes* |
| Alphanumeric | Yes | Yes | No | Yes | Yes* |
| Alphanumeric Edited | Yes | No | No | No | Yes* |
| Numeric Integer | No | No | No | Yes | No |
| Numeric Noninteger | No | No | No | Yes | No |
| Numeric Edited | No | No | No | No | No |
| Double-Byte Character | No | Yes | Yes | No | Yes |

* Such MOVE operations move string data byte by byte; no editing or conversion is done.

## Example of Working-Storage Section and Screen Section

The following example shows a sample Working-Storage Section and its corresponding Screen Section.

For this Working-Storage Section,

```
WORKING-STORAGE SECTION.
      :
01 WS-KANJI-ONLY-FLD                                  PIC N(10).
01 WS-KANJI-TO-PICX-REDEF REDEFINES WS-KANJI-ONLY-FLD PIC X(20).
01 WS-KANJI-ONLY-FLD2                                 PIC N(10).
01 WS-KANJI-ONLY-FLD3                                 PIC N(10).
01 WS-KANJI-ONLY-FLD4                                 PIC N(10).
01 WS-ALPHA-NUMERIC-FLD                               PIC X(20).
01 WS-ALPHA-NUMERIC-FLD2                              PIC X(20).
01 WS-ALPHA-NUMERIC-FLD3                              PIC X(20).
```

the Screen Section would be as follows:

```
SCREEN SECTION.

*    Data translation occurs, if necessary, for each field
*    declared in these screen examples.  Data translation
*    consists of converting double-byte characters in
*    external form to internal form (Shift JIS) on
*    input or from internal form to device-specific
*    external form on output.  Such translation is normally
*    required only for the 3270-class devices produced by IBM
*    and Fujitsu.  Data entered from the Japanese version of
*    PCT, running on IBM or Fujitsu personal computers, does
*    not require input or output translation because the

*    emulators transmit double-byte characters in Shift-JIS
*    format.
        :
        :
05 SS-ALPHA-NUMERIC          PIC X(20) at ....
                             USING WS-KANJI-TO-PICX-REDEF.

*    No editing.  This is effectively a PIC X to PIC X move.
*    The programmer is responsible for data integrity.


05 SS-ALPHA-NUMERIC-EDITED1  PIC AANXBAAN0NXNXNX at ....

*    This PIC field uses 20 bytes.

                             USING WS-ALPHA-NUMERIC-FLD.

*    Outbound editing performed by the TCP consists of simple
*    insertion only, for example, the B here.  Inbound
*    editing performed by the TCP is done according to the
*    screen-item characters in the PIC clause.  For example,
*    the TCP expects an alphabetic character, followed by a
*    numeric, followed by a double-byte character set
*    character,and so on.

05 SS-ALPHA-NUMERIC-EDITED2  PIC AXNXBAXN0NXNXNX at ....
                             USING WS-KANJI-ONLY-FLD2.

*    Outbound editing performed by the TCP consists of simple
*    insertion only, for example, the B here.  Inbound
*    editing is performed by the TCP according to the screen-
*    item PICTURE clause.  The Working-Storage field cannot
*    contain data that has only double-byte characters.  The
*    programmer is responsible for data integrity.
```

```
05 SS-KANJI-ONLY3            PIC N(10) at ....
                             USING WS-KANJI-ONLY-FLD3.

*    On outbound editing, the TCP ensures that the data
*    displayed contains only DBCS characters.  The program is
*    aborted if this is not the case.  On inbound editing,
*    the TCP requires the operator to enter DBCS characters.


05 SS-ALPHA-NUMERIC-FLD3     PIC X(20) at ....
                             USING WS-KANJI-ONLY-FLD4.

*    No outbound or inbound editing is done.
```

# TCP SETMODE Functions and CONTROL Operations

As a Pathway application programmer, you might need to know the *Guardian* operating environment SETMODE functions and CONTROL operations used by the TCP. This section describes the TCP's use of SETMODE functions and CONTROL operations.

The *Guardian Procedure Calls Reference Manual* provides full descriptions of all SETMODE functions and CONTROL operations. For programming information about the SETMODE and CONTROL file-system procedures, refer to the *Guardian Programmer's Guide*, the *Enscribe Programmer's Guide*, and the relevant data communications manuals.

Table 9-1 describes SETMODE functions and CONTROL operations that the TCP executes on terminal files when performing the indicated actions.

**Table 9-1. TCP SETMODE and CONTROL Activities**

| **TCP Activity** | **Guardian SETMODE Function** | | | | | | **Guardian CONTROL Operation** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **8** | **11** | **20** | **54** | **144** | **150** | **1** | **11** | **12** | **22** | **26** |
| START/RUN TERM (Open)[2] | **x** | **x (A)** | | | | **x (B)** | | **x** | | | **x (C)** |
| Program Termination (Close)[3] | **x** | **x (A)** | **x** | | | | | | **x** | **x (D)** | |
| DISPLAY[1] | **x** | **x** | **x (A)** | | **x** | | **x (A)** | | | | |
| ACCEPT[1] | | **x** | **x (A)** | | | | | | | | |
| SEND MESSAGE[1] | **x** | **x** | **x** | | | | | | | **x (D)** | **x (E)** |
| PRINT SCREEN[1] | | | **x** | **x** | **x** | | | | | | |

## SETMODE Functions

SETMODE is used to set device-dependent functions. When you design an intelligent device support (IDS) requester to communicate with a front-end process (FEP), you are concerned with the SETMODE functions described in Table 9-2.

## Table 9-2. TCP SETMODE Functions

| Operation | Description |
|---|---|
| 8 | Set system transfer mode (default is configured). |
| | param1.15 = 0 conversational mode |
| | = 1 page mode |
| | param2 sets the number of retries for I/O operations. |
| | NOTE: param2 is used with 6530 terminals only. |
| 11 | Set break ownership |
| | param1 = BREAK disabled (default setting) |
| | = cpu, pin BREAK enabled |
| | Terminal access mode after BREAK is pressed: |
| | param2 = 0 normal mode (any type file access permitted) |
| | = 1 BREAK mode (only BREAK-type file access permitted) |
| 20 | Set system echo mode (default is configured). |
| | param1.15 = 0 system does not echo characters as read |
| | = 1 system echoes characters as read |
| | param2 is not used with function 20. |
| 54 | Return control unit and device assigned to subdevice. |
| | param1 is not used with function 54. |
| | param2 is not used with function 54. |
| | last-params[0].0:7 = 0 |
| | .8:15 = subdevice number known by AM3270 |
| | [1].0:7 = standard 3270 control-unit address |
| | .8:15 = standard 3270 device address |
| 144 | Set LU character set and double-byte character code. |
| | param1 must be omitted for function 144. |
| | param2 must be omitted for function 144. |
| | last-params[0].0 = 1 : EBCDIC ASCII conversion is done by ACCESS process |
| | .1:7 = IBM device type |
| | 1 : IBM-3277 |
| | 2 : not 3277 or 3276 |
| | 3 : IBM-3276 |
| | .8:15 = value of LU attribute ALLOWEDMIX |
| | [1].0:7 = LU characterset |
| | 0 : ASCII (USASCII) |
| | 9 : EBCDIC (IBM-EBCDIC) |
| | 14 : KATAKANA EBCDIC |
| | [1].8:15 = LU DBCS |
| | 0 : No DBCS |
| | 2 : IBMKANJI |
| | 3 : IBMMIXED |
| | 5 : JEFKANJI |

## CONTROL Operations

CONTROL is used to perform device-dependent I/O operations. When you design an intelligent device support (IDS) requester to communicate with a front-end process (FEP), you are concerned with the CONTROL operations described in Table 9-3. CONTROL 26 is discussed in detail later in this section.

**Table 9-3. TCP CONTROL Operations**

| Operation | Description | Parameter |
|---|---|---|
| 1 | Control forms for conversational mode (subtypes 0,2,3) | 0 = form feed (send %014)<br>1-15 = vertical tab (send %013)<br>16-79 = skip param - 16 lines |
| 11 | Wait for modem connect | none |
| 12 | Disconnect modem (hang up) | none |
| 22 | Cancel an AM3270 I/0 operation | none |
| 26 | Request immediate completion of all outstanding I/O requests without loss of data by the recipient of the CONTROL 26 request | none |

## Pathway/iTS and CONTROL 26

Compaq *NonStop*™ Pathway/iTS intelligent device support (IDS) SCREEN COBOL requesters can interact with front-end processes outside of the PATHMON environment that control intelligent devices such as automated teller machines, airline reservation terminals, and personal computers. One type of front-end process, for example, could be a SNAX/HLS application process.

The IDS requester sends messages to the FEP through SEND MESSAGE statements and receives responses by the associated REPLY clauses.

As it interprets SEND MESSAGE statements and the associated REPLY clauses, the TCP initiates the necessary write and read operations by issuing file system procedure calls such as WRITE, READ, and WRITEREAD.

When a SEND MESSAGE statement includes a TIMEOUT or ESCAPE ON UNSOLICITED MESSAGE clause, the TCP might have to terminate the underlying read operation prematurely if either of those events occurs.

Ordinarily the TCP does this by issuing a CANCEL file system procedure call. CANCEL calls are, however, ineffective for determining loss of data between the TCP and the FEP. If the TCP uses a CANCEL call to terminate an outstanding read request, the FEP cannot detect that the read no longer exists. When the FEP eventually responds to the canceled read, it is likely that data will be lost and the integrity of the context for subsequent operations compromised.

CONTROL 26 provides a better alternative.

## CONTROL 26 Defined

CONTROL 26 is a CONTROL file system procedure call that allows nonprivileged processes to cooperate with one another in bringing about the orderly termination of outstanding read operations.

When an IDS requester is communicating with an FEP that supports the use of CONTROL 26, the two processes use CONTROL 26 calls and the appropriate responses jointly to terminate execution of the underlying read operation with no loss of data or context. As used within the Pathway environment, the general format of a CONTROL 26 procedure call is as follows:

```
CONTROL ( fnum
        , 26
        , parameter
        , tag ) ;
```

*fnum*

identifies the file to the FEP.

*parameter*

= 1 triggers an initialization sequence.

= 300 specifies that a CONTROL 26 request is to be issued to the FEP and that both the original read and the CONTROL 26 must be completed within five minutes (300 seconds).

*tag*

is an optional tag field.

In the rest of this section, the phrase CONTROL 26,1 refers to a CONTROL 26 call whose *parameter* field contains the value 1, while the phrase CONTROL 26,300 refers to a CONTROL 26 call whose *parameter* field contains the value 300.

## How CONTROL 26 Works

Essentially, the TCP uses CONTROL 26 as follows. When a timeout or escape on unsolicited message occurs, the TCP issues a CONTROL 26 call to the FEP that is responsible for completing the read. That process then has up to five minutes (300 seconds) to do any of the following:

- Complete the outstanding read by sending the requested data (return code = 0)
- Complete the outstanding read by specifying that it has no data to send (return code = 187)
- Complete the outstanding read by specifying that something wrong has happened that compromises the integrity of subsequent data (return code = 188)

- Complete the outstanding read by specifying that an associated input operation within the FEP is still in progress and that the resultant data can be obtained by a subsequent read operation (return code = 189)

The FEP must complete both the original read request and the CONTROL 26 call within the allotted five minutes or the TERMINATION-STATUS register is set to 14 and control passes to the SEND MESSAGE statement's ON ERROR clause. If there is no ON ERROR clause, the TCP suspends the requester and logs error 3174 to the devices specified by the PATHCOM commands LOG1 and LOG2.

## CONTROL 26 Initialization

As soon as the TCP opens the front-end process, the two processes engage in an initialization sequence in which the TCP determines whether the FEP detects the use of CONTROL 26.

That sequence is as follows:

1. The TCP issues a CONTROL 26,1 call.
2. The FEP sends back a reply code of 70 to indicate that it supports CONTROL 26.

Under normal circumstances, the TCP opens front-end processes with a nowait depth of 1 (allowing only a single I/O request to be outstanding at any given time).

To be able to use CONTROL 26 calls, however, the TCP must open the FEP with a nowait depth of 2 (allowing both the original read request and a CONTROL 26 request to be outstanding concurrently).

The IOPROTOCOL parameter of the PATHCOM SET TERM command allows you to tell the TCP that the FEP supports the use of CONTROL 26. The TCP still performs the initialization sequence but does it differently depending upon the value of IOPROTOCOL.

### IOPROTOCOL = 0

IOPROTOCOL = 0, which is the default value, declares that you do not know if the FEP supports CONTROL 26.

In this case, the TCP opens the FEP with a nowait depth of 1 and then issues the CONTROL 26,1 call.

If the FEP responds with 70, the TCP closes the file, reopens it with a nowait depth of 2, and subsequently uses CONTROL 26 calls to terminate read requests prematurely.

If the FEP responds with any code other than 70, the TCP presumes that it does not understand the use of CONTROL 26; the TCP subsequently uses CANCEL calls to terminate read requests prematurely.

### IOPROTOCOL = 1

IOPROTOCOL = 1 declares that the FEP does support CONTROL 26.

In this case, the TCP opens the FEP with a nowait depth of 2 and then issues the CONTROL 26,1 call.

If the FEP responds with 70, the TCP subsequently uses CONTROL 26 calls to terminate read requests prematurely.

If the FEP responds with any code other than 70, the TCP suspends the IDS requester and records the event with an error code of 3054 in the PATHMON log file.

By specifying IOPROTOCOL = 1 when you are certain that CONTROL 26 will be used, you eliminate some of the overhead inherent to the initialization sequence. Overhead could be substantial if you are configuring and starting many requesters.

A requester written to handle CONTROL 26 does not operate properly if the FEP does not support CONTROL 26.

## Subsequent CONTROL 26 Calls

After the IDS requester and FEP have agreed to use CONTROL 26, the FEP should always respond to any subsequent CONTROL 26 calls with a return code of 0.

The other supported return codes (187, 188, and 189) should be used only with the underlying read operation—never with a CONTROL 26 call.

When the TCP issues a CONTROL 26 call, the FEP must, if it is able to do so, respond in either of the following ways:

- By completing the read with valid data and a return code of 0 and by completing the CONTROL 26 call with a return code of 0
- By completing the read with a return code of 187, 188, or 189 and by completing the CONTROL 26 call with a return code of 0

## Testing TERMINATION Codes

As a SCREEN COBOL programmer designing and coding an IDS requester, you do nothing to initiate the use of CONTROL 26. The TCP determines whether to use CONTROL 26, based on whether the external FEP supports the use of CONTROL 26 and on the SET TERM IOPROTOCOL specified at configuration.

What you must do, however, is test for certain TERMINATION-STATUS and TERMINATION-SUBSTATUS codes in your ON ERROR and ESCAPE ON UNSOLICITED MESSAGE paragraphs that can be generated by the use of CONTROL 26.

## Unsolicited Message Completions

When a CONTROL 26 call is issued as the result of an escape on unsolicited message, th TERMINATION-STATUS register contains an index pointing to the location of ESCAPE ON UNSOLICITED MESSAGE clause in the SEND MESSAGE statement. Table 9-4 summarizes the various TERMINATION-SUBSTATUS values that should then be tested for. The action in the Meaning column is only suggested; the action taken depemds on the FEP itself.

**Table 9-4. ESCAPE ON UNSOLICITED MESSAGE Completions**

| Terminatio nStatus | Terminatio nSubstatus | Meaning |
|---|---|---|
| * | 187 | The FEP assisted in terminating the outstanding read. No data was returned or lost by the FEP and the context is intact.<br><br>Process the unsolicited message and start another normal SEND MESSAGE statement. |
| * | 188 | The FEP assisted in terminating the outstanding read. The FEP might have lost some data, or the integrity of the context might have been compromised.<br><br>After processing the unsolicited message, terminate the session with the FEP and try to start a new one. |
| * | 189 | The FEP assisted in terminating the outstanding read; however, the operation is still in progress within the domain of the FEP. The data will be queued and you can retrieve it at a later time with a subsequent read operation. No data was lost by the FEP and the context is intact.<br><br>Process the unsolicited message and then issue a SEND MESSAGE statement that causes only a read operation to be posted. This allows for the retrieval of the outstanding read when it completes. |

* Index of the user's ESCAPE ON UNSOLICITED MESSAGE clause in the SEND MESSAGE statement.

Note: When the TCP issues a CONTROL 26 and the FEP replies to the original I/O request with data and error 0 instead of with data and one of the above errors, the ESCAPE ON UNSOLICTED clause is not processed even though a UMP message arrived. Instead, the SEND MESSAGE statement completes normally, using the returned data, and the UMP message is queued for processing at a later time. To detect the arrival of an unsolicited message, poll the PW-UNSOLICITED-MESSAGE-QUEUED register for a YES value.

## Timeout and Error Completions

When a CONTROL 26 call is issued as the result of a timeout, control passes to the ON ERROR code associated with the SEND MESSAGE statement. Table 9-5 summarizes the various TERMINATION-SUBSTATUS values that your ON ERROR clauses should test for.

**Table 9-5. Timeout and Error Completions**

| TerminationStatus | TerminationSubstatus | Meaning |
|---|---|---|
| 1 | 40 | A timeout occurred. The FEP assisted in terminating the outstanding read. No data was lost by the FEP and the context is intact.<br><br>Perform whatever timeout recovery action is appropriate for your particular application. |
| 1 | 188 | A timeout occurred. The FEP assisted in terminating the outstanding read. The FEP might have lost some data, or the integrity of the context might have been compromised.<br><br>Terminate the session with the FEP and try to start a new one. |
| 1 | 189 | A timeout occurred. The FEP assisted in terminating the outstanding read; however, the operation is still in progress within the domain of the FEP. The data will be queued, and you can retrieve it later with a subsequent read operation. No data was lost by the FEP and the context is intact.<br><br>Issue a new read operation with another SEND MESSAGE statement.. |
| 1 | Any other value | An I/O error occurred in conjunction with the original read operation. TERMINATION-SUBSTATUS specifies the file system error code returned with the read completion.<br><br>Perform whatever recovery action is appropriate for the particular type of file system error. |
| 13 | Nonzero value | A timeout or ESCAPE ON UNSOLICITED MESSAGE occurred. An I/O error also occurred in conjunction with the CONTROL 26 call. TERMINATION-SUBSTATUS specifies the file system error code returned with the CONTROL 26 completion.<br><br>Perform whatever recovery action is appropriate for the particular type of file system error. |
| 14 | Not used | A timeout or ESCAPE ON UNSOLICITED MESSAGE occurred. The FEP did not, however, respond to both the original read and the CONTROL 26 within the allotted five minutes.<br><br>Perform whatever timeout recovery action is appropriate for your particular application. Subsequent operation with the FEP can result in errors. |

# 10 Handling Errors

This section discusses the following topics related to error handling in SCREEN COBOL requesters:

- Terminal Errors
- Handling of SEND statement errors

The Pathway to TUXEDO translation server can also return error messages to requesters. Information about these errors is given in the *NonStop™ TUXEDO System Pathway Translation Servers Manual*.

## Terminal Errors

During terminal startup, the TCP retries, aborts, or suspends a terminal depending on the terminal error that had occured.

If an error occurs after the application is running on the terminal, the TCP invokes the user recovery routines in the SCREEN COBOL program whenever possible. The USE FOR SCREEN RECOVERY clause is invoked when there are terminal or communications errors, processor failures, or terminal suspension. Typically, the SCREEN COBOL program displays an advisory text message on the screen, and the user must then take corrective action.

If no user error recovery is provided in the SCREEN COBOL program, then the TCP takes its own action on terminal errors. The USE FOR TERMINAL-ERRORS clause is invoked when there is an irrecoverable error due to a terminal error or communications device error. This clause cannot be used for programs that communicate with intelligent devices. If the USE FOR TERMINAL-ERRORS clause is present, suspension is overridden (in most cases), so the USE FOR SCREEN RECOVERY clause would not be invoked.

| | |
|---|---|
| Error 140 | Suspends the terminal immediately, unless user exception-handling code is available in the the USE FOR TERMINAL-ERRORS clause or in an ON ERROR clause within a CALL statement, and invokes SCREEN COBOL exception-handling code if available. |
| Error 191 | Initiates a DISPLAY RECOVERY operation, and invokes any code in the USE FOR SCREEN RECOVERY clause. |
| Errors 300 to 511 | Suspends the program immediately, unless user exception code is available in the USE FOR TERMINAL-ERRORS clause or in an ON ERROR clause within a CALL statement, and invokes SCREEN COBOL exception-handling code if available. |
| Other Errors | If the retries do not succeed, then the TCP suspends the terminal, unless user exception-handling code is available in the USE FOR TERMINAL ERRORS clause or in an ON ERROR clause within a CALL statement, and invokes SCREEN COBOL exception-handling code, if available. |

## SEND Statement Errors

This subsection suggests ways to handle the processing of SEND statement errors. You can decide what is most appropriate to your particular application environment. For additional information about SEND and SEND MESSAGE errors, refer to the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

### Responding to SEND Errors

Tables 10-1 through 10-5 suggest how your requester ON ERROR code can respond to error conditions arising from the execution of a SEND statement.

The specified numeric values represent the contents of the TERMINATION-STATUS special register. The accompanying text indicates what the particular error code means.

For descriptions of what actions the system takes if you omit the ON ERROR clause, refer to the *Compaq NonStop™ Pathway/iTS SCREEN COBOL Reference Manual*.

The codes in Table 10-1 reflect error conditions that could be transient; the problem might go away spontaneously.

**Table 10-1. Requester SEND Errors for Transient Conditions**

| Numeric Value | Meaning |
|---|---|
| 1 | Server class frozen |
| 2,3 | Resource unavailable |
| 4 | Link denied by PATHMON process or link rejected by server |
| 12 | I/O error |
| 14 | Maximum number of PATHMON processes has been reached |
| 18 | I/O error in attempt to communicate with the PATHMON process |

Because the error conditions might be recoverable, you can retry the failed SEND statement, perhaps with a time delay, some finite number of times.

Before each retry, send a message to the terminal to let the operator know what is happening (such as TRANSIENT ERROR, RETRYING).

If the SEND statement fails all of the allotted retries, the ON ERROR code should send another message to the terminal telling the operator what is happening (such as PERSISTENT ERROR, TERMINATING EXECUTION), log an error message to an appropriate server, and then perform a STOP RUN statement.

Error 4 could be caused by the server’s allocating too little space for $RECEIVE messages. To avoid this problem, the number of links specified in the server (for example, in theCOBOL85 RECEIVE-TABLE OCCURS clause) should be greater than the value specified in the SET SERVER MAXLINKS parameter in PATHCOM. The default value for MAXLINKS is an unlimited number of links; therefore, to avoid this problem, MAXLINKS must be set to a value.

Other causes, such as a security violation, could also result in error 4.

Error 12 could be caused by a timeout error (termination substatus 40) when a server is in debug mode. If this situation occurs, the operator should do the following:

Use the PATHCOM STATUS PATHMON command to find server classes in the LOCKED state.

Identify the server program file for each locked server class.

Issue the TACL command STATUS *, PROG *object-file-name* to list all running processes.

Stop these processes from TACL.

For more information about timeout errors for servers in debug mode, refer to the *NonStop™ TS/MP Pathsend and Server Programming Manual*.

The codes in Table 10-1 reflect programming errors that are essentially nonrecoverable.

**Table 10-2. Requester SEND Errors for Nonrecoverable Programming Problems**

| Numeric Value | Meaning |
|---|---|
| 5 | Server class undefined |
| 6 | Invalid server-class name |
| 10 | Undefined reply code |
| 15 | Undefined system name |
| 16 | Invalid system name |
| 17 | Invalid PATHMON process name |

These errors typically occur only during application debugging. After the application modules have been thoroughly tested, these codes do not normally occur in a production environment.

Note that for error code 10 no data is available, even though a reply message is received. If your application cannot anticipate all valid reply codes, use the REPLY CODE OTHER syntax in the SEND statement to prevent error code 10.

Upon detecting any of these error conditions, your ON ERROR code should send a message to the terminal telling the operator what is happening (such as FATAL CONFIGURATION PROBLEM, TERMINATING EXECUTION), log an error record to an appropriate server, and then perform a STOP RUN statement.

The codes in Table 10-1 reflect configuration errors that are essentially nonrecoverable.

**Table 10-3. Requester SEND Errors for Configuration Problems**

| Numeric Value | Meaning |
|---|---|
| 7 | Message too large |
| 8 | Maximum reply too large |

These errors indicate that the SERVERPOOL (7) or MAXREPLY (8) parameter supplied in the PATHCOM SET TCP command at configuration time is smaller than that required by one of the message or reply definitions declared in the requester.

No message is sent and no reply data is available.

Upon detecting either of these error conditions, your ON ERROR code should send a message to the terminal telling the operator what is happening (such as FATAL CONFIGURATION ERROR, TERMINATING EXECUTION), log an error record to an appropriate server, and then perform a STOP RUN statement.

The code in Table 10-1 indicates that the reply message received from the server was either longer or shorter than the reply format defined within the requester.

**Table 10-4. Requester SEND Error for Invalid Reply Length**

| Numeric Value | Meaning |
|---|---|
| 11 | Invalid reply length |

If the received reply message is shorter than the working-storage structure defined for it, the message is available in the target working-storage data structure. The actual length of the received message is placed in the TERMINATION-SUBSTATUS special register.

If the received reply message is longer than the working-storage structure defined for it, the message is available in the target working-storage data structure; however, it is truncated to the length of the working-storage structure. In this case, TERMINATION-SUBSTATUS contains a value greater than the length of the working-storage structure.

You can design your program to use a value of 11 for the TERMINATION-STATUS special register to process variable-length replies, as described in the section that follows this table. If you design your program to reject variable length replies, your ON ERROR code should send a message to the terminal telling the operator what is happening (such as BAD MESSAGE LENGTH, TERMINATING EXECUTION), and then log an error record to an appropriate server and perform a STOP RUN statement.

The code in Table 10-1 indicates that the requester program is operating in transaction mode (that is, within the bounds of a BEGIN-TRANSACTION and END-TRANSACTION statement pair), but the server to which it is attempting to send data is not configured for Compaq Transaction Management Facility (*TMF*) operation (TMF OFF was specified in the applicable PATHCOM SET SERVER command).

**Table 10-5. Requester SEND Error for Transaction-Mode Violation**

| Numeric Value | Meaning |
|---|---|
| 13 | Transaction-mode violation |

This error condition is essentially nonrecoverable. In this case, your ON ERROR code should send a message to the terminal telling the operator what is happening (such as SERVER NOT CONFIGURED FOR TMF, TERMINATING EXECUTION), log an error record to an appropriate server, and then perform a STOP RUN statement.

## Processing Variable-Length Server Replies

When your requester is designed to receive variable-length replies from a server, error code 11 is a normal and common occurrence. In such a case, your ON ERROR code must be designed to respond to it properly.

Assume that the requester sends a message to a server asking for the names of all customers that have been added to the database during the past week. In this case, the response from the server at any given time contains a greater or lesser number of names.

The requester's SEND statement can provide a reply data structure to accommodate a reasonable maximum number of names. For example, if past performance shows that 10 to 12 new customers are typically added each week and that the best single week yielded 17 new customers, it is reasonable to use a reply data structure that can accommodate up to 20 customer names.

The code in this case—where the data is shorter than the maximum allowed—looks like the following. (If the data were longer than the maximum permitted, error 11 would reflect a nonrecoverable programming error.)

```
      DATA DIVISION.

         01 PROCESSING-STATE  PIC X(4), VALUE "GO  ".

         01 NEW-NAME-REQUEST    PIC 9(4) comp.

         01 NEW-NAME-REPLY.
            05 REPLY-CODE       PIC 9(4) comp.
            05 FUNCTION-CODE    PIC 9(4) comp.

*   Function-code 1 signifies a new name query.

            05 NUMBER-OF-NAMES  PIC 9(4) comp.
            05 NEW-CUSTOMER-NAMES
               PIC X(30) OCCURS 20 TIMES.

      PROCEDURE DIVISION.

        MAIN-PARAGRAPH.
             .
             .
          PERFORM new-name-query.
```

```
            IF PROCESSING-STATE = "STOP" GO TO
              MAIN-PARAGRAPH-EXIT.
                .
                .
          MAIN-PARAGRAPH-EXIT.
            EXIT.

          NEW-NAME-QUERY.
            SEND new-name-request TO customer-data-base
              REPLY CODE 1 YIELDS new-name-reply
                ON ERROR GO TO analyze-error.

          CONTINUE-PROCESSING.
*
*   Process the returned names.  Control passes here from the
*   ANALYZE-ERROR paragraph when the number of names returned
*   is greater than or less than 20.  Control passes here
*   from the NEW-NAME-QUERY paragraph when the number of
*   names returned is exactly 20.
*
            GO TO send-processing-exit.

          ANALYZE-ERROR.
            IF TERMINATION-STATUS = 11 AND
              function-code = 1 THEN GO TO
              CONTINUE-PROCESSING
            ELSE MOVE "STOP" TO PROCESSING STATE.

          SEND-PROCESSING-EXIT.
            EXIT.
```

# A The MAKEUL Macro

The MAKEUL macro performs pTAL compilations of user-written user conversion procedures and creates the TNS/R native user library for the TCP using the `nld` utility.

The syntax and options are as specified below:

```
MAKEUL command-option [command-option....]
```

*`command-option`*

is a space-separated list and can be one of the following:

`-src` *`source-filename`*

is the file name of the pTAL source file.

`-obj` *`object-filename`*

is the file name of the pTAL object file.

`-out` *`output filename`*

is the file name where the output is to be sent. If it is not specified, then the output is sent to the terminal.

`-lib` *`library-filename`*

is the file name of the user libary.

`-loc` *`tcplib-location`*

is the volume and subvolume where the TCPLIB file resides. The TCPLIB file is needed to build the user library. If this option is not specified, the MAKEUL macro will search for the TCPLIB file in the current subvolume and then in $SYSTEM.ZPATHWAY.

`-pTAL` *`ptal-location`*

is the volume and subvolume where the pTAL compiler resides. Default is taken as $SYSTEM.SYSTEM.

`-nld` *`nld-location`*

is the volume and subvolume where the `nld` utility resides. Default is taken as $SYSTEM.SYSTEM.

The MAKEUL macro has the following features:

- If an option is specified twice, the second option is used. For example, if the -src option is specified twice with two different file names, the second -src option becomes effective.
- If the object file name and the library file name are the same, the object file name is overwritten with the library file name.
- At the end of execution of the macro, status information is displayed. This information includes the names of the source file, object file, and library file and the locations of the TCPLIB file, the pTAL compiler, and the `nld` utility.
- If the -loc option is not specified, the MAKEUL macro looks for the TCPLIB file first in the current subvolume, then in the installation subvolume (ISV) $SYSTEM.ZPATHWAY. If the ISVs are in another volume, then you can write another macro that invokes MAKEUL, as follows:

  ```
  MAKEUL -loc $ISVVOL.ZPATHWAY %*%
  ```

  You can also specify where the TCPLIB file is located by setting the default value of the variable ISV_VOL to the volume where the ISVs are located, in the code of MAKEUL macro: for example,

  ```
  #SET ISV_VOL $MYISV
  ```

  For more information about this variable, see “Set the Default Values” within the MAKEUL code.
- By default, the SYMBOLS option is passed to the pTAL compiler. You can modify this option by setting the variable PTAL_OPTIONS accordingly. This variable should contain pTAL options separated by commas: for example,

  ```
  #SET ptal_options SYMBOLS,SUPPRESS
  ```

  For more information about this variable, see “Set the Default Values” within the MAKEUL code.

## Examples

For creating a pTAL object file called PTOBJ from the pTAL source file PTSRC and sending the output to the terminal, the syntax is as follows:

```
> MAKEUL -src PTSRC -obj PTOBJ
```

For creating a pTAL object file called PTOBJ from the pTAL source file PTSRC and sending the output to the file PTOUT, the syntax is as follows:

```
> MAKEUL -src PTSRC -obj PTOBJ -out PTOUT
```

For creating a user library file called USERLIB from the pTAL source file PTSRC and sending the output to the terminal, the syntax is as follows:

```
> MAKEUL -src PTSRC -lib USERLIB
```

For creating a user library file called USERLIB from the pTAL object file PTOBJ and sending the output to the file PTOUT, the syntax is as follows:

```
> MAKEUL -obj PTOBJ -lib USERLIB -out PTOUT
```

For creating a user library file called USERLIB from the pTAL source file PTSRC with the intermediate pTAL object file as PTOBJ and sending the output to the terminal, the syntax is as follows:

```
> MAKEUL -src PTSRC -obj PTOBJ -lib USERLIB
```

For creating a user library file called USERLIB from the pTAL source file PTSRC and sending the output to the terminal, the syntax is as follows:

```
> MAKEUL -src XYZ -lib USERLIB -src PTSRC
```

## Error Messages

```
*ERROR* Illegal command option specified. Can be only
-src, -obj, -lib, -out, -loc, -ptal, -nld
```

**Cause.** An option other than -src, -obj, -lib, -out, -loc, -nld, or -ptal was specified.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct option.

```
*ERROR*  At least two out of the options -src, -obj, -lib
must be specified
```

**Cause.** At least two out of the options -src, -lib, and -obj were not specified.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify at least two out of the options -src, -obj, and -lib.

```
*ERROR* Illegal source file specified
```

**Cause.** The source file specified is not a valid *Guardian* file name.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a valid Guardian file name as the pTAL source-file name.

```
*ERROR* Source file is not an EDIT file (Code 101) :
source-filename
```

**Cause.** The specified source file name is not an EDIT type of file.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a pTAL source file with the -src option.

```
*ERROR* Source file specified does not exist :
source-filename
```

**Cause.** The specified pTAL source file does not exist.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify an existing pTAL source file with the -src option.

```
*WARNING* Unable to delete object file : object-filename.
Error # error-number
```

**Cause.** The specified pTAL object file is an existing file that could not be deleted because of the error specified in the error message.

**Effect.** The macro automatically recovers from this error and creates an object file called ZZOB*. While doing so, it displays the following message:

```
*** Changing the object file to new-object-filename
```

**Recovery.** None required.

```
*ERROR* Illegal object file specified
```

**Cause.** The specified object file is not a legal Guardian file name.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a valid Guardian file as the pTAL object file name.

```
*ERROR* Object file specified is not a pTAL object file
(Code 700) : object-filename
```

**Cause.** The specified object file is not a pTAL object file. This error will occur only when the user is trying to create a user library file from the pTAL object file by using the following command :

```
> MAKEUL -obj object-filename -lib library-filename &
> [-out output-filename]
```

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a pTAL object file with the -obj option.

```
*ERROR* Object file specified does not exist :
object-filename
```

**Cause.** The specified object file does not exist. This error will occur only when the user is trying to create a user library file from the pTAL object file by using the following command :

```
> MAKEUL -obj object-filename -lib library-filename &
> [-out output-filename]
```

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify an existing pTAL object file with the -obj option.

```
*WARNING* Unable to delete library file : library-filename.
Error # error-number
```

**Cause.** The specified user library file could not be deleted because of the error specified in the error message.

**Effect.** The macro automatically recovers from this condition and creates a library file called ZZUL*. While doing so, it displays the following message:

```
*** Changing the library file to new-library-filename
```

**Recovery.** None required.

```
*ERROR* Illegal library file specified
```

**Cause.** The specified library file is not a legal Guardian file name.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a valid Guardian file as the user library file name.

```
*WARNING* Unable to delete output file : output-filename.
Error # error-number
```

**Cause.** The specified output file is an existing file that could not be deleted because of the error specified in the error message.

**Effect.** The macro automatically recovers from this error and creates an output file called $S.#MAKEUL. While doing so, it displays the following message:

```
*** Changing the output file to $S.#MAKEUL
```

**Recovery.** None required.

```
*ERROR* Illegal output file specified
```

**Cause.** The specified output file is not a legal Guardian file name.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a valid Guardian file as the output file name.

```
*ERROR* TCPLIB does not exist in the location specified :
tcplib-location.TCPLIB
```

**Cause.** TCPLIB does not exist in the location specified.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for TCPLIB.

```
*ERROR* Illegal location specified for TCPLIB
```

**Cause.** An invalid volume-subvolume name combination was specified as the location for TCPLIB.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for TCPLIB.

```
*ERROR* Invalid TCPLIB specified :  tcplib-location.TCPLIB
```

**Cause.** TCPLIB present in the location specified is not a valid TCPLIB.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the location of a valid TCPLIB.

```
*ERROR* User library cannot be the same as TCPLIB
```

**Cause.** The user specified the user library to be the same file as TCPLIB.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a correct user library file name.

```
*ERROR* Source filename cannot be the same as the object
filename
```

**Cause.** The source file specified with the -src option and the object file specified with the -obj option are the same.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a different name for  the object file name.

```
*ERROR* Source filename cannot be the same as the library
filename
```

**Cause.** The source file specified with the -src option and the library file specified with the -lib option are the same.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify a different name for  the library file name.

```
*ERROR* Invalid pTAL compiler specified : ptal-location.PTAL
```

**Cause.** The pTAL compiler in the specified location is not a valid pTAL compiler.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the location of a valid pTAL compiler.

```
*ERROR* pTAL does not exist in the location specified :
ptal-location
```

**Cause.** The pTAL compiler does not exist in the location specified.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for the pTAL compiler.

```
*ERROR* Illegal location specified for pTAL
```

**Cause.** An invalid volume-subvolume name combination was specified as the location for the pTAL compiler.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for the pTAL compiler.

```
*ERROR* Invalid NLD specified :  nld-location.NLD
```

**Cause.** The `nld` utility in the specified location is not valid.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the location where a valid `nld` utility resides.

```
*ERROR* NLD does not exist in the location specified :
nld-location
```

**Cause.** The `nld` utility does not exist in the location specified.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for the `nld` utility.

```
*ERROR* Illegal location specified for NLD
```

**Cause.** An invalid volume-subvolume name combination was specified as the location for `nld`.

**Effect.** The MAKEUL macro fails.

**Recovery.** Specify the correct location for the `nld` utility.

.

```
*ERROR* Error(s) encountered during pTAL compilation. Please
check the output file for details.
```

**Cause.** Compilation errors were encountered while compiling the pTAL source file.

**Effect.** The MAKEUL macro fails.

**Recovery.** Check the output file and correct the compilation errors.

```
*WARNING* Warnings encountered during pTAL compilation.
Please check the output file for details.
```

**Cause.** Warnings were encountered during pTAL compilation.

**Effect.** If the -lib option was specified, MAKEUL continues and attempts to build the user library file. If the -lib option was not specified, it terminates.

**Recovery.** Check the output file and correct the warnings.

```
*ERROR* Error(s) encountered during building the user
library. Please check the output file for details.
```

**Cause.** The `nld` utility reported errors while building the user library.

**Effect.** The MAKEUL macro fails.

**Recovery.** Check the output file, and correct the errors.

```
*WARNING* Warning(s) encountered during building the user
library. Please check the output file for details.
```

**Cause.** The `nld` utility reported warnings while building the user library.

**Effect.** The MAKEUL macro terminates.

**Recovery.** Check the output file and correct the warnings.

```
*ERROR* File not properly secured for execution : filename
```

**Cause.** The pTAL compiler or `nld` utility in the specified location does not have execution permission for the user running this macro.

**Effect.** The MAKEUL macro fails.

**Recovery.** Secure pTAL or `nld` to give execution permission to the user or change the user to a user who has execution permission for the pTAL compiler and `nld` utility being used.

# Index