Measure User’s Guide

Abstract

This manual describes how to use the Measure performance monitor to collect and examine data, through either a command interface or programmatic interface. This manual is for system operators, system managers, and analysts who balance and tune HP NonStop™ systems.

Product Version

Measure D45 and G12

Supported Release Version Updates (RVUs)

This manual supports D40.00 and all subsequent D-series RVUs and G06.03 and all subsequent RVUs unless otherwise indicated by its replacement publication.

Part Number	Published
520560-003	December 2004

Document History

Part Number	Product Version	Published
425663-001	Measure D45 Measure G08	July 2000
427634-001	Measure D45 Measure G09	April 2001
520560-001	Measure D45 Measure G10	August 2002
520560-002	Measure D45 Measure G11	April 2004
520560-003	Measure D45 Measure G12	December 2004

Measure User's Guide

Index Examples Figures Tables

Examples

Examples

Figures

Tables

What's New in This Manual

Manual Information

Abstract

This manual describes how to use the Measure performance monitor to collect and examine data, through either a command interface or programmatic interface. This manual is for system operators, system managers, and analysts who balance and tune HP NonStop™ systems.

Product Version

Measure D45 and G12

Supported Release Version Updates (RVUs)

This manual supports D40.00 and all subsequent D-series RVUs and G06.03 and all subsequent RVUs unless otherwise indicated by its replacement publication.

Part Number	Published
520560-003	December 2004

Document History

Part Number	Product Version	Published
425663-001	Measure D45 Measure G08	July 2000
427634-001	Measure D45 Measure G09	April 2001
520560-001	Measure D45 Measure G10	August 2002
520560-002	Measure D45 Measure G11	April 2004
520560-003	Measure D45 Measure G12	December 2004

New and Changed Information

This publication has been updated to reflect new product names:

- Since product names are changing over time, this publication might contain both HP and Compaq product names.
- Product names in graphic representations are consistent with the current product interface.

This publication contains these updates for the G11 Measure product version update (PVU):

- Added Measure Support for ANSI SQL Names on page 1-4
- Updated the description of MEASDDLS on page 1-5
- Added the new MEASDDLZ file on page 1-6
- Added the new commands LISTENAME and LISTEXTNAMES to Table 2-1, MEASCOM Commands, on page 2-2
- Added the new MEASCOM report attributes STYLE and DOTS to Table 4-1, MEASCOM REPORT Attributes, on page 4-3
- Added the new command option REPORT STYLE to Table 4-3, Command Option Effects on Data Written to Structured Files, on page 4-31
- Added Loading Data From Different Systems to Common Files on page 4-34
- Added these procedures to Table 6-1, Measure Callable Procedures, on page 6-2:
    - MEAS_ADJUSTZMSRECORD_
    - MEAS_GETDESCINFO_
    - MEASLISTENAME
    - MEASLISTEXTNAMES
    - MEASSQLNAME_COMPARE_
    - MEASSQLNAME_DECOMPOSE_
    - MEASSQLNAME_EDIT_
- Added Appendix E, Converting Existing Applications or Enform Reports to ZMS Style Record Formats

G10 PVU Changes

This publication edition contains these changes for the G10 Measure PVU:

- Added the new entities OSSCPU and OSSNS to:
  - The table in Measurable Resources on page 1-2
  - The table in Entity Types and Specifications on page 3-2
  - Table 6-2, MEASDECS Entity Descriptors, on page 6-4
  - The command descriptions throughout Section 2, Measure Command Interface (MEASCOM)
- Added LISTOSSNAMES, LISTGNAME, LISTPNAME, OSSPATH, and PAGESIZE to Table 2-1, MEASCOM Commands, on page 2-2

G09 PVU Changes

The G09 PVU of Measure provides an integrated view of the HP NonStop Open System Services (OSS) and Guardian environments. The OSS Measure enhancements assist you in the analysis and tuning of your OSS-based application by supporting the capture, display and use of OSS file pathnames in measurement configurations and reports. OSS file pathname support accounts for differences in command and programmatic interfaces, entity counters, identifiers, callable interfaces, error messages, and warning messages. For details on any of these items, see the *Measure Reference Manual.*

Measure G09 Command Interface Changes and Additions

- New commands:
  - LIST OSSNAMES maps Guardian file names or Measure MID values to OSS file pathnames.
  - LISTGNAME translates OSS file pathnames to the corresponding Guardian file name and creation volume sequence number (CRVSN).
  - LISTPNAME translates Guardian file names (gname) and creation volume sequence numbers (CRVSN) to the corresponding OSS file pathname.
  - OSSPATH specifies a default directory for expansion of OSS file pathnames.
  - PAGESIZE assists users in controlling the screen display of OSS file pathnames.
- Changed commands:
  - ENV
  - INFO MEASUREMENT

  - START MEASUREMENT
  - STATUS MEASSUBSYS
  - STATUS MEASUREMENT

Changed Entities

- DISCOPEN
- DISKFILE
- PROCESS
- PROCESSH
- SQLPROC
- SQLSTMT
- USERDEF

Callable Interface Changes and Additions

- New callable interfaces:
  - MEASLISTGNAME translates an OSS file pathname to its Guardian file name equivalent and MID content.
  - MEASLISTOSSNAMES lists OSS file pathname information to the file OSSNAMES.
  - MEASLISTPNAME translates a Guardian file name or an OSS pathid to its OSS file pathname equivalent.
- Changed callable interfaces:
  - MEASMONSTATUS
  - MEASOPEN
  - MEASREAD_DIFF_
  - MEASWRITE_DIFF_
  - MEASINFO
  - MEASREADCONF
  - MEASSTATUS

About This Manual

Introduction

This manual explains how to use the Measure performance monitor to collect and display system performance data.

This manual provides information for:

Server Type	Product Version	Supported RVUs
HP NonStop K-series	Measure D40 and later	D40.00 and later D4*x* RVUs
HP NonStop S-series	Measure G06.03 and later	G06.03 and later G-series RVUs

Who Should Use This Manual

This manual is written for system operators, system managers, and analysts who balance and tune NonStop systems. Readers should be familiar with the HP NonStop operating system and basic hardware and software characteristics of NonStop systems. Knowledge of performance monitoring, analysis, and tuning concepts is also helpful.

What Is in This Manual

Section	Title	This section...
1	Introduction to Measure	Describes the basic features of the Measure performance monitor
2	Measure Command Interface (MEASCOM)	Describes how to use Measure commands
3	Configuring and Running Measurements	Describes how to set up and run measurements and how to view measurement data
4	Formatting Reports and Plots	Describes how to customize Measure reports, generate graphs, and create files that can be used by other applications
5	Defining Custom Counters	Describes how to instrument and measure applications using user-defined counters
6	Creating a Custom Measurement Application	Describes how to use the Measure programmatic interface
7	Balancing and Tuning a System	Describes basic steps for balancing and tuning a system using Measure data

Section	Title	This section...
A	Creating an Enform Report From Measure Data	Describes how to use Measure data with the Enform report generator
B	Examples of RECORD Statements and FIND Queries	Provides examples of Data Definition Language (DDL) RECORD statements and Enform FIND queries
C	Loading Measure Data Into an SQL Table	Describes how to transfer measurement data to an SQL table
D	Example of Measurement Application in C	Provides a sample measurement application in the C programming language

Examples in This Manual

The screen displays in this manual represent various product versions of the Measure software. Your screen displays might differ slightly from the examples shown. Significant differences between product versions are described.

Within examples, the user's command entries are shown in boldface type.

Related Reading

The *Measure Reference Manual* provides detailed information about the Measure commands, data record formats, and callable procedures.

If you are interested in using other NonStop performance products to evaluate and improve system performance, you might be interested in:

For All RVUs

Manual	Describes...
Guardian User's Guide	Daily system maintenance procedures using the Measure performance monitor and other NonStop performance tools.
Introduction to NonStop Operations Management	
ENFORM Reference Manual	How to create a database and generate detailed reports from measurement data.
ENFORM User's Guide	
NonStop SQL/MP Report Writer Guide	How to convert structured Enscribe files to HP NonStop SQL/MP tables and how to produce reports using the report writer. (The *ENSCRIBE Programmer's Guide* describes the Enscribe database record manager.)
TCM Manual	How to use the MeasTCM product to summarize Measure data for capacity-planning studies.
File Utility Program (FUP) Reference Manual	How to use FUP to examine the performance of application files.

Manual	**Describes...**
PEEK Reference Manual	How to use PEEK to produce dynamically updated statistical views of a system.
Guardian Disk and Tape Utilities Reference Manual	How to use DSAP to examine disk activity.
Guardian Programmer’s Guide	Provides programming information for the NonStop operating system.
Guardian Procedure Calls Reference Manual	Syntax and programming considerations for system procedures.
Guardian Procedure Errors and Messages Manual	Error codes, system messages, and trap numbers for system procedures.

For D-Series RVUs

Manual	Describes...
Availability Guide for Performance Management	General principles of performance management on NonStop systems. It also describes a number of performance management tools.
Surveyor User's Guide *Surveyor Reference Manual*	The Surveyor performance database manager, which uses Measure files to create a database of performance information.
ViewPoint Manual	How Measure data can be used in the Pathway environment in displays of network status information provided by the ViewPoint application.
Communications Management Interface (CMI) Operator Reference Manual	How to use CMI to examine the performance of communication lines.
Peripheral Utility Program (PUP) Reference Manual	How to use PUP to examine disk activity.

For G-Series RVUs

Manual	Describes...
SCF Reference Manual for the Storage Subsystem	How to use the Subsystem Control Facility (SCF) to examine disk activity.
Asynchronous Terminals and Printer Processes Configuration and Management Manual *CP6100 Configuration and Management Manual* *EnvoyACP/XF Configuration and Management Manual*	How to use SCF to examine activity on communication lines.

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

This requirement is described under Backup DAM Volumes and Physical Disk Drives on page 3-2.

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual:

Bold Text. Bold text in an example indicates user input entered at the terminal.

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

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *Italic computer type* letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

pathname

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name

INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
LIGHTS [ ON                 ]
       [ OFF                ]
       [ SMOOTH [ num ] ]

K [ X | D ] address-1
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list may be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }

ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

... Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address-1 [ , new-value ]...
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

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 this example, there are no spaces permitted between the period and any other items:

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

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] CONTROLLER

   [ , attribute-spec ]...
```

Notation for Messages

This 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

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
ADD [ MEASUREMENT ] <filename> [, MEASFH <filename> ]
```

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list might be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LBU { X | Y } POWER FAIL
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
P=%p-register E=%e-register
```

1 Introduction to Measure

The Measure performance monitor is a data collection and measurement tool that provides a wide range of performance statistics on system resources. You can use the Measure performance monitor to gather information from systems, network components, and your own business applications.

You then use the Measure data to balance and tune your system, detect problem areas or inefficiencies, balance workloads, evaluate sizing for new applications, or for capacity planning. You can also use the Measure programmatic interface to access a collection of services and build your own high-level performance tools for capacity planning, resource accounting, and load balancing.

Measurable system resources include:

- Processors
- Disks
- Terminals
- Other I/O devices such as tape drives and printers
- I/O controllers, including ServerNet addressable controllers on systems running G-series RVUs
- Communication lines
- Network lines
- Fiber Optic Extension (FOX) cluster traffic on systems running D-series RVUs
- Processes
- Process code ranges
- Logical file opens
- Physical disk file opens
- Optical disks on systems running D-series RVUs
- SQL processes and statements

- HP NonStop Transaction Management Facility (TMF) transactions
- User-defined processes

You select the resources to measure and the time frame for data collection. You can include any combination of measurable resources in a measurement configuration. You can start or stop measurements at any time.

The Measure performance monitor’s predefined counters give a comprehensive picture of each resource’s performance characteristics. In addition, you can set up user-defined counters to measure application-specific performance parameters.

While a measurement runs, you can access and display performance data in system counter space or in the measurement data file. After a measurement ends, you can access and display the performance data in the measurement data file. You can then:

- Display data from active and inactive measurements as basic reports, plots, or bar graphs, depending on the reporting options you select
- Copy the data from a measurement data file to a structured file and create performance databases
- Generate more complex performance reports using the Enform query product or the HP NonStop SQL/MP Report Writer

Continuous Operation and Measurement

The Measure performance monitor’s low overhead makes continuous monitoring feasible. Multiple users can run up to 64 measurements concurrently without starting and stopping the Measure subsystem to do so.

Measurements are taken online in real time. If resources are specified for measurement and become active after a measurement has started, those resources are automatically included in the count. When the same resource is designated in more than one measurement, the data (counter) record for the resource specified in the first measurement is shared by the second and subsequent measurements.

The Measure performance monitor works with the operating system to determine whether processors selected for measurement or being measured are up or down. If a processor being measured halts during a measurement, the Measure monitoring process automatically reactivates measurement activity in that processor when it is reloaded.

Measurable Resources

The resources for which data is collected are called entity types. They include:

CLUSTER	DISCOPEN	OPDISK	SERVERNET	TMF
CONTROLLER	DISKFILE	OSSCPU	SQLPROC	USERDEF

CPU	FILE	OSSNS	SQLSTMT
DEVICE	LINE	PROCESS	SYSTEM
DISC	NETLINE	PROCESSH	TERMINAL

For each entity type you specify for measurement, information is gathered by a set of predefined counters. For example, if you specify a CPU entity type, predefined counters collect information such as CPU busy time, interrupt busy time, number of pages swapped, and so on.

You can measure all resources of a specific type, or you can identify specific resources to be measured. For example, you can gather data on all CPUs of a system or on selected CPUs, such as CPU 0, CPU 4, and CPU 12.

You can specify multiple entity types in the same measurement configuration. For example, you can measure all disks and processes on a system, plus several CPUs, files, and terminals in the same configuration.

You can define your own counters in user applications to set up new performance indicators for your applications, such as number of user transactions, the length of a request queue, or the amount of time spent performing certain functions.

You can specify both predefined and user-defined counters for measurement in the same configuration.

Displays of Measurement Data

The standard data display is a listing of counter values by entity type. You can select the entity type and manipulate the content and format of these displays. For example, you can select a report that shows only totals for each counter of an entity, or you can display both the totals and the individual counter values for an entity on the same report. You can also display actual counter values or select a report that converts actual counts into averages, percentages, and counts per second.

You can also display data in a data file in plot and bar-chart formats. These displays help when you compare counter values for the same or different entities. You can quickly change the type and scale of the graph and the time window of the data.

For further analysis, you can write Measure counter records to structured files, then customize reports of measurement data using the Enform query product or the SQL/MP command interpreter and report writer.

Customized Performance Tools

All Measure control, status, and data access functions are available through a callable interface. The Measure command interpreter, MEASCOM, uses this interface to control the Measure subsystem. You can also use the callable interface to implement custom performance evaluation tools based on Measure data.

Because multiple measurements can run concurrently, you can collect data with your custom tools at the same time you are collecting data using MEASCOM.

The user-defined counters give you a structure to further customize your own products. You can use these counters to measure application events and write the collected data to data files generated by Measure. Selected, continuously monitored events could form the basis of trend analyses for capacity planning or for better system performance management.

Measure Support for Open System Services (OSS)

The Open System Services (OSS) product provides a UNIX-like, standards-based programming and interactive environment on NonStop servers. The OSS run-time environment is present on G-series RVUs and includes the POSIX 1003.1 application programming interfaces (APIs) and the POSIX 1003.2 utilities.

Measure G09 and later PVUs provide an integrated view of the OSS and Guardian environments. The OSS Measure enhancements assist in the analysis and tuning of OSS-based applications by supporting the capture, display, and use of OSS file pathnames in measurement configurations and reports. OSS file pathname support accounts for differences in command and programmatic interfaces, entity counters, identifiers, callable interfaces, error messages, and warning messages. For details, see the *Measure Reference Manual.*

Measure Support for ANSI SQL Names

Measure G11 and later PVUs provide ANSI SQL names in various measurement reports and output. This lets an SQL/MX user see the SQL logical names instead of seeing only the underlying system-generated Guardian file name.

For details, see the *Measure Reference Manual.*

Measure Support for Dynamic-Link Libraries (DLLs)

Measure G12 and later PVUs support measurement of DLLs. For details, see the *Measure Reference Manual.*

Measure Processes and Files

Most Measure processes and files that make up the Measure software are stored in the subvolume $SYSTEM.SYS*nn*.

To find the subvolume name for SYS*nn* on your system, use the TACL STATUS * command. SYS*nn* is the subvolume that contains the OSIMAGE file.

Note. The MEASCTL, MEASFH, and MEASMON processes must be on the active system subvolume (SYS*nn*). Other Measure files can be located on other subvolumes.

MEASCHMA

The MEASCHMA file contains the structure declarations for C and TAL. Its output is similar to the output of the MEASDECS file.

MEASCOM

MEASCOM is the Measure command interpreter process. MEASCOM accepts and processes commands entered at the terminal or executed in a command file. You can control and access multiple measurements with one MEASCOM process.

MEASCTL

MEASCTL is the Measure control process. It performs these functions:

- Allocates and initializes counter records whenever a measurement starts
- Writes counter records to a data file whenever a measurement starts and stops and at user-specified collection intervals
- Deallocates counter records whenever a measurement stops
- Displays snapshots of active counter records for viewing

The MEASMON process creates a MEASCTL process in each CPU of the local system when the subsystem is started.

MEASDDLS

The MEASDDLS file contains the Data Definition Language (DDL) record definitions used for structured files of measurement data, After generating a structured file, you can use the Enform query language or SQL/MP to create custom reports from the data.

In Measure G11 and later PVUs, the DDL run command must include a DICT! directive to create a DDL dictionary.

MEASDDLF

The MEASDDLF file contains DDL record definitions for use by FORTRAN programs.

MEASDDLB

The MEASDDLB file contains DDL record definitions for use by COBOL programs.

MEASDDLZ

In Measure G11 and later PVUs, the MEASDDLZ file contains ZMS-style record templates using legacy-style naming for records and field access. MEASDDLZ is a temporary migration aid for applications moving to use of ZMS-style records.

MEASDECS

The MEASDECS file contains the structure declarations and literal value definitions used by the Measure callable procedures.

MEASFH

MEASFH is the measurement data file-handler process. It performs these functions:

- Creates and initializes data files
- Validates the format of new measurement configurations
- Builds indexes with entries for each record in a measurement data file
- Builds viewable counter records from data in the measurement data file
- Provides information about measurement configurations

Each MEASFH process handles one measurement data file. MEASCOM creates a MEASFH process whenever you start a measurement or examine the contents of a data file.

MEASIMMU

The MEASIMMU file contains the text for Measure error messages and online help displays.

MEASMON

MEASMON is the subsystem monitoring and coordinating process. It performs these functions:

- Creates Measure control processes (MEASCTLs) in each processor of the system
- Sends a copy of each measurement configuration to the MEASCTL processes whenever a measurement starts or a processor is restarted
- Sends control messages to and gathers status information from the MEASCTL processes
- Starts and stops measurements and the Measure subsystem

MEASMON runs as a process pair.

OMEASG

The OMEASG system library file contains the Measure callable procedures used by the operating system. OMEASG is included in the system library by the SYSGEN program.

OMEASP

The OMEASP system library file contains the Measure callable procedures used by applications (system-library procedures) for TNS systems. OMEASP is included in the system library by the SYSGEN program.

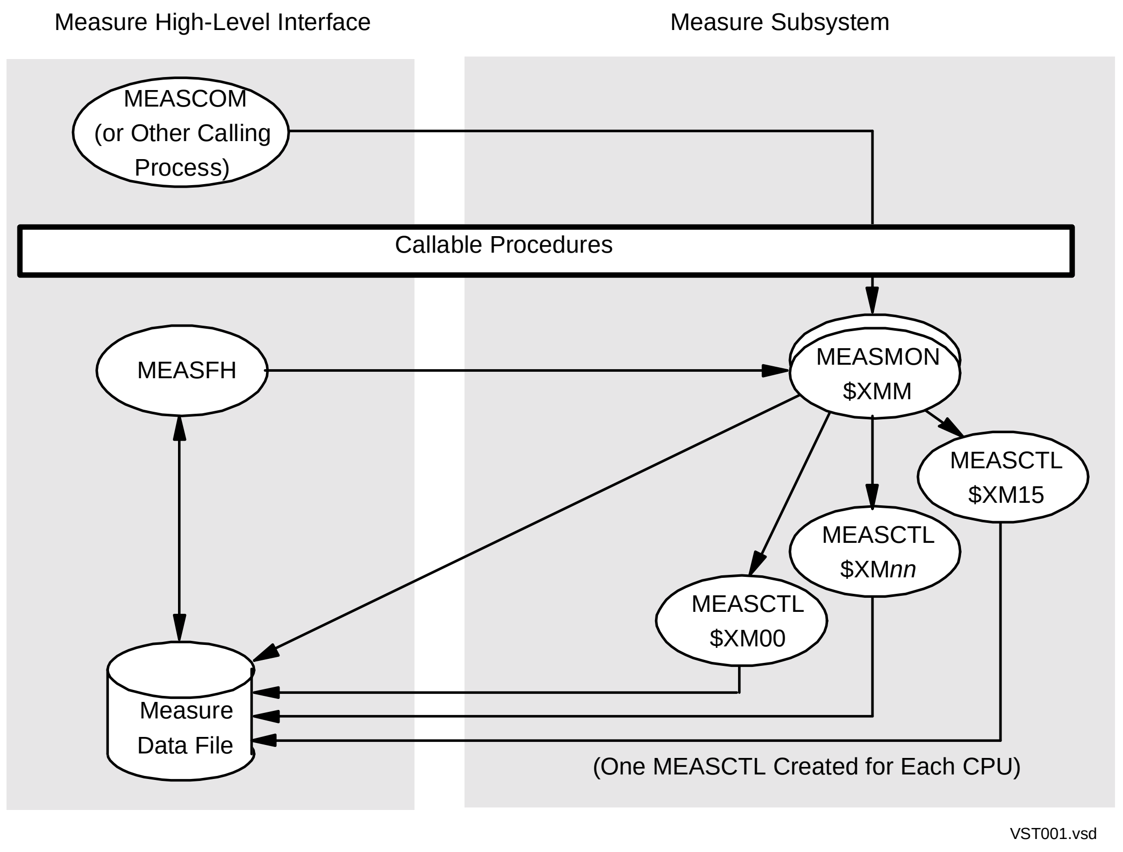
RMEASP

The RMEASP system library file contains the Measure callable procedures used by applications (system-library procedures) for TNS/R systems. RMEASP is included in the system library by the SYSGEN program.

Relationship Between the Measure Components

Figure 1-1 shows the relationship between the Measure components. For a description of how the components operate together to measure system resources, see Section 2, Measure Command Interface (MEASCOM), and Section 3, Configuring and Running Measurements.

Figure 1-1. Measure Subsystem and High-Level Interface



2 Measure Command Interface (MEASCOM)

Use the Measure command interface (MEASCOM) to start and stop the Measure subsystem, start and stop measurements, configure measurements, display and plot measurement data, and generate structured files of measurement data for use by other report-writing products.

For detailed descriptions of each command, object, and attribute, see the *Measure Reference Manual*.

Command-Language Format

MEASCOM uses the NonStop command-language standard format:

```
command  [object]  [attribute]
```

where:

command

is any MEASCOM command.

object

is any MEASCOM command object: COUNTER, MEASSUBSYS, MEASUREMENT, PLOT, REPORT, or a Measure entity type.

attribute

is one or more attributes for *object*.

Table 2-1 summarizes all MEASCOM commands. Most of these commands are described, with examples of their use, in Sections 3 through 5 of this manual. For complete descriptions of all commands, see the *Measure Reference Manual*.

Table 2-1. MEASCOM Commands (page 1 of 3)

Command	**Object**	**Function**
Controlling the Subsystem		
START	MEASSUBSYS	Starts the Measure subsystem.
STATUS	MEASSUBSYS	Displays information about the Measure subsystem.
STOP	MEASSUBSYS	Stops the Measure subsystem.
Configuring Measurements		
ADD	COUNTER	Specifies a user-defined counter for measurement.
ADD	*entity-type*	Selects an entity for measurement.
DELETE	COUNTER	Deletes a user-defined counter from a measurement configuration.
DELETE	*entity-type*	Deletes an entity from a measurement configuration.
INFO	COUNTER	Displays information about user-defined counters.
INFO	*entity-type*	Displays entities selected for measurement.
Controlling Measurements		
START	MEASUREMENT	Starts a measurement.
STATUS	MEASUREMENT	Displays information about a measurement.
STOP	MEASUREMENT	Stops a measurement.
Selecting and Displaying Data		
ADD	MEASUREMENT	Selects a data file for examination.
DELETE	MEASUREMENT	Deletes a data file from those selected for examination.
INFO	MEASUREMENT	Displays information about a data file.
LIST	*entity-type*	Displays a report on a selected entity.
LISTACTIVE	*entity-type*	Displays a report on an entity being measured.
LISTALL	*entity-type*	Displays a report on an entity by interval records.
RESET	REPORT	Returns format options for data displays to default values.
SET	REPORT	Sets format options for data displays.
SHOW	REPORT	Displays format options for data displays.
Plotting Data		
ADD	PLOT	Selects a counter for plotting.
DELETE	PLOT	Deletes a counter from those selected for plotting.

Table 2-1. MEASCOM Commands (page 2 of 3)

Command	Object	Function
Plotting Data (continued)		
INFO	PLOT	Displays information about the counters selected for plotting.
LIST	PLOT	Displays a plot of selected counters.
RESET	PLOT	Returns plot format options to their default values.
SET	PLOT	Sets plot format options.
SHOW	PLOT	Displays plot format options.
ANSI SQL Name Interaction		
LISTENAME		Translates a Guardian file name and CRVSN into its corresponding external name (ANSI SQL name or OSS pathname).
LISTEXTNAMES		Creates or appends to a key-sequenced file of records for mapping Measure OSS PATHID values to OSS pathnames and Guardian file names or MIDs to ANSI SQL names and Guardian file names.
OSS File Pathname Interaction		
LISTGNAME		Translates an OSS file pathname to its corresponding Guardian file name and creation volume sequence number (CRVSN).
LISTOSSNAMES		Maps Guardian file names or Measure ID values to OSS file pathnames.
LISTPNAME		Translates a Guardian file name (gname) and creation volume sequence number (CRVSN) to its corresponding OSS file pathname.
OSSPATH		Specifies a default directory for expansion of OSS file pathnames.
PAGESIZE		Assists you in controlling the screen display of OSS file pathnames.
Simplifying Command Entry		
ASSUME		Sets the command object for succeeding commands. The initial ASSUME default is the MEASUREMENT command object.
FC		Displays a previous command for editing and reexecution.
HELP		Displays information on commands, error messages, and counters.
HISTORY		Displays a history of entered commands. The default is 10 lines.

Table 2-1. MEASCOM Commands (page 3 of 3)

Command	Object	Function
OBEY		Executes any file containing valid MEASCOM commands.
RUN		Runs another process from within a MEASCOM session.
SYSTEM		Sets default system.
VOLUME		Sets default volume and subvolume.
!		Retrieves and immediately reexecutes previously entered MEASCOM commands.
Tracking MEASCOM Sessions		
ENV		Displays environment parameter settings.
EXIT		Ends MEASCOM session.
LOG		Starts and stops logging activity for a session.
OUT		Directs Measure command output to a file.
SETPROMPT		Displays current environmental information, such as the assumed object, along with the MEASCOM prompt.
SWAPVOL		Specifies swap volume for MEASFH work files.
TIME		Displays the local civil time of a system.
Suppressing MEASCOM Messages		
COMMENTS		Specifies comments to be suppressed or displayed.
WARNINGS		Specifies warnings to be suppressed or displayed.

Entering Commands

- Enter commands only at a Measure prompt (a single plus sign, +).
- To enter multiple commands on the same line, end each command with a semicolon (;).
- To continue a command over several lines, place an ampersand (&) at the end of each line to be continued.
- To delimit comments on a command line, begin the comment with double hyphens (--). The comment ends either at the first occurrence of another double hyphen or semicolon, or the end of the line.

Note. Hyphens can be used in user-defined names. Therefore, use a blank to separate the dashes from the text of the comment to avoid possible command errors.

Using Abbreviations in Commands

When you enter MEASCOM commands interactively (at the + prompt), you can abbreviate the keywords for commands, objects, attributes, and counter names. For example, instead of entering STATUS MEASUREMENT, you can enter STAT MEASU.

- MEASCOM matches abbreviations word by word and gives priority to exact matches. For example, LIST is always recognized as the LIST command and not as an abbreviation of LISTALL.

  Note. MEASCOM recognizes DISK as an alternate spelling for the keyword DISC. You cannot use DISK as an abbreviation for DISKFILE.

- For commands, objects, and attributes, MEASCOM compares each abbreviation against all other commands, objects, and attributes. Each abbreviation you use must be unique among all commands, objects, and attributes. For example, RE is not an abbreviation for the RESET command because it also matches the REPORT object.
- For counter names, MEASCOM compares abbreviations only against other counter names for the specified entity type. For example, the DISC entity has counters named REQUEST-QTIME and REQUEST-QLEN-MAX. You could abbreviate these names as REQUEST-QT and REQUEST-QL. Hyphens are treated as part of the name, not as spaces. You cannot abbreviate REQUEST-QTIME as REQ-QT.
- As new keywords are added to MEASCOM, you might need to modify the abbreviations you use so that they remain unique.
- You cannot use abbreviations in noninteractive command entry (that is, when you execute a MEASCOM command from the TACL prompt).
- You cannot use abbreviations in a command (OBEY) file or in an input file used during noninteractive command entry. Because abbreviations can change from RVU to RVU, this prevents command files from becoming obsolete.
- You cannot abbreviate the topic in a HELP command. For example, you must enter HELP ADD TERMINAL, not HELP ADD TER.

Starting and Stopping the Measure Subsystem

The Measure subsystem must be installed and operating before you can use the Measure command interpreter or callable procedures to configure and run measurements and display measurement data in active data files or counter space. To display only data in inactive data files, you do not need to start the subsystem. Only a super-group user (255, *n*) can start or stop the Measure subsystem.

Starting the subsystem creates the monitoring process (MEASMON) and its backup process. MEASMON then immediately creates a counter-record control process (MEASCTL) in each CPU of the local system.

To start the subsystem:

```
35> MEASCOM
MEASURE Performance Monitor - T9086G10 - (16DEC03) - \HATI
(C)1986 Tandem (C)2003 Hewlett Packard Development Company, L.P.
1+ START MEASSUBSYS
```

You can also start the subsystem from the command interpreter prompt without starting MEASCOM interactively:

```
36> MEASCOM START MEASSUBSYS
```

Note. If you start the subsystem from a Multilan window session, the Multilan Resource Manager (MLRM) stops the MEASMON process when the window session terminates. The MEASCTL processes continue to run. To stop them, reissue a START MEASSUBSYS command from a Multilan session, then issue a STOP MEASSUBSYS command.

If a CPU fails while the subsystem is running, MEASMON automatically re-creates and restarts a MEASCTL control process in that CPU after the CPU is reloaded. The re-created control process receives information on all currently active measurements and starts data collection.

The data file for any measurement that spans the CPU's failure and reload contains at least one record for each measured entity in that CPU: the one from the CPU restart to the end of the measurement. If a collection interval was specified for the measurement, the data file contains records at each specified interval up to the time of CPU failure (except for records lost from the write buffer) and at each interval after the CPU reload.

To stop the Measure subsystem:

```
6+ STOP MEASSUBSYS
```

Stopping the Measure subsystem aborts all currently active measurements. Closing records are not issued, so data files might be incomplete.

▲ **WARNING.** Do not bring down MEASCTL's swap volume while Measure is still running. Doing so could cause all CPUs to halt.

To list the currently active measurements before stopping the Measure subsystem:

```
43> MEASCOM
MEASURE Performance Monitor - T9086G10 - (16DEC03) - \HATI
(C)1986 Tandem (C)2003 Hewlett Packard Development Company, L.P.
1+ STATUS MEASSUBSYS
Number of Active (or Configured) Measurements =  0
2+ STOP MEASSUBSYS
```

If you enter the STOP MEASSUBSYS command from an interactive MEASCOM session and measurements are active, this prompt appears:

```
n measurement(s) still active; still stop the subsystem (y/n)?
```

A YES response aborts all active measurements before the subsystem is stopped. If transient entities do not exist or the measurement was run without a collection interval, the records are lost. Closing records are not written when a measurement is aborted. A NO response cancels the STOP MEASSUBSYS command.

Starting and Stopping MEASCOM

Use the TACL RUN command to start the MEASCOM command interpreter process.

```
14> [ RUN $disk.subvol ] MEASCOM
```

where:

`RUN $disk.subvol`

is required only if MEASCOM is not located in $SYSTEM.SYSTEM or $SYSTEM.SYS*nn*.

To stop a MEASCOM session:

```
10+ EXIT
4>
```

Redirecting Command Output

You can redirect command output to save a Measure report, plot, or INFO command display in a file:

- To redirect the output from one MEASCOM command, specify the OUT *filename* option immediately after the command, on the same line and before any arguments. For example:

```
4+ LIST /OUT $PERF.DATA.MYVOL/ CPU 1, FROM 12:30, TO 12:45
```

- To redirect the output of multiple MEASCOM commands, enter the OUT *filename* command. The file named in this command will contain the prompts and MEASCOM commands as well as the command output. For example:

```
6+ OUT $PERF.DATA.MYVOL
```

The output files created by both the OUT option and the OUT command can be edited, printed, included in reports, or used as command (OBEY) files.

This example creates a command file named DAILY. The command file can be used, with minimal editing, to configure a measurement.

```
7+ ADD MEASUREMENT NOV5
8+ INFO MEASUREMENT /OUT DAILY/ NOV5
9+ EXIT
45> FUP COPY DAILY
Add measurement $PERF.DATA.NOV5 -- Current Data File --
From  5 Nov 2003, 10::36,  To  5 Nov 2003, 13:49:27
Cpu                     12 Entities             564 Words
Process               1325 Entities           95400 Words
File                   180 Entities            6120 Words
Discopen               161 Entities            2415 Words
-- Add Cpu *
-- Add Process *
-- Add File $SYSTEM.SYSTEM.*
-- Add Discopen $SPOOL.*.*

11 RECORDS TRANSFERRED
46>
```

You can also save a copy of all or part of a MEASCOM session by using the LOG TO and LOG STOP commands.

The LOG TO command causes MEASCOM to write a copy of all succeeding prompts, command lines, and command output to the specified file. The LOG STOP command causes MEASCOM to stop writing to the log file and close it. MEASCOM automatically closes the log file at the end of a session.

Creating a Custom Startup File

You can create a file of MEASCOM commands to be executed similarly to a command (OBEY) file each time MEASCOM is invoked. The file must be an EDIT file named MEASCSTM that contains only MEASCOM commands.

You can set up multiple MEASCSTM files to perform different measurements. Each MEASCSTM file must be in a different subvolume. MEASCOM first looks for the MEASCSTM file in the current subvolume. If the file is not found, MEASCOM looks in your default subvolume.

This example shows a MEASCSTM file:

```
COMMENTS SUPPRESS 2000
WARNINGS SUPPRESS 3014
SETPROMPT VOLUME
SET REPORT RATE OFF
SET REPORT LOADID TEST
```

Accessing Online Help

During an interactive MEASCOM session, you can access online help text for the various MEASCOM elements. Enter the HELP command to display help text.

The help text for some topics is more than one screen long. For these topics, MEASCOM uses a double prompt (++) at the end of the first screenful of text to indicate more information. To view additional help text, press RETURN. To return to command entry, press CTRL Y or type the word BREAK at the double prompt.

For a list of all MEASCOM keywords (commands, objects, and attributes) as well as guidance to more specific help topics, enter HELP ALL.

- To display the cause, result, and recovery action to be taken for any error message, enter HELP and the error number. For example:

```
31+ HELP 3055
Cause:    The parser encountered a syntax error
Result:   The requested operation is ignored
Recovery: Correct the syntax according to HELP or the MEASURE
          Reference Manual
```

- To get information about how to use a Measure command keyword, enter HELP and the command. For example:

```
32+ HELP ADD
    The ADD command can operate on three objects: 1) entities
(such as CPU, Process, etc.), 2) a measurement datafile, and
3) the plot list.

    Before a measurement can be taken, the measurement
configuration must be built.  To do this, one or more ADD
entity commands are executed.  For example, ADD CPU *; ADD
PROCESS $SYSTEM.SYSTEM.*, would cause all CPUs and any
```

```
process whose programfile was in $SYSTEM.SYSTEM to be
measured.  To view the measurement configuration for each
entity type use the INFO entity command.  Once the
configuration is complete, a START MEASUREMENT command is
executed.

    To view the results of the measurement the ADD
MEASUREMENT command is used (the START and STOP MEASUREMENT
commands perform an implicit ADD MEASUREMENT).  This
operation starts a MEASURE file handler process (MEASFH)
which is used to retrieve the data stored in the file.  To
view the data, use the LIST or LISTALL entity commands.

    After a LIST command, the ADD PLOT command can be used to
add counters to the current plot list.  For each ADD PLOT,
one or more entities are added to the list, along with the
counter of interest.  To display the PLOT, use the LIST PLOT
command.

   For specific command syntax perform:
++
    HELP ADD ENTITY        (for general syntax)
    HELP ADD <entity type> (e.g.HELP ADD CPU, for specific
syntax)
    HELP ADD MEASUREMENT
    HELP ADD PLOT
```

- To get command syntax information, enter HELP, the command, and the object:

```
33+ HELP ADD MEASUREMENT
ADD [ MEASUREMENT ] < filename> [, MEASFH <filename> ]

If either <filename> is a logical Define name, the define
must exist, and map to a permanent disk filename.
```

- If the command object is an entity type, you can get both general syntax information and entity-specific syntax information. To get general syntax, enter HELP, the command, and ENTITY. For example:

```
34+ HELP LIST ENTITY

LIST [ <entity type> ] <entity spec> [, <list spec> ] ...

 where <list spec> is:
  { IF <item name> <relop> <value>                    }
  { BY <item name> [ ( ASCENDING | DESCENDING ) ] }
  { FROM <time>                                       }
  { FOR <time interval>                               }
  { TO <time>                                         }
  { LOADID <name8>                                    }
  { TOLERANCE { ON | OFF }                      } defaults to ON
  { <report attribute>                                }

    <relop> is { < | > | = | <> }
    <value> is { 0:2147482.999  }
    <name8> is { 1-8 character name, first character must be
```

```
                 alphabetic,
}
               { remaining characters can be digits,
                 hyphens (-),
}
               { underscores (_), or carets (^).
}

See HELP ADD <entity type> for <entity spec>, except for
     SQLSTMT, see HELP LIST SQLSTMT.
See HELP ADD PLOT for <item name>.
See HELP REPORT for <report attribute>, <time>, and <time
     interval>.
```

- To get syntax for a specific entity type, enter HELP ADD and the entity type:

```
35+ HELP ADD DISKFILE
ADD [ DISKFILE ] <diskfile spec> [, <diskfile spec> ] ...

  where <diskfile spec> is one of the following:

    { *                }

    { <disk fname set> }--a permanent or temporary disk
                          file name (or file name set), or a
                          logical Define name that maps to
                          such a file name. MEASCOM expands
                          partially entered names using the
                          current SYSTEM and VOLUME defaults.

Any file name entered or expanded must describe a local disk
file name. Only local disk files can be measured.
```

- To get information about an object's attributes, type HELP and the object name:

```
36+ HELP PLOT
VERT-BASE    { ON | OFF }           -- defaults to ON
TIME-BASE    [ ON | OFF }           -- defaults to unspecified
SCALE-FROM   { 0:999999999999.999 } -- defaults to   0
SCALE-TO     { 0:999999999999.999 } -- defaults to 100
WIDE-ITEM    { ON | OFF }           -- defaults to OFF
FROM         { <item> }             -- defaults to unspecified
TO           { <item> }             -- defaults to unspecified
FOR          { <item> }             -- defaults to unspecified

 <time>          { [ <date>, ] <time of day>
}
 <date>         { <dd> <mmm> [<yyyy>] | <mmm> <dd> [<yyyy>]
}
 <time of day>   { <hour> : <minute> [ :<second>]
}
 <time interval> {{ 1:9999 } {SECOND[S]| MINUTE[S]| HOUR[S]}}

PLOTs can be produced with rated or unrated values. To do
```

```
this, change the RATE attribute of the REPORT object. For
example:

SET REPORT RATE ON;  LIST PLOT -- generates a plot with rated
                                    values

SET REPORT RATE OFF; LIST PLOT -- generates a plot with
                                   unrated values
```

- To view possible counter names for an entity type, type HELP, the entity type, and COUNTERS. For example:

```
6+ help diskfile counters
block-splits         extent-allocations   max-lockwait-time
cache-read-hits      fcb-number+          open-qlen-max
cache-write-cleans   file-code+           open-qtime
cache-write-hits     file-name*           os-version+
cpu-num+             file-type+           requests
delta-time+          from-timestamp+      requests-blocked
driver-input-calls   loadid*              starting-eof
driver-output-calls  lock-bounces         system-name*
ending-eof           lock-timeouts        to-timestamp+
error+               lockwait-time        transient-opens

+ -- Item can only be used in the IF and BY clauses (i.e. it
     cannot be used in a PLOT command)
* -- Item can only be used in the BY clause
```

- To view counter types and descriptions and their appearance in different reports, type HELP and the counter name. For example:

```
38+ HELP READS
File Report:
  Calls to READ, READUPDATE, or READUPDATELOCK.
     Counter type: Incrementing

Disc and Opdisk Reports
  Physical disk reads.  Counter type: Incrementing

Device, Line, Netline, and Terminal Reports
  Physical reads.  Counter type: Incrementing.
```

- To view a list of MEASCOM keywords, type HELP ABBREVIATIONS. This list helps you determine the number of characters needed to make an abbreviation unique.

Modifying D-Series Command Files for Systems Running G-Series RVUs

After you migrate from a D-series RVU to a G-series RVU, you might have to change your Measure command (OBEY) files.

In command files written for D-series RVUs, you use channel, controller, and unit numbers to identify specific storage devices (DEVICE, DISC) and communication devices (LINE, NETLINE, TERMINAL). In G-series RVUs, instead of channel, controller, and unit numbers, you must specify group, module, and slot numbers for storage devices, or track ID, CLIP, and line numbers for communication devices. Therefore, you must change any command file that specifies channel, controller, or unit.

Note. OPDISK entities are also identified by channel, controller, and unit numbers. Optical disks are currently supported only in D-series RVUs.

You might not have to change a command file that specifies all devices of a particular type (such as ADD DISC *) or that specifies only a CPU number or device name. Assuming that the CPU and device exist on the G-series RVU, the command file runs without modification.

The CONTROLLER entity type does not exist in G-series PVUs. It is replaced by the SERVERNET entity type, which measures ServerNet addressable controllers (SACs). However, you can use a command file that specifies all controllers (ADD CONTROLLER *) to measure all SACs on a G-series RVU. To measure a specific SAC, you must use the SERVERNET entity type and specify the SAC identifiers.

Similar considerations apply to custom measurement applications migrated from a D-series RVU to a G-series RVU. To modify applications to measure G-series entities, see Modifying D-Series Applications for G-Series Systems on page 6-23.

Accessing D-Series Measurement Files From a System Running a G-Series RVU

You can access a measurement data file created by a D-series PVU of the Measure performance monitor from a system running a G-series RVU. However, you must use a D-series PVU of MEASCOM. To ensure compatibility with all D-series RVUs, always use the most recent D-series MEASCOM.

In all Measure PVUs, the MEASFH version and the data file version must be the same. You must use D30 MEASFH to access a D30 data file, G02 MEASFH to access a G02 data file, and so on.

3 Configuring and Running Measurements

Taking a measurement involves selecting the resources to be measured, creating a measurement configuration for those resources, and starting the measurement. While a measurement runs, you can check its status and display data in the measurement data file or in the active counters associated with the measurement configuration. After it stops, you can display or plot data from the measurement data file.

The Measurement Configuration

The measurement configuration identifies the system resources you want to measure.

Measurement configurations vary depending on the kind of performance information you need. For example:

- A configuration useful for collecting data for performance analyses and capacity planning projects might include all CPUs, disks, devices, communication lines, Fiber Optic Extension (FOX) messages, and Expand traffic, as well as major application processes on a system.
- A configuration aimed at monitoring and balancing the performance of processes on a system might include all CPUs and all major processes.
- A configuration aimed at debugging or fine-tuning process code might include process code-ranges for all the processes on a system that are potential problems.

A typical measurement configuration usually includes all CPUs, processes, and disks, and selected disk opens and files on a system.

Note. When you start a measurement using the START MEASUREMENT command, you specify other information needed for a measurement, such as the starting and ending times, the name of the data file associated with the configuration, and the intervals at which data is written to the data file.

Entity Types and Specifications

The measurement configuration describes each resource to be measured in terms of its entity type and entity specification.

This list describes each entity type:

CLUSTER	Measures the number of FOX messages sent and received by all processes on the local server. Currently supported only in D-series PVUs.
CONTROLLER	Measures I/O activity on disk, tape, terminal, and other device controllers on NonStop K-series servers.
CPU	Measures one or more CPUs on the local server.
DEVICE	In D-series RVUs, measures all devices on the local server except disks, communication lines, subdevices, and asynchronous terminals. In G-series RVUs, measures tape devices.
DISC	Measures one or more disks on the local server.
DISCOPEN	Measures the I/O operations performed by the disk process on a specified file (physical file access). Reports measurements separately for each opener process.
DISKFILE	Measures the I/O operations performed by the disk process on a specified file (physical file access). Reports measurements collectively for the whole file.
FILE	Measures the I/O operations performed by a user process on an explicitly opened file (logical file access).
LINE	Measures communication lines.
NETLINE	Measures network communication lines.
OPDISK	Measures optical disks on the local server. Currently supported only in D-series PVUs.
OSSCPU	OSS elements in each system processor.
OSSNS	OSS Name Server processes.
PROCESS	Measures system and user processes.
PROCESSH	Measures how often a program executes specified code ranges.
SERVERNET	Measures I/O activity on ServerNet addressable controllers (NonStop S-series servers only). Measures interprocessor communication (IPC). For HP NonStop ServerNet Cluster (Measure G08 and later), measures remote interprocessor communication (RIPC).
SQLPROC	Measures SQL/MP processes.
SQLSTMT	Measures statements within an SQL/MP process.
SYSTEM	Measures network traffic through Expand line handlers.

TERMINAL	Measures terminal and subdevice I/O.
TMF	Measures TMF transactions.
USERDEF	Measures activity in a user process that has been instrumented to maintain user-defined counters.

An entity specification describes a particular resource within an entity type. For example, the specification for a CPU entity is a CPU number (0, 1, 2, and so on). The specification for a DISC entity is a disk name ($DATA, $INFO, $BUYER, and so on). The specification for a CLUSTER entity is a system name or a system name and CPU number. You can measure specific resources or all resources of a particular type.

For entity specifications for each type of resource, see the *Measure Reference Manual*.

Creating the Configuration

To create a measurement configuration, use the ADD *`entity-type`* command to specify the entities to be measured. For example, to build a configuration to measure all CPUs, processes, and disks on a system plus all disk opens on $MYVOL:

```
4+ ADD CPU *
5+ ADD PROCESS *
6+ ADD DISC *
7+ ADD DISCOPEN $MYVOL.*.*
```

To exclude specific entities from a group of entities to be measured, use the DELETE *`entity-type`* command. For example, to exclude CPU 0 from the measurement configuration:

```
8+ DELETE CPU 0
```

Also use the DELETE *`entity-type`* command to delete an ADD command from an existing configuration. For example, to delete the ADD DISC * command:

```
9+ DELETE DISC *
```

To view the entities specified in a configuration, use the INFO *`entity-type`* command. For example:

```
10+ INFO DISCOPEN
Add Discopen $MYVOL.*.*
11+ INFO CPU
Add Cpu *
Delete Cpu 0
```

To reinclude an entity in a configuration, use the ADD *`entity-type`* command. For example, to reinclude CPU 0 in the previously defined configuration:

```
12+ ADD CPU 0
13+ INFO CPU
Add CPU *
```

If you exit MEASCOM before starting the measurement, the configuration is not saved.

Running a Measurement

1. By default, MEASCOM starts a measurement immediately after you issue the START MEASUREMENT command and ends it when you enter the STOP MEASUREMENT command. However, you can specify starting and ending times for the measurement in the START MEASUREMENT command.
2. When you start a measurement, MEASCOM creates a file-handling process (MEASFH), converts your measurement configuration to an internal format, then passes the configuration to MEASFH.
3. MEASFH can perform additional processing on the records before passing the configuration to MEASMON. When MEASMON receives the configuration, it passes a copy of it to each MEASCTL process in each CPU of the system.
4. The MEASCTL processes allocate and initialize the counter records for each entity in the measurement configuration. If an entity is already being measured, the MEASCTL process writes the current counter values for that entity to the data file as the initial counter values. It does not allocate and initialize new counters for the entity. If an entity is not already being measured, its initial value is zero.
5. As the measurement continues running, the NonStop operating system, file system, SQL executor, disk processes, communications processes, and TMF update (bump) the counters.
6. For every entity being measured, the MEASCTL process writes counter information to the data file at least twice: when the measurement starts and when it ends. If you specify a collection interval in the START MEASUREMENT command, the MEASCTL process also writes counter values to the data file at the interval you specify.

   A collection interval lets you see cumulative counter values from specified intervals within a measurement. A collection interval is particularly useful to view data while a measurement is running or to analyze part of a measurement. However, a collection interval uses more system resources during a measurement. For more information, see Specifying a Collection Interval on page 3-6.
7. The data file is made up of variable-length records that hold information about each resource and the value of each counter associated with each resource. The MEASFH process creates an index on the data file with pointers to the records in it and then sorts the index. The larger the data file, the longer it takes MEASFH to scan the records and sort the index.

Predefined Counters

For each entity type you specify in a measurement configuration, the counters predefined for that entity are used to collect information during the measurement.

For example, when you specify the CPU entity in a configuration, all predefined CPU counters are activated when the measurement starts: the CPU-BUSY-TIME counter collects information about the amount of time the CPU was busy, the INTR-BUSY-TIME counter records the amount of time the CPU spent handling interrupts, the SWAPS counter records the number of pages the CPU swapped, and so on.

Similarly, when you specify the DISC entity in a configuration, all predefined DISC counters are activated when the measurement starts. The REQUESTS counter records the number of requests the disk received, the READ-BUSY-TIME and WRITE-BUSY-TIME counters determine the amount of time the disk spent reading and writing data, and so on.

Each counter is one of these types:

Accumulating	Totals the number of bytes input or output to a file, disk, or other entity.
Busy	Measures the time during which a resource is busy.
Elapsed	Measures the time required for an event to take place, start to finish.
Fixed Accumulating	Totals the number of bytes input or output to a file, disk, or other entity. This is a 64-bit accumulating counter type.
Incrementing	Counts the number of times a particular event occurs (for example, swaps and disk I/Os).
Max value	Tracks the maximum value of a counter.
Max queue	Tracks the maximum length of a queue.
Queue	Measures the amount of time elements spent on a queue.
Queue-busy	Measures the time during which there are elements on a queue.
Response time	Tracks the time between a read of a terminal and the following write to the same terminal.
Sampling	Finds the approximate busy time for procedure code ranges.
Snapshot	Records a value taken at a specific time, such as at the start of a measurement.
Transaction	Increments each time a response-time counter is bumped.

For descriptions of the types of counters predefined for each entity type, the kind of information each collects, the record structure for each entity, and descriptions of each field in the record, see the *Measure Reference Manual*.

Starting a Measurement

To specify a data file name and start the measurement, use the START MEASUREMENT command.

Typically, you specify a nonexistent data file, and MEASFH creates an unstructured file (code 175). If you specify the name of a data file that already exists, MEASFH purges the contents of that file so it can receive measurement data.

To start a measurement that collects data in the file named $PERF.DATA.NOV5:

```
16+ START MEASUREMENT $PERF.DATA.NOV5
```

Specifying a Start and Stop Time

Use the TO, FROM, and FOR clauses of the START MEASUREMENT command to direct MEASCOM to start or stop the measurement at a specific time. The TO and FOR clauses cannot be used together. For detailed descriptions of the FROM, TO, and FOR clauses, see the *Measure Reference Manual*.

FROM *date/time*	Specifies the date and time to start the measurement.
TO *date/time*	Specifies the date and time to end the measurement.
FOR *duration*	Specifies how long the measurement will run. *duration* is a whole number followed by HOURS, MINUTES, or SECONDS. The FOR clause can be used by itself or with the FROM clause.

These clauses let you collect data at particular times of the day or run regularly scheduled performance checks. For example, you might take a standard set of measurements each day to monitor system performance at peak and off-peak hours.

To start a measurement that runs from 9:00 a.m. to 5:00 p.m.:

```
4+ START MEASUREMENT $PERF.DATA.NOV5, FROM 9:00, TO 17:00
```

To specify the same timeframe using the FROM and FOR clauses:

```
4+ START MEASUREMENT $PERF.DATA.NOV5, FROM 9:00, FOR 8 HOURS
```

For example, to enter MEASCOM, build a measurement configuration to collect information about all CPUs and two disks ($DATA and $BUYER), start the measurement, directing the data to a file named $DATA.PERF.NOV05, and exit from MEASCOM:

```
45> MEASCOM
MEASURE Performance Monitor - T9086G10 - (16DEC03) - \HATI
(C)1986 Tandem (C)2003 Hewlett Packard Development Company, L.P.
1+ ADD CPU *
2+ ADD DISC $DATA
3+ ADD DISC $BUYER
4+ START MEASUREMENT $DATA.PERF.NOV05
5+ EXIT
```

This measurement starts as soon as the START command is issued and runs until you issue the STOP MEASUREMENT command.

Specifying a Collection Interval

A collection interval records measurement data at specified intervals within a measurement. A collection interval can help you identify peak periods within a measurement or analyze interactions between measured resources.

This command starts a measurement at 9:00 a.m., ends the measurement at 5:00 p.m., and specifies a 30-minute collection interval:

```
4+ START MEASUREMENT $PERF.DATA.NOV5, FROM 9:00, TO 17:00,&
4& INTERVAL 30 MINUTES
```

Using a collection interval does not cause counters to be reset during a measurement. Counter values are always accumulated for the entire measurement.

Use a collection interval only when you need a detailed view of system performance. A collection interval can increase Measure overhead and the size of the data file. The shorter the interval and the longer the measurement, the higher the overhead.

Note. On systems running G-series RVUs, some data files might be smaller than the files for comparable measurements on systems running D-series RVUs. The difference is due to a change in how MEASCTL writes data records when there is a collection interval. The difference is most noticeable in large measurements, especially those measuring all FILE and DISCOPEN entities.

Stopping a Measurement

When a measurement stops:

1. MEASMON notifies all active MEASCTL processes that the command was issued.
2. The MEASCTL processes write ending records to the data file, deallocate the counter records, perform some cleanup work, and notify MEASMON.
3. MEASMON writes wrap-up information in the data file and breaks the connection.

If you use the FROM-FOR, FROM-TO, or FOR clauses in the START MEASUREMENT command, the measurement stops automatically at the specified time.

If you have not used these clauses or if you choose to stop a measurement before the specified time, you must use the STOP MEASUREMENT command. To use this command, you must have the same user ID as the process that started the measurement or be a super-group user (255,*n*).

You must also name the data file associated with the measurement configuration in the STOP command:

```
4+ STOP MEASUREMENT $DATA.PERF.NOV05
```

When a measurement is stopped using this command, an ADD MEASUREMENT operation is performed automatically on the data file so you can access it. To stop a measurement without accessing its data file, use the NO ADD option of the STOP MEASUREMENT command. For example:

```
4+ STOP MEASUREMENT $DATA.PERF.NOV05, NO ADD
5+ EXIT
10>
```

Checking Measurement Activity and Data Files

To find out how many measurements are active on or configured for a system, use the STATUS MEASSUBSYS, STATUS MEASUREMENT, and INFO MEASUREMENT commands.

Active includes not only measurements that are running, but also any measurements whose data files have filled up but that have not been stopped. Configured includes both measurements for which a START MEASUREMENT command has been issued but whose start time has not yet been reached and measurements that have aborted. For more information, see Potential Data File Errors on page 3-10.

Note. Starting with G08 Measure, the data file size increased from 127.5 MB to 1 GB.

Checking Subsystem Status

The STATUS MEASSUBSYS command shows the number of measurements active or configured on a system and lists their data file names. It also shows the total number of MEASCTL processes active or to be activated for all the measurements listed and their associated CPUs. For example:

```
20+ STATUS MEASSUBSYS
Number of Active (or Configured) Measurements = 2
  $DATA.PERF.JUL01
  $DATA.PERF.JUL02

Number of Active MEASCTL Processes = 16
  in CPU(s): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15
```

Checking Data File Size and Content

To check the contents and size of active or configured measurement data files, enter the STATUS MEASUREMENT command. To display information with this command, you must be a member of the super-user group or be the same user ID that started the measurement. Otherwise, when you enter a STATUS MEASUREMENT command, this message appears:

```
MEAS 3302 The requestor does not have proper security rights
          to perform this action; check the request from
          Measstatus
```

For a configured measurement, this command shows the starting time, the ending time (if specified), and the current and maximum size of the data file. For example:

```
28+ STATUS MEASUREMENT $DATA.PERF.JUL01
From  2 Jul 2003, 15:30:00,  To  2 Jul 2003, 17:30:00
Current EOF        256 B       Maximum EOF        15728640 B
                               Percentage Used        0.00 %
```

For an active measurement, this command shows the number of entities being measured, the space allocated to each, the current and maximum size of the data file, and the percentage of space used. For example:

```
30+ STATUS MEASUREMENT $DATA.PERF.JUL01
From  2 Jul 2003, 15:30:00,  To  2 Jul 2003, 17:30:00
Cpu                 16 Entities            752 Words
File              1399 Entities          47566 Words
Discopen          5670 Entities          85050 Words
Disc               320 Entities          33920 Words
   Current EOF  464978 B       Maximum EOF         15728640 B
                               Percentage Used         2.95 %
```

Checking Data File Accessibility

To list the measurement data files accessible to MEASCOM, enter the INFO MEASUREMENT command. The file specified in the most recent ADD MEASUREMENT command is always marked as the current data file.

In this example, the first and third measurements are active; the second measurement is inactive; and the third data file is the current data file:

```
3+ INFO MEASUREMENT *
Add measurement $DATA.PERF.JUL01A
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:07:30
-- Add Cpu *
-- Add File $*
-- Add Discopen *
-- Add Disc $*

Add measurement $DATA.PERF.JUL01B
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:08:42,  To  2 Jul 2003, 15:09:49
Cpu                      15 Entities            705 Words
File                   1405 Entities          47770 Words
Discopen               5677 Entities          85155 Words
Disc                    320 Entities          33920 Words
-- Add Cpu *
-- Delete Cpu 6
-- Add File $*
-- Add Discopen *
-- Add Disc $*

Add measurement $DATA.PERF.JUL01C -- Current Data File --
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:30:00,  To  2 Jul 2003, 17:30:00
-- Add Cpu *
-- Delete Cpu 6
-- Add File $*
-- Add Discopen *
-- Add Disc $*
```

MEASCOM can display configuration information for any data file displayed by the INFO MEASUREMENT command. However, it can read measurement data only from the current data file. To change the current data file, you must issue the ADD MEASUREMENT `data-file-name` command.

For example, to make $DATA.PERF.JUL01A the current data file:

```
5+ ADD MEASUREMENT $DATA.PERF.JUL01A
6+ INFO MEASUREMENT *
Add measurement $DATA.PERF.JUL01A -- Current Data File --
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:07:30
-- Add Cpu *
-- Add File $*
-- Add Discopen *
-- Add Disc $*

Add measurement $DATA.PERF.JUL01B
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:08:42,  To  2 Jul 2003, 15:09:49
Cpu                      15 Entities             705 Words
File                   1405 Entities           47770 Words
Discopen               5677 Entities           85155 Words
Disc                    320 Entities           33920 Words
-- Add Cpu *
-- Delete Cpu 6
-- Add File $*
-- Add Discopen *
-- Add Disc $*

Add measurement $DATA.PERF.JUL01C
Data collected from system \HATI, MEASURE release version G10
From  2 Jul 2003, 15:30:00,  To  2 Jul 2003, 17:30:00
-- Add Cpu *
-- Delete Cpu 6
-- Add File $*
-- Add Discopen *
-- Add Disc $*
```

Potential Data File Errors

If the measurement data file becomes full before the end of a measurement, each MEASCTL process stops writing data to the file and displays this console message:

```
Measurement nn datafile: write to datafile failed.
                         File system err: 45
```

Closing information such as the ending measurement time and counter-space usage statistics cannot be written to the file. In this case, the measurement remains active and appears as an active measurement when you enter the STATUS commands. The counters continue to be updated in memory, but the data is not written out. To stop the measurement and close the data file, issue the STOP MEASUREMENT command.

When a write to a data file encounters any file-system error (except an error 43 or 45), the MEASCTL processes abort the measurement. The aborted measurement is inactive, but MEASMON keeps the file open until you stop the measurement. To stop the measurement and close the file, enter the STOP MEASUREMENT command.

If you issue a STATUS MEASSUBSYS command against an aborted measurement, the measurement is included in the list of active and configured measurements. If you try to access the active counters of an aborted measurement, an error 3407 is produced because the measurement is no longer active but is still configured:

```
MEAS 3407 Invalid measurement number; correct and retry.
MEAS 2000 Comment.  For more information type HELP 3407
```

Message 3407 indicates that MEASCOM passed an invalid measurement number to the MEASREADACTIVE procedure (the Measure procedure that returns the information for the LISTACTIVE command). To stop the measurement and close the data file, enter the STOP MEASUREMENT command.

For more information about MEASREADACTIVE, see Section 6, Creating a Custom Measurement Application, and the *Measure Reference Manual*.

For error message descriptions, see the online help or the *Measure Reference Manual*.

Viewing Reports of Measurement Data

You can view reports of measurement data from active data files, inactive data files, or active counters. To view a report, you must first make the measurement data file accessible to MEASCOM by using the ADD MEASUREMENT command.

To use this command, you need read access to the data file, and you might need free disk space up to three times the size of the data file on the disk containing the data file.

For example, to make the file $PERF.DATA.NOV5 accessible to MEASCOM:

```
2+ ADD MEASUREMENT $PERF.DATA.NOV5
```

When you finish with a data file, to stop the MEASFH process and free the disk space it was using, either:

- Enter the DELETE MEASUREMENT command to delete access to the file.
- Exit MEASCOM.

Viewing Reports From Data Files

A measurement data file can contain data from either a currently active or an inactive measurement. Each MEASCTL process writes counter values to the data file:

- At the beginning of the measurement
- When a transient entity starts or stops
- At each collection interval (if specified)
- At the end of the measurement

When you access a data file associated with a currently active measurement, you might encounter a delay before counters appear because the MEASCTL processes buffer data before writing it to the file. Also, an active data file contains data primarily from transient entities unless you started the measurement with a collection interval.

The ADD MEASUREMENT command creates a file-handling process (MEASFH) that reads configuration information and counter records from the measurement data file and translates them into external counter record formats. MEASFH creates an index in the data file and translates records from internal to external format. It requires sufficient free disk space to hold these structures.

MEASFH reads the data from the counter record, subtracts its initial value (if any), and passes the adjusted value to MEASCOM, which displays this adjusted value.

To examine data in an active or inactive measurement data file, enter the LIST *`entity-type`* and LISTALL *`entity-type`* commands. The LIST *`entity-type`* command displays the data collected for all counters of the entity since the measurement began. The LISTALL command displays the data collected for all counters of the entity at each collection interval specified in the START MEASUREMENT command.

Using the LIST *entity-type* Command

The LIST *`entity-type`* command displays one set of counter values for one or more entity specifications. These values represent data collected from the time the measurement began to the time the LIST command is issued.

In this example, the LIST *`entity-type`* command displays counter values for CPU 1 on a TNS system:

```
12+ LIST CPU 1
Cpu  1 VLX          Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 16        PCBs    483                Page Size  2048
Local System \HATI From 6 Dec 2003, 11:20:00 For 30 Minutes

Cpu-Busy-Time        64.23 %    Swaps                 0.76
Cpu-Qtime             1.36 #    Cpu-Qlen-Max            39 #
Mem-Qtime             0.10 #    Mem-Qlen-Max            13 #
Dispatches          205.30      Intr-Busy-Time        5.70 %
Process-Ovhd          0.12 %    Send-Busy-Time        0.61 %
Disc-IOs                        Cache-Hits
Transactions                    Response-Time
Page-Requests                   Page-Scans
Ending-Free-Mem       4092      Ending-UCME
Ending-UDS             500      Ending-SDS            1500
Ending-UCL             100      Ending-SCL            2000
```

Use an asterisk to select all specifications of an entity type. For example, this command displays counter values for all CPUs on a two-CPU TNS/R system running NSR-L processors:

```
13+ LIST CPU *
Cpu  0          Initial Lock Pgs 2048         Mem Pages  8192
```

```
Memory MB 32        PCBs    483               Page Size  4096
Local System \SPAR From 10 Nov 2003, 15:48:29 for 3.6 Minutes

Cpu-Busy-Time        23.81 %    Swaps
Cpu-Qtime             0.26 #    Cpu-Qlen-Max              63 #
Mem-Qtime                       Mem-Qlen-Max              24 #
Dispatches          126.83      Intr-Busy-Time          2.28 %
Process-Ovhd                    Send-Busy-Time          0.22 %
Disc-IOs              0.77      Cache-Hits              0.54
Transactions                    Response-Time
Page-Requests                   Page-Scans
Ending-Free-Mem       4092      Ending-UCME
Ending-UDS             500      Ending-SDS              1500
Ending-UCL             100      Ending-SCL              2000
Accel-Busy-Time      14.35 %    TNS-Busy-Time           0.13 %
TNSR-Busy-Time        7.05 %    Comp-Traps              2.04 #

++

Cpu 1 NSR-L         Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32        PCBs    483                Page Size 4096
Local System \SPAR From 10 Nov 2003, 15:48:29 for 3.6 Minutes

Cpu-Busy-Time         9.41 %    Swaps                   0.19
Cpu-Qtime             0.04 #    Cpu-Qlen-Max              63 #
Mem-Qtime             0.01 #    Mem-Qlen-Max              24 #
Dispatches           20.30      Intr-Busy-Time          0.65 %
Process-Ovhd          0.08 %    Send-Busy-Time          0.02 %
Disc-IOs              1.15      Cache-Hits              1.29
Transactions                    Response-Time
Page-Requests                   Page-Scans
Ending-Free-Mem       4092      Ending-UCME
Ending-UDS             500      Ending-SDS              1500
Ending-UCL             100      Ending-SCL              2000
Accel-Busy-Time       3.68 %    TNS-Busy-Time           4.13 %
TNSR-Busy-Time        0.95 %    Comp-Traps              2.04 #
```

Note. The ACCEL-BUSY-TIME, TNS-BUSY-TIME, COMP-TRAPS, and TNSR-BUSY-TIME counters appear only in measurements of TNS/R systems. The PROCESS and PROCESSH entity types also have counters specific to TNS/R systems. For counter descriptions, see the *Measure Reference Manual*.

This INFO MEASUREMENT command displays the measurement configuration for data file FILDATA. The LIST FILE command displays all files opened by processes in CPU 1 and their counter values.

```
14+ INFO MEASUREMENT FILDATA
Add measurement $PERF.DATA.FILDATA -- Current Data File --
Data collected from system \HATI, MEASURE release version G10
From  6 Dec 2003, 11:20:37,  Interval   10 Seconds
-- Add Cpu *
-- Add File *
15+ LIST FILE * (1, *)
File Open  $SYSTEM.SYSTEM.USERID
Device Type  3  (Disk)
```

```
Opener 1,66    ($PRF1) Program $SYSTEM.SYSTEM.PERF  File Num  11
Local System \HATI From  6 Dec 2003, 11:20:38 For 0.2 Seconds

File-Busy-Time               54.85 % Disc Reads
Reads                         4.66   Writes
Updates-or-Replies            4.66   Deletes-or-Writereads
Records-Used                         Records-Accessed
Messages                     18.62   Message-Bytes        2,830
Lock-Waits                           Timeouts
Escalations                          Info-Calls

++

File Open  $SYSTEM.SYSTEM.FSEARCH
Device Type  3  (Disk)
Opener 1,76   ($ZTA)  Program $SYSTEM.SYSTEM.AUDIT  File Num   1
Local System \HATI From 6 Dec 2003, 11:23:27 For  0.5 Seconds

File-Busy-Time               10.63 % Disc-Reads           1.98
Reads                         3.97   Writes
Updates-or-Replies                   Deletes-or-Writereads
Records-Used                         Records-Accessed
Messages                      21.83  Message-Bytes        1,702
Lock-Waits                           Timeouts
Escalations                          Info-Calls           1.98
       .
       .
       .
```

Note. Because you can open processes as files, the FILE entity type can display file names that do not match MEASCOM syntax. For example, a name such as \BUYER.05,048, which is syntactically incorrect, might be listed under File Open. To refer to the file associated with such a name, you must specify its file number, the CPU number, and the PIN of the opener process in the LIST command. For example:

```
9+ LIST FILE * (5,40,1)
File Open \BUYER.05,048
Device Type  0 (Process)
Opener 5,40      ($USR)  Program $SYSTEM.SYSTEM.TRANS   File Num   1
Local System \BUY From 5 Nov 2003, 17:15:11 For 0.112 Seconds

File-Busy-Time                0.01    Disc-Reads%
Reads                         0.03    Writes
Updates-or-Replies            0.03    Deletes or Writereads
     .
     .
     .
```

When using the LIST *entity-type* command, if you specify an entity that returns no records, the record for that entity probably contains all zero values. By default, MEASCOM suppresses the display of such records. To override this default, you can set the ZERO-REPORTS option of the REPORT object or reenter the LIST *entity-type* command with its ZERO-REPORTS INCLUDE option to indicate that all records should be displayed. For example:

```
16+ LIST FILE * (5,*), ZERO-REPORTS INCLUDE
```

You can control the content and format of reports by using the BY and IF options of the LIST command or by setting REPORT object attributes. For details and examples, see Section 4, Formatting Reports and Plots.

MEASCOM automatically allocates an extended segment to hold retrieved counter records, increasing the segment size as needed up to 16 megabytes. MEASCOM displays a warning message each time it resizes the segment. If a report uses so much memory that it causes system problems, stop MEASCOM and redefine the LIST command to retrieve fewer records (for example, by defining a time window, using fewer wild-card characters, or using the BY or IF clause).

Using the LISTALL *entity-type* Command

The counter values displayed by the LISTALL `entity-type` command are associated with the collection interval specified for a measurement. If you do not specify a collection interval, LISTALL `entity-type` operates like the LIST `entity-type` command. If you do specify a collection interval, LISTALL `entity-type` displays the values recorded at the end of each elapsed interval.

In this example, a measurement was started at 11:30 a.m. with a collection interval of 30 minutes. At 12:30, a LISTALL command was issued, requesting information on CPU 1. Two sets of counter values are displayed for CPU 1—one for each collection interval that has elapsed since the measurement started.

```
3+ START $PERF.DEC06.DATA1, FROM 11:30, INTERVAL 30 MINUTES
      .
      .
      .
20+ LISTALL CPU 1
Cpu 1 VLX          Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 16       PCBs    256                Page Size  2048
Local System \HATI From 6 Dec 2003, 11:30:00 For 30 Minutes
Cpu-Busy-Time        64.23 %     Swaps                0.76
Cpu-Qtime             1.36 #     Cpu-Qlen-Max           39 #
Mem-Qtime             0.10 #     Mem-Qlen-Max           13 #
Dispatches          205.30       Intr-Busy-Time       5.70 %
Process-Ovhd          0.12 %     Send-Busy-Time       0.61 %
Disc-IOs                         Cache-Hits
Transactions                     Response-Time
Page-Requests                    Page-Scans
Ending-Free-Mem       4092       Ending-UCME
Ending-UDS             500       Ending-SDS           1500
Ending-UCL             100       Ending-SCL           2000

Cpu 1 VLX          Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 16       PCBs    256                Page Size  2048
Local System \HATI From 6 Dec 1995, 12:00:00 For 30 Minutes

Cpu-Busy-Time        60.34 %     Swaps                2.61
Cpu-Qtime             1.78 #     Cpu-Qlen-Max           39 #
Mem-Qtime             0.24 #     Mem-Qlen-Max           13 #
Dispatches          162.35       Intr-Busy-Time       6.00 %
Process-Ovhd          0.16 %     Send-Busy-Time       1.11 %
```

```
Disc-IOs                              Cache-Hits
Transactions                          Response-Time
Page-Requests                         Page-Scans
Ending-Free-Mem       4092            Ending-UCME
Ending-UDS             500            Ending-SDS              1500
Ending-UCL             100            Ending-SCL              2000
```

Use an asterisk to select all specifications of an entity. When you use the measurement described previously, the next command lists two reports for each CPU being measured:

```
21+ LISTALL CPU *
```

Use the FROM, TO, and FOR clauses to display data collected between specific times. For example, to request listings for CPU 2 for intervals completed between 9:00 a.m. and noon:

```
22+ LISTALL CPU 2, FROM 9:00, TO 12:00
```

You can specify the same timeframe using the FROM-FOR clause. For example:

```
23+ LISTALL CPU 2, FROM 9:00, FOR 3 HOURS
```

To control the content and format of LISTALL reports, use BY and IF clauses and REPORT attributes. For more information, see Section 4, Formatting Reports and Plots, and the *Measure Reference Manual*.

Viewing Reports From Active Counters

Active counter records are read by the MEASCTL processes rather than by the MEASFH process. Each MEASCTL process reads the data from the counters and passes the information to MEASCOM.

Differences between active counter values and active data file counter values occur in two cases:

- Examining a CPU entity

  CPU counters are initialized when the CPU is system loaded. Each MEASCTL process stores current counter values in the data file when a measurement starts. These values are used to compute CPU use during the measurement. Because MEASCTL does not initialize the CPU counters when a measurement starts, it returns the current CPU counter values, which reflect CPU activity since the system load.

- Examining an entity included in more than one active measurement

  Counter values are stored in a data file for each measurement when the measurement starts. These initial values are used to compute the counter values for the life of a measurement. MEASCTL does not keep the counter values for the start of each measurement, so it returns counter values that reflect entity activity since the counter was initialized by the first measurement started.

Identifying Active Measurements

To list currently active measurements, enter the STATUS MEASSUBSYS command:

```
10+ STATUS MEASSUBSYS
Number of Active (or Configured) Measurements =  1
  $PERF.DATA.FILCPU
```

To list the configuration of an active measurement, enter the ADD MEASUREMENT command to make the measurement available to MEASCOM and the INFO MEASUREMENT command to display the configuration:

```
11+ ADD MEASUREMENT $PERF.DATA.FILCPU
12+ INFO MEASUREMENT $PERF.DATA.FILCPU
Add measurement $PERF.DATA.FILCPU -- Current Data File --
Data collected from system \HATI, MEASURE release version G10
From  5 Apr 2003,  17:15:10,  Interval   30 Minutes
-- Add Cpu *
-- Add File $MYVOL.MYSUBVOL.*
```

Using the LISTACTIVE *entity-type* Command

To examine currently active counters, enter the LISTACTIVE *entity-type* command. The format and output of this command are similar to those of the LIST *entity-type* command. However, the LISTACTIVE *entity-type* command displays data for only one entity at a time.

Note. The LISTACTIVE *entity-type* command cannot be used to read active counters for DISCOPEN or PROCESSH entities. If you specified a collection interval, you can read data for these entities in an active data file by using either the LIST *entity-type* or the LISTALL *entity-type* command. If you did not specify a collection interval, you cannot view the data until the measurement stops.

You must specify a unique entity in the LISTACTIVE command because each MEASCTL process uses the entity specification to locate the active counter record in system counter space. In this example, the CPU, PIN, and file number (0,38,1) uniquely identify the open of the user ID file.

```
13+ LISTACTIVE FILE * (0,38,1)
File Open  $SYSTEM.SYSTEM.USERID
Opener 0,38   ($TMP) Program $SYSTEM.SYSTEM.TRANSACT  File Num 1
Local System \HATI From 6 Dec 2003, 11:20:01 For 12 Seconds

File-Busy-Time                          Disc-Reads             1.98
Reads                    0.43           Write                  0.92
Updates-or-Replies                      Deletes-or-Writereads  0.67
Records-Used                            Records-Accessed
Messages                 3.51           Message-Bytes          0.10
Lock-Waits                              Timeouts
Escalations                             Info-Calls
```

You can use the FOR clause of the LISTACTIVE command to define a specific timeframe for monitoring. For example, this command displays counter values for CPU

0 collected over a 5-minute period. The period begins when you enter the LISTACTIVE command.

```
24+ LISTACTIVE CPU 0, FOR 5 MINUTES
Cpu 0 VLX          Initial Lock Pgs 2048        Mem Pages  8192
Memory MB 16       PCBs    256                  Page Size  2048
Local System \HATI From 6 Dec 2003, 11:25:11 For 5 Minutes

Cpu-Busy-Time        15.83 %       Swaps
Cpu-Qtime             0.19 #       Cpu-Qlen-Max           24 #
Mem-Qtime                          Mem-Qlen-Max            7 #
Dispatches           88.79         Intr-Busy-Time       3.58 %
Process-Ovhd          0.01 %       Send-Busy-Time       0.32 %
Disc-IOs             11.71         Cache-Hits          30.09
Transactions                       Response-Time
Page-Requests                      Page-Scans
Ending-Free-Mem       4092         Ending-UCME
Ending-UDS             500         Ending-SDS           1500
Ending-UCL             100         Ending-SCL           2000
```

When you enter a LISTACTIVE command with a FOR clause interactively, your terminal remains locked until the time period ends and the listing is displayed.

4 Formatting Reports and Plots

Measure provides two types of data displays:

Display Type	Display...	Commands That Produce Them
Reports	Lists of counter values	`LIST` *`entity-type`*, `LISTALL` *`entity-type`*, `LISTACTIVE` *`entity-type`*
Plots	Data values in graphs	`LIST PLOT`

You can also use MEASCOM to convert Measure data files, which are unstructured, to structured files. Report-writing products, such as the Enform query product and the NonStop SQL/MP Report Writer, can access structured files to produce more customized reports of measurement data.

Controlling Content and Format of Reports

When you enter the LIST *`entity-type`*, LISTALL *`entity-type`*, and LISTACTIVE *`entity-type`* commands, MEASCOM displays the measurement data for the specified entity vertically in lists on a screen or page. Multiple reports are displayed one after another. Continuations are indicated by a double plus sign(++) in the prompt area on a screen.

To change the content and format of reports, either:

- Set REPORT attributes in MEASCOM. The settings apply until you reset them, change them, or end the session. They return to their default values when you exit MEASCOM.
- Specify REPORT attributes and reporting clauses in the LIST *`entity-type`*, LISTALL *`entity-type`*, and LISTACTIVE *`entity-type`* commands. The settings override the REPORT attribute settings but apply only to the command that contains them.

REPORT Attributes

The LIST *`entity-type`* and LISTALL *`entity-type`* commands generate reports for one or more entities from data in active and inactive data files. The LISTACTIVE *`entity-type`* command generates reports for one entity at a time from data in active counters. With REPORT attributes, you can modify the content and format of any reports produced by these commands.

To display the current settings of REPORT attributes, use the SHOW REPORT command. For example, to display the setting for the REPORT TOTALS attribute:

```
5+ SHOW REPORT TOTALS
  Set Report Totals suppress
```

To display all REPORT attributes:

```
6+ SHOW REPORT *
  Set Report Dots off
  Set Report Format normal
  Set Report Rate on
  Set Report Style legacy
  Set Report Totals suppress
  Set Report Zero-Values suppress
  Set Report Zero-Reports suppress
  Set Report From
  Set Report To
  Set Report For
  Set Report LoadId
  Set Report Style legacy
  Set Report Dots off
```

These values are the initial defaults for the REPORT attributes. They are used for all reports unless you set new values or specify different attributes in the LIST `entity-type`, LISTALL `entity-type`, or LISTACTIVE `entity-type` commands.

For a summary or these attributes, see Table 4-1 on page 4-3. For detailed descriptions, see the *Measure Reference Manual*.

Setting Report Attributes

To set the REPORT attributes in MEASCOM, use the SET REPORT command. For example, to set the default FORMAT attribute to BRIEF and verify it with the SHOW command:

```
7+ SET REPORT FORMAT BRIEF
8+ SHOW REPORT FORMAT
  Set Report Format brief
```

Unless you specify a different FORMAT attribute in a LIST `entity-type`, LISTALL `entity-type`, or LISTACTIVE `entity-type` command or set or reset this attribute, all subsequent reports of data from this data file appear in BRIEF format.

You can set several REPORT attributes in the same command. For example:

```
9+ SET REPORT TOTALS INCLUDE, ZERO-VALUES INCLUDE, RATE OFF
10+ SHOW REPORT *
  Set Report Dots off
  Set Report Format brief
  Set Report Rate off
  Set Report Style legacy
  Set Report Totals include
  Set Report Zero-Values include
  Set Report Zero-Reports include
  Set Report From
  Set Report To
  Set Report For
  Set Report LoadId
```

Table 4-1. MEASCOM REPORT Attributes (page 1 of 2)

Attribute	**Description**	
FORMAT	Sets the basic format of reports:	
	NORMAL	displays all counters (default).
	BRIEF	displays a Measure-defined subset of counters.
	STRUCTURED	writes data to structured files for use by report-writing products.
RATE	Specifies whether to display interpreted or uninterpreted counter values:	
	ON	displays interpreted counter values (default).
	OFF	displays uninterpreted counter values.
TOTALS	Specifies whether to create a summary record containing the total values of each counter across the reports produced by the LIST command:	
	SUPPRESS	displays specified reports without totals (default).
	ONLY	displays counter totals only.
	INCLUDE	displays specified reports and counter totals.
ZERO-VALUES	Specifies whether to display counter values of less than 0.005:	
	SUPPRESS	displays counter values less than 0.005 as zeros or blanks (default).
	INCLUDE	displays all actual values.
ZERO-REPORTS	Specifies whether to display records whose counter values are all zeros:	
	SUPPRESS	does not display these records (default).
	INCLUDE	displays all counter records regardless of value.
FROM [*date/time*]	Defines the beginning of a report window. Date and time are optional. Use by itself or with the TO or FOR clause.	
TO [*date/time*]	Defines the end of a report window. Date and time are optional. Use by itself or with the FROM clause.	
FOR *duration*	Defines the duration of a report window. Use by itself or with the FROM clause.	
STYLE	Specifies whether displays or structured reports are based on the ZMS-style (Measure G11 and later) or legacy-style records and report formats.	
	ZMS	uses ZMS-style records and reports.
	LEGACY	uses legacy-style format records and reports (default).

Table 4-1. MEASCOM REPORT Attributes (page 2 of 2)

Attribute	Description	
DOTS	Specifies whether displays include a string of dots between counter labels and a formatted numeric value (Measure G11 and later).	
	ON	adds dots to the report display.
	OFF	formats spaces between labels and numeric values (default).
LOADID *loadid*	Specifies the name to be put in the LOADID field of the records generated by this command.	
CR-NAME-LEN	Controls the displayed length of procedure (code-range) names:	
	SHORT	truncates procedure (code-range) names (if necessary) at 32 characters.
	LONG	displays both short and long forms of procedure names.
CR-NAME-FORM	Controls whether demangled procedure (code-range) names are displayed (if applicable):	
	STANDARD	displays whatever form was specified in the code file (mangled) or EDIT file (demangled). STANDARD is the default.
	DEMANGLED	displays demangled procedure names.
	BOTH	displays both STANDARD and DEMANGLED procedure names.
CR-NAME-QUAL	Controls whether procedure (code-range) names are displayed with object file name qualifiers, if available:	
	UNQUALIFIED	displays in traditional form with no qualifiers (default).
	QUALIFIED	with the Guardian or OSS object file name of the associated code.

Resetting REPORT Attributes

To reset a REPORT attribute to its initial default setting, use the RESET REPORT command. For example, to reset the FORMAT attribute to its initial default value:

```
9+ RESET REPORT FORMAT
10+ SHOW REPORT FORMAT
  Set Report Format normal
```

To reset all REPORT attributes to their initial defaults, use an asterisk in the RESET command. For example:

```
11+ RESET REPORT *
12+ SHOW REPORT *
  Set Report Dots off
  Set Report Format normal
  Set Report Rate on
  Set Report Style ZMS
  Set Report Totals suppress
  Set Report Zero-Values suppress
  Set Report Zero-Reports suppress
  Set Report From
  Set Report To
  Set Report For
  Set Report LoadId
```

Overriding REPORT Attributes

To override REPORT attributes, specify them in the LIST *entity-type*, LISTALL *entity-type*, and LISTACTIVE *entity-type* commands. The values specified in these commands remain in effect only for the life of the command. The settings of the REPORT attributes remain unchanged. For example, to change the NORMAL format named in the previous RESET command to BRIEF format for this display of CPU data:

```
14+ LIST CPU *, FORMAT BRIEF
```

In addition, to control report format and content, use the BY and IF clauses of the LIST *entity-type* and LISTALL *entity-type* commands. Because the LISTACTIVE command displays only one entity at a time, it does not support the BY and IF clauses.

The IF clause displays reports based on the value of a specified counter or a numeric identification item such as PIN or PRIORITY. (The DDL record for each entity type lists the identification items associated with that entity. See the *Measure Reference Manual*.)

To display only the records that contain a CPU-BUSY-TIME value greater than 10:

```
5+ LIST CPU *, IF CPU-BUSY-TIME > 10
```

For PROCESSH entities, the object of the IF clause is a code-range name rather than a counter name. For example:

```
8+ LIST PROCESSH $ABC, BY CODE-RANGE, IF CODE-RANGE > 0
```

For DISC entities, the counter name for a cache counter must be preceded by C0-, C1-, C2-, or C3- to differentiate between the different cache block sizes. To display only disk records that contain a HITS value for the 512-byte cache (C0) greater than 10:

```
9+ LIST DISC *, IF C0-HITS > 10
```

The BY clause arranges reports in order based on the value of a specified counter or numeric identification item. You can specify an ascending or descending sort order. By default, counter items are sorted in descending order, and identification items are sorted in ascending order.

For example, to display all CPU reports in ascending order according to the value of the CPU-BUSY-TIME counter:

```
6+ LIST CPU *, BY CPU-BUSY-TIME (ASCENDING)
```

For PROCESSH entities, the object of the BY clause is a code-range name rather than a counter name.

For DISC entities, the counter name for a cache counter must be preceded by C0-, C1-, C2-, or C3- to differentiate between the different cache block sizes. For example, to display all DISC records in descending order according to the value of the HITS counter for the 512-byte cache (C0):

```
7+ LIST DISC *, BY C0-HITS
```

Note. Changes made to the REPORT attributes ZERO-REPORTS, FORMAT, and TOTALS and the IF clause of the LIST *entity-type* and LISTALL *entity-type* commands also change the information written to structured report files. For example, if you specify FORMAT STRUCTURED and TOTALS ONLY, only the record containing counter totals is written to the structured report file.

Changes made to the REPORT attributes RATE and ZERO-VALUES and the BY clause of the LIST *entity-type* and LISTALL *entity-type* commands do not change the information written to structured report files.

Controlling the Report Window

Specifying a report window directs MEASCOM to read and display only records written to the data file within a specified time period (plus or minus half a collection interval). Because you are viewing a subset of the available data, the statistics displayed by a report window are typically different from those displayed by default.

Use the TO, FROM, and FOR clauses of the LIST *entity-type* and LISTALL *entity-type* commands or the FOR clause of the LISTACTIVE *entity-type* command to adjust the report window.

Example 4-1 is a standard data display showing all the data collected for CPU 1 during a 2-hour measurement. The last four lines of the report apply to measurements taken on TNS/R systems. These lines do not appear for measurements taken on TNS systems.

Example 4-1. Legacy Format Report (Listed Format)

```
9+ LIST CPU 1

Cpu  1 NSR-L       Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32        PCBs    256                Page Size 4096
Local System \SPAR From 17 Aug 2003, 11:20:00  For 2 Hours

Cpu-Busy-Time        48.19 %  Swaps                1.26 #
Cpu-Qtime             1.07 #  Cpu-Qlen-Max           39 #
Mem-Qtime             0.11 #  Mem-Qlen-Max           13 #
Dispatches          155.31 #  Intr-Busy-Time       4.73 %
Process-Ovhd          0.16 %  Send-Busy-Time       0.52 %
Disc-IOs              1.70    Cache-Hits           0.76
Transactions                  Response-Time
Page-Requests                 Page-Scans
Ending-Free-Mem       4092    Ending-UCME             0
Ending-UDS             500    Ending-SDS           1500
Ending-UCL             100    Ending-SCL           2000
Accel-Busy-Time       0.95 %  TNS-Busy-Time        0.10 %
TNSR-Busy-Time        1.00 %  Comp-Traps          13.37 #
```

Example 4-2 shows the brief form of this report. Counters displayed are predefined for each entity. For a list of the counters included in brief reports, see the *Measure Reference Manual*.

Example 4-2. Brief Version of Legacy Format Report

```
10+ LIST CPU 1, FORMAT BRIEF
Cpu  1 NSR-L       Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32       PCBs    256                Page Size 4096
Local System \SPAR From 17 Aug 1996, 11:20:00  For 2 Hours

Cpu-Busy-Time        48.19 %  Swaps                1.26
Ending-Free-Mem       4096    Ending-UCME             0
```

Example 4-3 shows data collected for the same CPU (1), but between the hours of 11:30 and 12:00. The FROM and TO clauses specify a report window of 30 minutes, beginning at 11:30.

Example 4-3. Setting a Report Window

```
11+ LIST CPU 1, FROM 11:30, TO 12:00
Cpu  1 NSR-L       Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32       PCBs    256                Page Size 4096
Local System \SPAR From 17 Aug 2003, 11:30:00 For 30 Minutes

Cpu-Busy-Time        38.73 %  Swaps                  1.49
Cpu-Qtime             1.01 #  Cpu-Qlen-Max             39 #
Mem-Qtime             0.13 #  Mem-Qlen-Max             13 #
Dispatches          111.27    Intr-Busy-Time         6.16 %
Process-Ovhd          0.11 %  Send-Busy-Time         0.64 %
Disc-IOs                      Cache-Hits
Transactions                  Response-Time
Page-Requests                 Page-Scans
Ending-Free-Mem     4092      Ending-UCME               0
Ending-UDS           500      Ending-SDS            1500
Ending-UCL           100      Ending-SCL            2000
Accel-Busy-Time      18.45 %  Accel-Busy-Time        3.10 %
TNSR-Busy-Time       11.02 %  Comp-Traps             9.07 #
```

Displaying Interpreted and Uninterpreted Values

You can display interpreted or uninterpreted counter values in reports:

- Interpreted values are raw counts divided by delta-time per second. Interpreted values are particularly useful for making relative comparisons of data.
- Uninterpreted values are actual counts taken during the measurement period. Uninterpreted values are particularly useful for determining the time needed to run a process or application, how often an event occurs over the life of a process or application, and so on.

To display interpreted values, set REPORT RATE ON (the default). To display uninterpreted values, set REPORT RATE OFF.

Example 4-4 shows uninterpreted counter values displayed for CPU 1 of a TNS VLX system. Compare this report with the report in Example 4-3 on page 4-8, which shows interpreted values (for a different measurement).

The REPORT RATE attribute also determines whether plots contain interpreted or uninterpreted values. For information on creating plots, see Plotting Measurement Data on page 4-10.

Example 4-4. Report of Uninterpreted Counter Values

```
44+ LIST CPU 1, RATE OFF
Cpu  1 NSR-L       Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32       PCBs    256                Page Size 4096
Local System \SPAR From 16 Aug 1996, 11:20:00 For 3 Minutes

Cpu-Busy-Time        52.04 sec Swaps
Cpu-Qtime            56.47 sec Cpu-Qlen-Max            63 #
Mem-Qtime                      Mem-Qlen-Max            24 #
Dispatches          27,599 #   Intr-Busy-Time        4.99 sec
Process-Ovhd                   Send-Busy-Time      473.82 ms
Disc-IOs               169 #   Cache-Hits             119 #
Transactions                   Response-Time
Page-Requests                  Page-Scans
Ending-Free-Mem       4092     Ending-UCME              0
Ending-UDS             500     Ending-SDS            1500
Ending-UCL             100     Ending-SCL            2000
Accel-Busy-Time      35.76 sec TNS-Busy-Time       106.00 ms
TNSR-Busy-Time       10.23 sec Comp-Traps             445 #
```

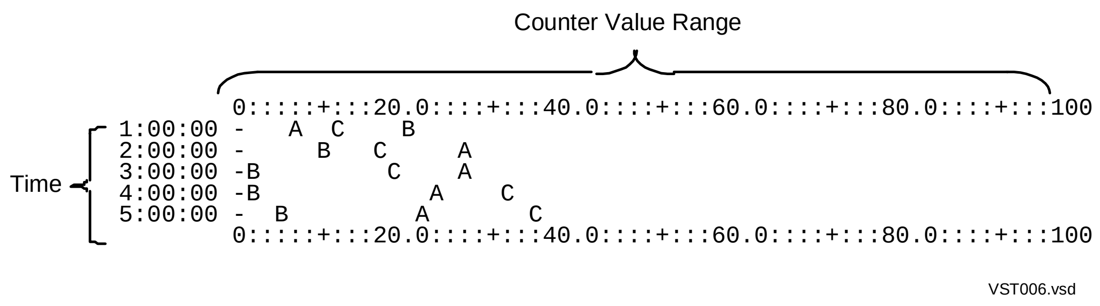
Plotting Measurement Data

Plots are used to compare counters graphically. You can plot data from an active or inactive measurement data file. You cannot plot data from active counter records or from multiple data files.

MEASCOM provides two types of plots: two-axis plots and bar graphs. Both types of plots can be modified using PLOT attributes.

A two-axis plot shows time on one axis and the counter value range on the other. Plot characters inside the graph mark counter values for each entity at each time. The two-axis plot is the default plot type when a collection interval is used to collect the data.

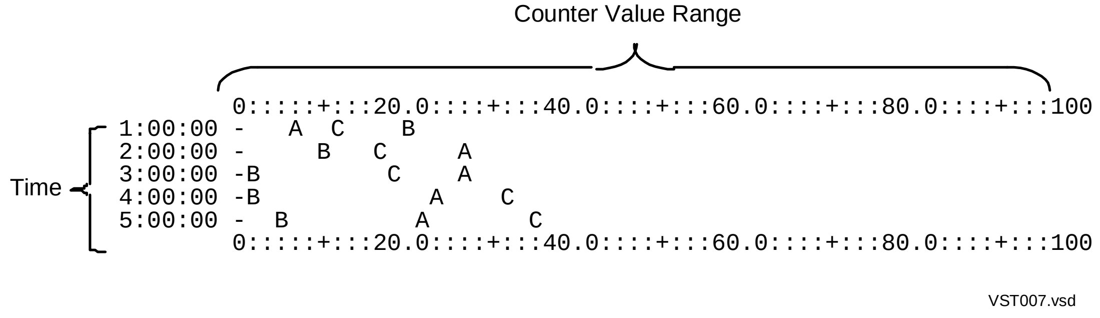
Figure 4-1. Example: Basic Two-Axis Plot Format



```
        0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
1:00:00 -   A  C    B
2:00:00 -     B   C     A
3:00:00 -B         C    A
4:00:00 -B            A    C
5:00:00 -  B         A       C
        0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
```

A bar graph shows the plot character for each entity-counter pair on the vertical axis and the counter value range on the horizontal axis. Bars of asterisks show average counter values for each entity-counter pair during the measurement period. (For the PROCESSH entity, the bars show counter value totals rather than averages.) The bar graph is the default plot type when a collection interval is not used to collect the data.

Figure 4-2. Example: Basic Bar Graph Format



```
        0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
1:00:00 -   A  C    B
2:00:00 -     B   C     A
3:00:00 -B         C    A
4:00:00 -B            A    C
5:00:00 -  B         A       C
        0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
```

Variations on these basic plot formats are described and illustrated in this section.

To plot data:

1. Define the plot using the LIST *`entity-type`* command followed by the ADD PLOT commands.
2. Set PLOT attributes.
3. Generate the plot using the LIST PLOT command.

The Plot Definition

A plot definition is a list of entity-counter pairs built by MEASCOM. The LIST *`entity-type`* command specifies the entities to plot. The ADD PLOT command specifies the counters to plot.

A plot definition can contain up to 26 counters. When you specify a counter using the ADD PLOT command, MEASCOM reads the list of entities specified by the most recent LIST *`entity-type`* command, pairs the same counter to each entity in that list, and adds the entity-counter pairs to the plot definition.

In Measure G11 and later PVUs, a plot can display ANSI SQL names as well as Guardian file names.

In this example, CPU-BUSY-TIME counters from all CPUs (specified in the LIST CPU command) are included in a plot definition (specified by the ADD PLOT command):

```
5+ LIST CPU *
Cpu  0 NSR-L       Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 32       PCBs    256                Page Size 4096
Local System \HATI  From 6 Dec 2003, 11:20:00  For 2 Hours

Cpu-Busy-Time        16.00 %    Swaps
Cpu-Qtime             0.21 #    Cpu-Qlen-Max            24 #
Mem-Qtime                       Mem-Qlen-Max             7 #
Dispatches          101.52      Intr-Busy-Time        6.51 %
Process-Ovhd                    Send-Busy-Time        0.57 %
Disc-IOs              9.28      Cache-Hits           24.27
Transactions                    Response-Time
Page-Requests                   Page-Scans
Ending-Free-Mem       4092      Ending-UCME              0
Ending-UDS             500      Ending-SDS            1500
Ending-UCL             100      Ending-SCL            2000
Accel-Busy-Time        1.34 %   TNS-Busy-Time          0.95 %
TNSR-Busy-Time         8.20 %   Comp-Traps             0.50

++
Cpu  1 VLX         Initial Lock Pgs 2048      Mem Pages  8192
Memory MB 16       PCBs    483                Page Size  2048
Local System \HATI  From  6 Dec 2003, 11:20:00  For 2 Hours

Cpu-Busy-Time        48.19 %    Swaps                 1.26
Cpu-Qtime             1.07 #    Cpu-Qlen-Max            39 #
```

```
Mem-Qtime             0.11 #    Mem-Qlen-Max           13 #
Dispatches          155.31      Intr-Busy-Time       4.73 %
Process-Ovhd          0.16 %    Send-Busy-Time       0.52 %
Disc-IoS                        Cache-Hits
Transactions                    Response_Time
Page-Requests                   Page-Scans
Ending-Free-Mem       4092      Ending-UCME             0
Ending-UDS             500      Ending-SDS           1500
Ending-UCL             100      Ending-SCL           2000
```

```
6+ ADD PLOT CPU-BUSY-TIME
```

To add other entity-counter pairs to the plot definition, you must specify another LIST *entity-type* command followed by the ADD PLOT *counter-name* command. (For an example of graphing different entity-counter pairs in the same plot, see Generating Plots on page 4-16.)

Displaying the PLOT Definition

The INFO PLOT command displays the current plot definition, the LIST *entity-type* command used to identify the entities, and the ADD PLOT command used to specify the counters. For example, this INFO command displays the plot definition of the previous CPU example:

```
7+ INFO PLOT *
Add measurement $SPOOL.PERF.DATA1
--A-- List Cpu 0
      Add plot CPU-BUSY-TIME
--B-- List Cpu 1
      Add plot CPU-BUSY-TIME
--C-- List Cpu 2
      Add plot CPU-BUSY-TIME
  .
  .
  .

--L-- List Cpu 11
      Add plot CPU-BUSY-TIME
```

Deleting Entity-Counter Pairs From a Plot Definition

To delete one or more counters from a plot definition, enter the DELETE PLOT command. For example, this command deletes the CPU-BUSY-TIME counter for CPU 0 from the previous definition:

```
8+ DELETE PLOT CPU-BUSY-TIME (A)
```

If more than one counter of the same name is specified in a plot definition, you must specify both its name (CPU-BUSY-TIME) and its plot character (A) in the DELETE PLOT command to distinguish the entity-counter pair to be deleted. If the counter name is unique in the plot definition, you can specify only its name in the DELETE PLOT command.

When the plot definition is redisplayed, the counter for CPU 0, associated with plot character A, has been deleted:

```
9+ INFO PLOT *
Add measurement $SPOOL.PERF.DATA1
--B-- List Cpu 1
      Add plot CPU-BUSY-TIME
--C-- List Cpu 2
      Add plot CPU-BUSY-TIME
  .
  .
  .

--L-- List Cpu 11
      Add plot CPU-BUSY-TIME
```

To delete all counters of the same type from a plot definition, use an asterisk in place of the plot character. For example, this command deletes all entity-counter pairs defined for CPU-BUSY-TIME from the plot definition:

```
8+ DELETE PLOT CPU-BUSY-TIME (*)
```

When you issue the LIST PLOT command after deleting entity-counter pairs, the pairs are relettered sequentially beginning with the letter A. Even though you delete an entity-counter pair from a definition, its letter can be reused in the plot display. A key below the plotted data lists each plot character shown in the graph, its counter name, and its entity specification. Use this key to identify the newly assigned plot characters. (For examples of the key, see Generating Plots on page 4-16.)

PLOT Attributes

With PLOT attributes, you can control the format of the graphic displays generated by the LIST PLOT command.

To display the current settings of PLOT attributes, enter the SHOW PLOT command:

```
5+ SHOW PLOT
  Set Plot Vert-Base  on
  Set Plot Time-Base
  Set Plot Scale-From       0.000
  Set Plot Scale-To       100.000
  Set Plot Wide-Item  off
  Set Plot From
  Set Plot To
  Set Plot For
```

These values are the initial defaults of the PLOT attributes. They are used for all data plots unless you set new values or specify different attributes in the LIST PLOT command. When you select a new data file using the ADD MEASUREMENT command or exit from MEASCOM, the PLOT attributes return to their default settings. For a summary of PLOT attributes, see Table 4-2 on page 4-14. For detailed descriptions, see the *Measure Reference Manual*.

Note. To select interpreted or uninterpreted counter values for plotting, you must set the REPORT RATE attribute. If REPORT RATE is ON, interpreted values are plotted when you issue the LIST PLOT command. If REPORT RATE is OFF, uninterpreted values are plotted when you issue the LIST PLOT command. For more information about the REPORT RATE attribute, see Displaying Interpreted and Uninterpreted Values on page 4-8.

Table 4-2. MEASCOM PLOT Attributes

Attribute	Description	
VERT-BASE	For a two-axis plot, varies the axis on which time is displayed. For a bar graph, varies the orientation of the bars:	
	ON	Two-axis plot: displays time on the vertical axis (default). Bar graph: bases bars on the vertical axis and positions them horizontally (default).
	OFF	Two-axis plot: displays time on the horizontal axis. Bar graph: bases bars on the horizontal axis and positions them vertically.
TIME-BASE	Specifies whether a two-axis plot or a bar graph is displayed:	
	ON	Displays a two-axis plot of counter values over time. If a collection interval was used to collect data, ON is the default.
	OFF	Displays a bar graph of average percentage counter values. If a collection interval was not used to collect data, OFF is the default.
SCALE-FROM *n*	Sets the lower boundary of the value axis. *n* is a number from 0 through 999999999999.999. The default is 0.	
SCALE-TO *n*	Sets the upper boundary of the value axis. *n* is a number from 0 through 999999999999.999. The default is 100.	
WIDE-ITEM	Sets the density of a plot.	
	OFF	For a two-axis plot, displays one plot character for each counter value at each interval (default). For a bar graph, uses bars that are one-character wide (default).
	ON	For a two-axis plot, fills the area between the time axis and the lowest counter value. For a bar graph, uses bars that are from two to six characters wide (adjusted to the number of values on the graph).
FROM [*date/time*]	Defines the beginning of a plot window. Date and time are optional. Use by itself or with the TO or FOR clause.	
TO [*date/time*]	Defines the end of a plot window. Date and time are optional. Use by itself or with the FROM clause.	
FOR *duration*	Defines the duration of a plot window. Use by itself or with the FROM clause.	

Setting PLOT Attributes

To set the PLOT attributes in MEASCOM, use the SET PLOT command. For example, to set the default VERT-BASE attribute to OFF:

```
7+ SET PLOT VERT-BASE OFF
8+ SHOW PLOT VERT-BASE
  Set Plot Vert-Base  off
```

The SHOW command displays the new setting.

You can set several PLOT attributes in the same command. For example:

```
9+ SET PLOT TIME-BASE OFF, SCALE-FROM 15, SCALE-TO 50
10+ SHOW PLOT
  Set Plot Vert-Base  off
  Set Plot Time-Base  off
  Set Plot Scale-From      15.000
  Set Plot Scale-To        50.000
  Set Plot Wide-Item  off
  Set Plot From
  Set Plot To
  Set Plot For
```

To reset any PLOT attribute to its default setting, use the RESET PLOT command:

```
9+ RESET PLOT VERT-BASE
10+ SHOW PLOT VERT-BASE
  Set Plot Vert-Base  on
```

To reset all PLOT attributes to their initial defaults, use an asterisk in the RESET command:

```
11+ RESET PLOT *
12+ SHOW PLOT
  Set Plot Vert-Base  on
  Set Plot Time-Base
  Set Plot Scale-From       0.000
  Set Plot Scale-To       100.000
  Set Plot Wide-Item  off
  Set Plot From
  Set Plot To
  Set Plot For
```

Overriding PLOT Attributes

You can override PLOT attribute settings by specifying them in the LIST PLOT command. The values specified in this command remain in effect only for the life of the command. The setting of the PLOT attributes remain unchanged. For example, to change the orientation of the data in the plot from vertical to horizontal:

```
14+ LIST PLOT, VERT-BASE OFF
```

Generating Plots

The LIST PLOT command in Example 4-5 generates a two-axis plot of CPU-BUSY-TIME for 12 CPUs. The measurement ran from 11:20 a.m. to 1:20 p.m. A 30-minute interval was used to collect the data. The alphabetic (plot) characters on the plot correspond to the CPUs listed in the key below the plot. Each plot character represents the value of a CPU-BUSY-TIME counter at the time shown on the vertical axis.

Example 4-5. Typical Two-Axis Plot of CPU-BUSY-TIME

```
24+ LIST PLOT

        0 :::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 -   I   JFH A  LGE         K     C       B      D                -
11:50:00 -   I  GAF     JE C  KL              D B                         -
12:20:00 -    I  GF AJ    L   C   E    D  K                               -
12:50:00 -       H  A   J E            F  BK        D C                   -
13:20:00 -H IEL      A CG       K    D        F                           B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 2.152                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 0
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 3
      E: CPU-BUSY-TIME                 Cpu 4
      F: CPU-BUSY-TIME                 Cpu 5
      G: CPU-BUSY-TIME                 Cpu 6
      H: CPU-BUSY-TIME                 Cpu 7
      I: CPU-BUSY-TIME                 Cpu 8
      J: CPU-BUSY-TIME                 Cpu 9
      K: CPU-BUSY-TIME                 Cpu 10
      L: CPU-BUSY-TIME                 Cpu 11
```

Example 4-6 on page 4-17 shows how to use the INTERVAL attribute of the LIST PLOT command to divide the measurement data into different time intervals for plotting (1-hour intervals in the figure). The INTERVAL specified in the LIST PLOT command must be equal to or greater than the collection interval used to collect the data, and it should be a multiple of the collection interval. When you specify an INTERVAL smaller than the collection interval, the plot is blank.

Example 4-6. Two-Axis Plot Showing One-Hour Intervals

```
21+ LIST PLOT, INTERVAL 1 HOUR
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 -   I     A G    E L    K  C            B  D                     -
12:20:00 -     I  H A GJ      E B        C K D                            -
13:20:00 -H IEL      A CG       K    D        F                           B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 2.152                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 0
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 3
      E: CPU-BUSY-TIME                 Cpu 4
      F: CPU-BUSY-TIME                 Cpu 5
      G: CPU-BUSY-TIME                 Cpu 6
      H: CPU-BUSY-TIME                 Cpu 7
      I: CPU-BUSY-TIME                 Cpu 8
      J: CPU-BUSY-TIME                 Cpu 9
      K: CPU-BUSY-TIME                 Cpu 10
      L: CPU-BUSY-TIME                 Cpu 11
```

A plot character can be absent from a display line because the value for its associated counter fell outside the plot range or was overwritten by another letter contending for its place in the display. To possibly improve display resolution, narrow the plot range. See Changing the Scale on page 4-22. You can also avoid this kind of overwriting by generating plots of individual entities.

To change a two-axis plot to a bar graph, use the TIME-BASE attribute in the LIST PLOT command, as shown in Example 4-7 on page 4-18.

Example 4-7. Two-Axis Plot Converted to Bar Graph

```
26+ LIST PLOT, TIME-BASE OFF
         0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+::::100
       A ***********|                                                -
       B *********************************|                          -
       C *******************************|                            -
       D ******************************************|                 -
       E *******************|                                        -
       F ***************|                                            -
       G ************|                                               -
       H *********|                                                  -
       I *****|                                                      -
       J *************|                                              -
       K ******************************|                             -
       L *****************|                                          -
         0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+::::100

      Min value = 8.453               Max value = 60.852

      A: CPU-BUSY-TIME                Cpu 0
      B: CPU-BUSY-TIME                Cpu 1
      C: CPU-BUSY-TIME                Cpu 2
      D: CPU-BUSY-TIME                Cpu 3
      E: CPU-BUSY-TIME                Cpu 4
      F: CPU-BUSY-TIME                Cpu 5
      G: CPU-BUSY-TIME                Cpu 6
      H: CPU-BUSY-TIME                Cpu 7
      I: CPU-BUSY-TIME                Cpu 8
      J: CPU-BUSY-TIME                Cpu 9
      K: CPU-BUSY-TIME                Cpu 10
      L: CPU-BUSY-TIME                Cpu 11
```

Each bar in the graph in Example 4-7 corresponds to a CPU listed in the key following the graph and represents the average value of the counter for that CPU during the measurement interval.

You can combine information from more than one counter type to compare values or track trends. For example, this command sequence adds the FILE-BUSY-TIME counter for each file measured to the CPU-BUSY-TIME plot definition:

```
13+ LIST FILE *
      .
```

Display of all files appears here.

```
      .
14+ ADD PLOT FILE-BUSY-TIME
MEAS 3008 WARNING. Plot became full during ADD; there can be
                   a maximum of 26 entities; some entities
                   were not ADDed.
For more information about error 3008 type HELP 3008
15+ INFO PLOT *
Add measurement $SPOOL.PERF.DATA1
--A-- List Cpu 0
      Add plot CPU-BUSY-TIME
--B-- List Cpu 1
      Add plot CPU-BUSY-TIME
--C-- List Cpu 2
      Add plot CPU-BUSY-TIME
```

```
--D-- List Cpu 3
      Add plot CPU-BUSY-TIME
--E-- List Cpu 4
      Add plot CPU-BUSY-TIME
--F-- List Cpu 5
      Add plot CPU-BUSY-TIME
--G-- List Cpu 6
      Add plot CPU-BUSY-TIME
--H-- List Cpu 7
      Add plot CPU-BUSY-TIME
--I-- List Cpu 8
      Add plot CPU-BUSY-TIME
--J-- List Cpu 9
      Add plot CPU-BUSY-TIME
--K-- List Cpu 10
      Add plot CPU-BUSY-TIME
--L-- List Cpu 11
      Add plot CPU-BUSY-TIME
--M-- List File $SYSTEM.SYSTEM.DMON (9, 0, 4)
      Add plot FILE-BUSY-TIME
--N-- List File $SYSTEM.SYSTEM.NSSMON (9, 0, 15)
      Add plot FILE-BUSY-TIME
--O-- List File $SYSTEM.SYSTEM.USERID (9, 48, 1)
      Add plot FILE-BUSY-TIME
--P-- List File $SYSTEM.SYSTEM.DMON (6, 0, 2)
      Add plot FILE-BUSY-TIME
--Q-- List File $SYSTEM.SYSTEM.FULIST (6, 0, 56)
      Add plot FILE-BUSY-TIME
--R-- List File $SYSTEM.SYSTEM.NSSMON (6, 0, 77)
      Add plot FILE-BUSY-TIME
--S-- List File $SYSTEM.SYSTEM.USERID (6, 31, 2)
      Add plot FILE-BUSY-TIME
--T-- List File $SYSTEM.SYSTEM.DMON (0, 0, 14)
      Add plot FILE-BUSY-TIME
--U-- List File $SYSTEM.SYSTEM.SPOOL (0, 0, 16)
      Add plot FILE-BUSY-TIME
--V-- List File $SYSTEM.SYSTEM.CSPOOL (0, 0, 19)
      Add plot FILE-BUSY-TIME
--W-- List File $SYSTEM.SYSTEM.NULL (1, 0, 16)
      Add plot FILE-BUSY-TIME
--X-- List File $SYSTEM.SYSTEM.IMON (1, 0, 18)
      Add plot FILE-BUSY-TIME
--Y-- List File $SYSTEM.SYSTEM.DMON (1, 0, 20)
      Add plot FILE-BUSY-TIME
--Z-- List File $SYSTEM.SYSTEM.NETACL (0, 0, 41)
      Add plot FILE-BUSY-TIME
```

Because a plot definition can include only 26 entity-counter pairs, only the first 14 files displayed can be added to the 12 CPU entity-counter pairs already defined for the plot. Error message 3008 indicates this restriction.

To order the files so the 14 files with the highest FILE-BUSY-TIME values are added to the plot definition, use the BY clause of the LIST command. In this example, the DELETE PLOT command deletes all the FILE-BUSY-TIME counters from the plot definition. The LIST FILE command orders the files by FILE-BUSY-TIME, from busiest to least busy. The ADD PLOT command adds the 14 busiest files to the plot definition:

```
16+ DELETE PLOT FILE-BUSY-TIME (*)
17+ LIST FILE *, BY FILE-BUSY-TIME
      .
      .
```

(Display of all files from busiest to least busy appears here.)

```
      .
      .
18+ ADD PLOT FILE-BUSY-TIME
MEAS 3008 WARNING. Plot became full during ADD; there can be a
maximum of 26 entities; some entities were not ADDed
For more information about error 3008 type HELP 3008
19+ INFO PLOT *
Add measurement $SPOOL.PERF.DATA1
--A-- List Cpu 0
      Add plot CPU-BUSY-TIME
--B-- List Cpu 1
      Add plot CPU-BUSY-TIME
--C-- List Cpu 2
      Add plot CPU-BUSY-TIME
--D-- List Cpu 3
      Add plot CPU-BUSY-TIME
--E-- List Cpu 4
      Add plot CPU-BUSY-TIME
--F-- List Cpu 5
      Add plot CPU-BUSY-TIME
--G-- List Cpu 6
      Add plot CPU-BUSY-TIME
--H-- List Cpu 7
      Add plot CPU-BUSY-TIME
--I-- List Cpu 8
      Add plot CPU-BUSY-TIME
--J-- List Cpu 9
      Add plot CPU-BUSY-TIME
--K-- List Cpu 10
      Add plot CPU-BUSY-TIME
--L-- List Cpu 11
      Add plot CPU-BUSY-TIME
--M-- List File $SYSTEM.SYSTEM.SQLCI2 (4, 99, 1)
      Add plot FILE-BUSY-TIME
--N-- List File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      Add plot FILE-BUSY-TIME
--O-- List File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      Add plot FILE-BUSY-TIME
--P-- List File $SYSTEM.SYSTEM.USERID (2, 174, 1)
      Add plot FILE-BUSY-TIME
--Q-- List File $SYSTEM.SYSTEM.TEDPROFL (10, 151, 3)
      Add plot FILE-BUSY-TIME
--R-- List File $SYSTEM.SYSTEM.USERID (7, 80, 3)
      Add plot FILE-BUSY-TIME
--S-- List File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      Add plot FILE-BUSY-TIME
--T-- List File $SYSTEM.SYSTEM.SQLCI2 (2, 180, 1)
```

```
      Add plot FILE-BUSY-TIME
--U-- List File $SYSTEM.SYSTEM.XRLSTOOL (3, 166, 6)
      Add plot FILE-BUSY-TIME
--V-- List File $SYSTEM.SYSTEM.USERID (2, 96, 6)
      Add plot FILE-BUSY-TIME
--W-- List File $SYSTEM.SYSTEM.USERID (8, 84, 6)
      Add plot FILE-BUSY-TIME
--X-- List File $SYSTEM.SYSTEM.USERID (7, 73, 6)
      Add plot FILE-BUSY-TIME
--Y-- List File $SYSTEM.SYSTEM.USERID (11, 145, 2)
      Add plot FILE-BUSY-TIME
--Z-- List File $SYSTEM.SYSTEM.USERID (4, 101, 3)
      Add plot FILE-BUSY-TIME
```

Example 4-8 shows the two-axis plot of the combined CPU-BUSY-TIME and FILE-BUSY-TIME data defined in the previous example.

Example 4-8. Plot of CPU-BUSY-TIME and FILE-BUSY-TIME Data

```
34+ LIST PLOT
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 -   I   JFH A  LGE        K       C      B       D        QP     -
11:50:00 -   I GAF     JE C  KL                D B              Y  R      -
12:20:00 -    I GF AJ    L   C   E    D  K                      ZUT     M -
12:50:00 -      H  A   J E            F  BK      D C            W S    ON -
13:20:00 -H IEL     A CG      K    D         F                            B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 2.152                Max value = 100.000

      A: CPU-BUSY-TIME             Cpu 0
      B: CPU-BUSY-TIME             Cpu 1
      C: CPU-BUSY-TIME             Cpu 2
      D: CPU-BUSY-TIME             Cpu 3
      E: CPU-BUSY-TIME             Cpu 4
      F: CPU-BUSY-TIME             Cpu 5
      G: CPU-BUSY-TIME             Cpu 6
      H: CPU-BUSY-TIME             Cpu 7
      I: CPU-BUSY-TIME             Cpu 8
      J: CPU-BUSY-TIME             Cpu 9
      K: CPU-BUSY-TIME             Cpu 10
      L: CPU-BUSY-TIME             Cpu 11
      M: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.SQLCI2 (4, 99, 1)
      N: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      O: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      P: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (2, 174, 1)
      Q: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.TEDPROFL (10, 151, 3)
      R: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (7, 80, 3)
      S: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.SQLCI2 (8, 60, 1)
      T: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.SQLCI2 (2, 180, 1)
      U: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.XRLSTOOL (3, 166, 6)
      V: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (2, 96, 6)
      W: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (8, 84, 6)
      X: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (7, 73, 6)
      Y: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (11, 145, 2)
      Z: FILE-BUSY-TIME            File $SYSTEM.SYSTEM.USERID (4, 101, 3)
```

Changing the Scale

Changing the scale of a graph can make its data more distinct and accessible. To change the scale on a two-axis plot or a bar graph, use the SCALE-FROM and SCALE-TO plot attributes of the SET PLOT or LIST PLOT command.

The default scale is 0 to 100. To plot values larger than 100, you must change the scale default. The minimum and maximum data values shown in each plot appear following each graph. These values can help you select different scale values.

By default, the time intervals on the Y-axis of two-axis plots are the collection intervals. To change the time interval, use the INTERVAL clause of the LIST PLOT command. To change the beginning or ending times of the time axis, use the FROM and TO clauses of the LIST PLOT command. See Changing the Plot Window on page 4-27.

Example 4-9 shows a two-axis plot of CPU-BUSY-TIME for the five busiest CPUs on the system. These CPUs were identified by listing all 12 CPUs by CPU-BUSY-TIME, then adding each of the top five CPUs to the plot definition.

Example 4-9. Two-Axis Plot of Five Busiest CPUs

```
68+ LIST PLOT, INTERVAL 1 HOUR
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 -               E      D  C              B A                     -
12:20:00 -                   E B        C D A                             -
13:20:00 -   E        C        D    A                                     B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 5.838                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 3
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 10
      E: CPU-BUSY-TIME                 Cpu 4
```

Example 4-10 uses the minimum and maximum values listed following the graph in Example 4-9 to change the report window. The plot is scaled from 5 to 70 rather than from 0 to 100.

Example 4-10. Plot of Five Busiest CPUs, Narrowed Report Window

```
70+ LIST PLOT, INTERVAL 1 HOUR, SCALE-FROM 5, SCALE-TO 70
      5.00:::::+:::18.0:::::+:::31.0::::::+:::44.0:::::+:::57.0:::::+:::70.0
11:20:00 -                  E           D   C                     B   A    -
12:20:00 -                       E   B               C  D  A              -
13:20:00 E               C           D       A                             -
      5.00:::::+:::18.0:::::+:::31.0::::::+:::44.0:::::+:::57.0:::::+:::70.0

      Min value = 5.838                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 3
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 10
      E: CPU-BUSY-TIME                 Cpu 4
```

Changing the Orientation

For a two-axis plot, changing the orientation means changing the time interval from the vertical axis (the default) to the horizontal axis.

For a bar graph, changing the orientation means changing the base of the bars from the vertical axis, running horizontally (the default), to the horizontal axis, running vertically.

Use the VERT-BASE attribute of the LIST PLOT or SET PLOT command to set the orientation for a plot. VERT-BASE ON (the default) specifies a vertical orientation. VERT-BASE OFF specifies a horizontal orientation.

Example 4-11 on page 4-24 shows two-axis plots in both vertical (default) and horizontal orientations. Example 4-12 on page 4-25 shows bar graphs in both vertical (default) and horizontal orientations.

Example 4-11. Changing the Orientation of a Two-Axis Plot

```
56+ LIST PLOT, VERT-BASE ON-- Vertical Orientation (Default)
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0::::+::::100
11:20:00 -               E         D      C       B      A               -
11:50:00 -             E C  D                 A B                        -
12:20:00 -         B        C   E      A D                               -
12:50:00 -              E                BD        A C                   -
13:20:00 -   E        C      D    A                                      B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0::::+::::100

      Min value = 5.838                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 3
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 10
      E: CPU-BUSY-TIME                 Cpu 4

57+ LIST PLOT, VERT-BASE OFF-- Horizontal Orientation
     100 ---------B-
    95.0 -         -
    90.0 -         -
    85.0 -         -
    80.0 -         -
    75.0 -A        -
    70.0 -      C  -
    65.0 -B B   A  -
    60.0 -  A      -
    55.0 -C   D B  -
    50.0 -    A    -
    45.0 -D       A-
    40.0 -    E    -
    35.0 -  D C   D-
    30.0 -E C      -
    25.0 -  E   E C-
    20.0 -    B    -
    15.0 -         -
    10.0 -        E-
    5.00 -         -
       0 -----------
   11:20:00  20  20
           50  50

      Min value = 5.838                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 3
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 10
      E: CPU-BUSY-TIME                 Cpu 4
```

Example 4-12. Changing the Orientation of a Bar Graph

```
59+ LIST PLOT, VERT-BASE ON, TIME-BASE OFF-- Vertical Orientation
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
       A ******************************************|                      -
       B *********************************|                               -
       C *******************************|                                 -
       D ******************************|                                  -
       E *******************|                                             -
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 27.771            Max value = 60.852

      A: CPU-BUSY-TIME              Cpu 3
      B: CPU-BUSY-TIME              Cpu 1
      C: CPU-BUSY-TIME              Cpu 2
      D: CPU-BUSY-TIME              Cpu 10
      E: CPU-BUSY-TIME              Cpu 4

60+ LIST PLOT, VERT-BASE OFF, TIME-BASE OFF-- Horizontal Orientation
     100 -----------------------------------------
    95.0 -                                       -
    90.0 -                                       -
    85.0 -                                       -
    80.0 -                                       -
    75.0 -                                       -
    70.0 -                                       -
    65.0 - -                                     -
    60.0 -| |                                    -
    55.0 -| |                                    -
    50.0 -| |-                                   -
    45.0 -| | |- -                               -
    40.0 -| | | | |                              -
    35.0 -| | | | |                              -
    30.0 -| | | | |-                             -
    25.0 -| | | | | |                            -
    20.0 -| | | | | |                            -
    15.0 -| | | | | |                            -
    10.0 -| | | | | |                            -
    5.00 -| | | | | |                            -
       0 -| | | | | |-----------------------------
           A B C D E

      Min value = 27.771            Max value = 60.852

      A: CPU-BUSY-TIME              Cpu 3
      B: CPU-BUSY-TIME              Cpu 1
      C: CPU-BUSY-TIME              Cpu 2
      D: CPU-BUSY-TIME              Cpu 10
      E: CPU-BUSY-TIME              Cpu 4
```

Changing the Density

For a two-axis plot, setting the density means deciding whether to display one point for each plotted value at each time interval (the default) or to display asterisks between the time axis and the lowest value at each time interval.

For a bar graph, setting the density means deciding whether to display narrow bars or wide bars. Narrow bars are one character wide (the default). Wide bars are two to six characters wide.

To set the density for a plot, use the WIDE-ITEM option of the SET PLOT or LIST PLOT command. WIDE-ITEM OFF (the default) fills in less of the plot than WIDE-ITEM ON.

In Example 4-13, the density of the two-axis plot in Example 4-9 on page 4-22 is changed by setting WIDE-ITEM ON. Each line in this plot represents the value of the counter at the time shown on the Y axis. Asterisks fill the area from zero to the least active entity at each time interval.

Example 4-13. Changing the Density of a Two-Axis Plot

```
62+ LIST PLOT, WIDE-ITEM ON
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 *****************E        D      C       B      A                -
11:50:00 ***************E C  D                A B                         -
12:20:00 **********B         C   E     A  D                               -
12:50:00 ****************E                BD      A C                     -
13:20:00 ****E        C        D   A                                      B
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 5.838                Max value = 100.000

      A: CPU-BUSY-TIME                 Cpu 3
      B: CPU-BUSY-TIME                 Cpu 1
      C: CPU-BUSY-TIME                 Cpu 2
      D: CPU-BUSY-TIME                 Cpu 10
      E: CPU-BUSY-TIME                 Cpu 4
```

In Example 4-14 on page 4-27, the two-axis plot in Example 4-13 is converted to a bar graph (TIME-BASE OFF). WIDE-ITEM ON is still in effect. The five bars represent the average values of the CPU-BUSY-TIME counter for each CPU.

Example 4-14. Changing the Density of a Bar Graph

```
63+ LIST PLOT, VERT-BASE OFF, TIME-BASE OFF, WIDE-ITEM ON
     100 -----------------------------------------
    95.0 -                                       -
    90.0 -                                       -
    85.0 -                                       -
    80.0 -                                       -
    75.0 -                                       -
    70.0 -                                       -
    65.0 - -----                                 -
    60.0 -|     |                                -
    55.0 -|     |                                -
    50.0 -|     |-----                           -
    45.0 -|     |     |----- -----               -
    40.0 -|     |     |     |     |              -
    35.0 -|     |     |     |     |              -
    30.0 -|     |     |     |     |-----         -
    25.0 -|     |     |     |     |     |        -
    20.0 -|     |     |     |     |     |        -
    15.0 -|     |     |     |     |     |        -
    10.0 -|     |     |     |     |     |        -
    5.00 -|     |     |     |     |     |        -
       0 -|     |     |     |     |     |---------
             A     B     C     D     E

      Min value = 27.771                Max value = 60.852

      A: CPU-BUSY-TIME                  Cpu 3
      B: CPU-BUSY-TIME                  Cpu 1
      C: CPU-BUSY-TIME                  Cpu 2
      D: CPU-BUSY-TIME                  Cpu 10
      E: CPU-BUSY-TIME                  Cpu 4
```

Changing the Plot Window

Specifying a plot window directs MEASCOM to read only records written to the data file within the specified period. Because you are viewing a subset of the available data, the plots displayed using the plot window are typically different from those displayed by default.

Use the TO, FROM, and FOR clauses of the LIST PLOT command to adjust the time window of the plot.

Note. Specifying a time window in the LIST *`entity-type`* command affects the entities included in the plot, not its time window.

Example 4-15 on page 4-28 shows a two-axis plot of data collected between 11:00 a.m. and 12:00 p.m. for the five busiest CPUs. The first interval shown on the plot corresponds to the starting time of the measurement.

Example 4-15. Five Busiest CPUs, One-Hour Time Window

```
73+ LIST PLOT, FROM 11:00, TO 12:00
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100
11:20:00 -               E         D      C      B      A             -
11:50:00 -             E C  D                  A B                    -
         0:::::+:::20.0:::::+:::40.0:::::+:::60.0:::::+:::80.0:::::+::::100

      Min value = 23.915                Max value = 74.388

      A: CPU-BUSY-TIME                  Cpu 3
      B: CPU-BUSY-TIME                  Cpu 1
      C: CPU-BUSY-TIME                  Cpu 2
      D: CPU-BUSY-TIME                  Cpu 10
      E: CPU-BUSY-TIME                  Cpu 4
```

Plotting Execution Modes

Reports for the PROCESSH (process histogram) entity type show what percentage of code samples executed in each of three execution modes:

- TNS mode
- Accelerated mode
- TNS/R native mode

Example 4-16 shows a PROCESSH report for a process that runs in both the TNS and accelerated modes.

Example 4-16. Typical PROCESSH Report Showing TNS and Accelerated Modes

```
Process                     Program $SYSTEM.SYS02.MEASCOM
CPU 7    Pin    37          Priority 140  Userid     2,63  Creatorid  255,255
Local System \HOME          From    17 Aug 1996, 16:51:00   For 45.6 seconds

Total Samples for the process =          786 #

Code-Map  UC.0        Samples     257 #       32.69 %

                                                        Samples in     Percent
     Code-Range Name          Accel       TNS          Code-Map       of Total
-------------------------- --------- --------- ------------ -----------
FORMAT                        40 #        25 #         65 #        8.26 %
PRINT^RESULTS                 50 #         2 #         52 #        6.62 %
FINDT                         85 #        55 #        140 #       17.81 %
```

Example 4-17 on page 4-29 shows a PROCESSH report for a process that runs in TNS/R native mode.

Example 4-17. Typical PROCESSH Report Showing TNS/R Native Mode

```
Process                  Program $SYSTEM.SYS02.MEASCOM
CPU 8    Pin    10       Priority 140  Userid     2,63  Creatorid  255,255
Local System \BANK       From   17 Aug 1996, 16:51:00   For 53.2 seconds

Total Samples for the process =         799 #

Code-Map  UCr            Samples     137 #       17.43 %

                                                 Samples in   Percent
     Code-Range Name        TNSR                 Code-Map     of Total
-------------------------- ---------            ---------     ---------------
FORMAT                         65 #             65 #          8.26 %
PRINT^RESULTS                  52 #             52 #          6.61 %
FINDT                          20 #             20 #          2.54 %
```

To display execution-mode statistics in a plot, use four options of the ADD PLOT command:

ACCEL-BUSY-SAMPLES	Adds accelerated code samples to the plot
CODE-RANGE	Adds code samples obtained in all execution modes to the plot, combining all three modes into one plot item for each procedure
TNS-BUSY-SAMPLES	Adds TNS code samples to the plot
TNSR-BUSY-SAMPLES	Adds TNS/R native code samples to the plot

You can use the ADD PLOT options separately or together. With TIME-BASE ON, each counter is plotted separately, as in other Measure plots. With TIME-BASE OFF, the counters can be shown separately, together, or both. For example, to generate a plot from the report shown in Example 4-16 on page 4-28, you might enter:

```
75+ ADD PLOT CODE-RANGE
76+ ADD PLOT TNS-BUSY-SAMPLES
77+ SET PLOT TIME-BASE OFF
78+ LIST PLOT
```

Example 4-18 on page 4-30 shows the resulting plot. The asterisk (*) represents accelerated code samples, and the plus sign (+) represents TNS code samples. Statistics for TNS samples are displayed in two ways:

- The ADD PLOT CODE-RANGE command plots accelerated and TNS samples as one item.
- The ADD PLOT TNS-BUSY-SAMPLES command plots TNS samples separately.

Example 4-18. Plotting Execution Modes—TNS and Accelerated Code Samples

```
           0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
         A ***********************+++++++++++++++++|                  -
         B ******************************++|                          -
         C ***************************************************++++++++-
         D ++++++++++++++|                                            -
         E +|                                                         -
         F +++++++++++++++++++++++++++++++++|                         -
           0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100

               MIN VALUE =  2.000           MAX VALUE =  88.000

A: FORMAT                              Processh $SYSTEM.SYS02.MEASCOM (7,37)
B: PRINT^RESULT                        Processh $SYSTEM.SYS02.MEASCOM (7,37)
C: FINDT                               Processh $SYSTEM.SYS02.MEASCOM (7,37)
D: FORMAT                              Processh $SYSTEM.SYS02.MEASCOM (7,37)
   (TNS)
E: PRINT^RESULT                        Processh $SYSTEM.SYS02.MEASCOM (7,37)
   (TNS)
F: FINDT                               Processh $SYSTEM.SYS02.MEASCOM (7,37)
   (TNS)
```

These commands generate the plot shown in Example 4-19. The pound sign (#) represents TNS/R native code.

```
85+ ADD PLOT TNSR-BUSY-SAMPLES
86+ SET PLOT TIME-BASE OFF
87+ LIST PLOT
```

Example 4-19. Plotting Execution Modes—TNS/R Native Code Samples

```
           0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100
         A #####################################|                     -
         B ##############################|                            -
         C ######|                                                    -
           0::::+:::20.0::::+:::40.0::::+:::60.0::::+:::80.0::::+:::100

               MIN VALUE =  2.000           MAX VALUE =  65.000

A: AC1^CKPT^GET^BLOCK                  Processh $SYSTEM.SYS02.TSYSDP2 (7,20)
   (TNSR)
B: AC1^RECMOVE^DATA                    Processh $SYSTEM.SYS02.TSYSDP2 (7,20)
   (TNSR)
C: AC1^REDOCHAIN^PURGE                 Processh $SYSTEM.SYS02.TSYSDP2 (7,20)
   (TNSR)
```

Producing Structured Files of Measurement Data

The Enform query product and the NonStop SQL/MP command interpreter (SQLCI) can access Measure data files only after they are converted to structured files and a data dictionary that describes the structure of the file is created.

To format and write Measure data to structured data files and create a data dictionary using MEASCOM:

1. Enter the SET REPORT command to set the format for structured report files.
2. Enter the LIST *entity-type* or LISTALL *entity-type* command to load the structured files.
3. Exit from MEASCOM and use the RUN DDL command to build the data dictionary.

Step 1: Produce Structured Report Files

In MEASCOM, to produce structured report files, use the SET REPORT command followed by a LIST command. For example:

```
45> MEASCOM
MEASURE Performance Monitor - T9086G10 - (16DEC03) - \HATI
(C)1986 Tandem (C)2003 Hewlett Packard Development Company, L.P.

1+ SET REPORT FORMAT STRUCTURED
2+ LIST PROCESS *
```

The REPORT attributes and LIST and LISTALL clauses you use can affect the content of structured files.

Table 4-3. Command Option Effects on Data Written to Structured Files (page 1 of 2)

Command Option	Effect
REPORT FORMAT	Determines whether counters are written to the structured file.
REPORT LOADID	Determines the LOADID field value of the records written to the structured file.
REPORT STYLE	Determines if ZMS-style or legacy-style record templates and file names will result. (Measure G11 and later)
REPORT RATE	No effect. Uninterpreted counter values are always written to the structured file.
REPORT TOTALS	Determines whether counter totals are written to the structured file.
REPORT ZERO-REPORTS	Determines whether entities in which all counter values are zero are written to the structured file.

Table 4-3. Command Option Effects on Data Written to Structured Files (page 2 of 2)

Command Option	Effect
REPORT ZERO-VALUES	No effect. Counter values of zero are always written to the structured file as zero.
LIST/LISTALL FROM, TO, FOR clauses	Determines which records are written to the structured file.
LIST/LISTALL BY clause	Sorts entities before data is written. The order of entities in the structured files does not matter.
LIST/LISTALL IF clause	Determines which records are written to the structured file.

Step 2: Load the Structured Files

For each LIST *entity-type* or LISTALL *entity-type* command, MEASFH searches the current subvolume for a structured file named for the entity type (CPU, PROCESS, DISC, and so on). If the file exists, MEASFH adds the records for the specified entities to the file. If a structured file for a specified entity type does not exist on the subvolume, MEASFH creates a file for it and adds its records to the newly created file.

The fields within the records of the structured files contain uninterpreted counter values. That is, busy counters are in microseconds busy, rates are in number of occurrences, and so on. For a description of the record format and content for each Measure entity type, see the *Measure Reference Manual*.

Load the structured files using the LIST *entity-type* or LISTALL *entity-type* command. For example:

```
2+ ADD MEASUREMENT $PERF.DATA.JUN04
3+ SET REPORT FORMAT STRUCTURED
4+ LIST PROCESS SYSTEM-PROCESSES
5+ LIST PROCESS $*.*.BILLING
```

In this example, the ADD MEASUREMENT command makes measurement data file $PERF.DATA.JUN04 accessible to MEASCOM. The SET REPORT command indicates to write counters to a structured file. The first LIST command creates the structured file PROCESS (if it does not already exist), then writes the counter values for all system processes to the file. The second LIST command writes the counter values for the PROCESS $*.*.BILLING entity in $PERF.DATA.JUN04 to the newly created file PROCESS.

When MEASFH loads structured files, it does not check for duplicate records but lists all records separately in the structured file. For example:

```
6+ LIST PROCESS SYSTEM-PROCESSES
7+ LIST PROCESS $*.*.*
```

These two commands cause all records for the system processes to appear twice in the structured file. To avoid duplicate records:

1. Build a command (OBEY) file containing the appropriate LIST *`entity-type`* or LISTALL *`entity-type`* commands.
2. Load the structured report files by executing the OBEY file from within MEASCOM.

Measure appends new records to existing structured files. To purge existing files before writing out new values, issue this command or add it to your OBEY file:

```
42+ RUN FUP PURGE filename [, filename ] ...!
```

Creating Records for Different Time Windows

Each record in the structured report file describes the performance of a single entity over the specified report window. For example, for the CPU file, each record describes the performance of a single CPU over the report window. For the PROCESS file, each record describes the performance of a single process over the report window.

To create separate records for different time windows, use the LOADID clause in the MEASCOM LIST *`entity-type`* or LISTALL *`entity-type`* command to identify each window. You can also use this clause to identify the different measurements whose records you want to write to the structured files. Finally, you can use the LOADID clause to identify sets of records for the same entity type. For example, this LOADID clause identifies two sets of process records:

```
8+ ADD MEASUREMENT $PERF.DATA.JUN04
9+ LIST PROCESS SYSTEM-PROCESSES, LOADID SYSPROC
10+ LIST DISC $DATA1
11+ LIST PROCESS $*.*.BILLING, LOADID BILLING
```

Note. When you use the LOADID clause with the LISTALL command, the LOADID field contains the interval number, in ASCII, associated with the time the record was copied to the data file. LOADID is no longer unique per LISTALL command. Therefore, if you are performing Enform queries that use LOADID as a unique identifier, you must use the LOAD-ID.PREFIX-ID format or a LOADID of six or more characters to maintain the uniqueness of the field. To reference any part of the LOAD-ID redefine, you must create a new dictionary:

```
VOLUME your-subvolume
DDL /IN $SYSTEM.SYSnn.MEASDDLS/ DICT!
```

Step 3: Build the Data Dictionary

The DDL source file, $SYSTEM.SYS*nn*.MEASDDLS, contains RECORD statements that describe the record structure for each structured report file of each entity type. To find the subvolume name for SYS*nn* on your system, use the TACL STATUS * command. SYS*nn* is the subvolume that contains the OSIMAGE file. Use this subvolume name in the TACL DDL command that builds the dictionary. For example:

```
45> DDL /IN $SYSTEM.SYSnn.MEASDDLS, OUT $S/ DICT!
```

Or issue this command from MEASCOM:

```
23+ RUN DDL /IN $SYSTEM.SYSnn.MEASDDLS, OUT $S/ DICT!
```

The DDL command creates a dictionary in your current subvolume and writes to an out file the records entered into it. The OUT option sends the list of records to the default printer. There should be 14 files on your subvolume, each beginning with DICT.

Generating Reports Using Enform and SQL/MP Products

For a description of how to produce reports using Enform queries, see Appendix A, Creating an Enform Report From Measure Data. For more Enform information, see the *ENFORM User's Guide* and the *ENFORM Reference Manual*.

To generate reports using the NonStop SQL/MP Report Writer:

1. Use the SQLCI CONVERT utility to load the data in the structured file into an SQL table.
2. Use the NonStop SQL/MP Report Writer to perform any report writing function on the data.

For More Information

- For brief procedures for converting a Measure data file to an SQL table, see Appendix C, Loading Measure Data Into an SQL Table.
- For detailed information about the CONVERT utility, see the *NonStop SQL/MP Reference Manual.*
- For information about the NonStop SQL/MP Report Writer, see the *NonStop SQL/MP Report Writer Guide*.

Loading Data From Different Systems to Common Files

Data from multiple systems running the same Measure PVU can be combined in a common set of structured files. However, in general, report processing time increases if a query must sort through more records to find the ones pertinent to the request. To distinguish records from different data sets in queries and reports, use LOADID.

If systems are running different Measure PVUs, combining data is more difficult:

- For legacy-style data (Measure G10 and earlier), records can be combined only if the record format for the entity is unchanged between the PVUs. If the record lengths differ, MEASFH does not write to the file. To combine data in this case:
  1. Create files in separate subvolumes.
  2. Use FUP COPY with the truncate or pad options to load the additional data from one subvolume into the files of the other subvolume.
- For ZMS-style records (Measure G11 and later), MEASFH writes data from any PVU to an existing file in the format of the data already in the file. MEASFH truncates records or pad hdr, id, or ctr sections of the record with zeros as needed to fit the format of the existing file. The padding might place binary zeros in a field defined for ASCII data. Use a dictionary and files of the oldest PVU format your

systems run as the common format. Update to ZMS style only if your application or query is being updated to use new counter or identifier fields.

5 Defining Custom Counters

You can define custom counters to collect information about an application. Any counters you define are of the USERDEF entity type. To use these counters, you must first instrument the application to measure (that is, modify it so it bumps the counters).

Task 1: Instrument an Application

To instrument an application, you must determine the counters the application requires, choose a name for each counter, and specify its type. You must then decide how frequently the counters should be bumped and modify the application source code to bump the counters at the selected times.

Table 5-1 lists the four types of user-defined counters you can include in an application. For detailed descriptions, see the *Measure Reference Manual*.

Table 5-1. User-Defined Counters

Type	Name	Description
Accumulating	ACCUM	Same as a Measure incrementing or accumulating counter. You can specify either of two bump functions: ● INC increments the counter each time an event occurs. ● ADD adds a user-specified value to the counter whenever data is transferred.
Busy	BUSY	Same as a Measure busy counter. It measures the amount of time a resource is busy. You must specify two bump functions: ● SETBUSY starts the counter. ● RESETBUSY stops the counter.
Fixed Accumulating	FACCUM	Same as a Measure incrementing or accumulating counter. It is a 64-bit accumulating or incrementing counter type. You can specify either of two bump functions: ● INC increments the counter each time an event occurs. ● ADD adds a user-specified value to the counter whenever data is transferred.
Queue	QUEUE	Same as a Measure queue counter. It measures the time that elements spend on a queue. You must specify two bump functions: ● INCQUEUE increases the queue length by one. ● DEQUEUE decreases the queue length by one.

Step 1: Source in the MEASDECS File

The MEASDECS file contains the literal values that are passed to the MEASCOUNTERBUMP procedure. MEASDECS is located on $SYSTEM.SYS*nn*, where SYS*nn* is the subvolume that contains the OSIMAGE file.

Step 2: Declare Integer Variable, Array, and Offset

Declare the following:

error	An integer variable that holds the function values returned by the MEASCOUNTERBUMPINIT and MEASCOUNTERBUMP procedures.
counter	An integer array, eight words long, that contains the counter name. Declare one *counter* variable for each counter name.
offset	An integer variable that holds the location of the counter within the internal user-defined counter record. Declare one *offset* variable for each counter.

When you name counters:

- You can specify a separate name for each counter or a single name that refers to an array of counters. (In either case, the counter name is declared in the same way.)
- You can use uppercase and lowercase letters. Alphabetic characters are not case sensitive.
- Counters of the same name in the same program refer to the same counter, but counters of the same name in different programs refer to different counters.

Step 3: Initialize the Counter (Call MEASCOUNTERBUMPINIT)

1. Call MEASCOUNTERBUMPINIT to initialize the counter within the user-defined record.

   MEASCOUNTERBUMPINIT returns the counter offset in *offset*.

2. Pass *counter* as the first parameter and *offset* as the second parameter.

   If the counter specified is not part of a currently active measurement, MEASCOUNTERBUMPINIT returns the literal value ERR^UDCNOTPRESENT in *error* and -1 in *offset.* Always check for the ERR^UDCNOTPRESENT error.

Step 4: Bump the Counter (Call MEASCOUNTERBUMP)

1. Call MEASCOUNTERBUMP to bump the counter.
2. Pass *offset* as the first parameter and *bumptype* as the second parameter.

   bumptype determines how MEASCOUNTERBUMP bumps a counter. Literal values for *bumptype* are ADD, INC, SETBUSY, RESETBUSY, INCQUEUE, and DECQUEUE, as described in Example 5-1 on page 5-3.

When specifying *bumptype*:

- If you specify ADD, specify the value to add to the counter as the third parameter.
- If you specify SETBUSY, you must have a matching RESETBUSY call.
- If you specify INCQUEUE, you must have a matching DECQUEUE call.

△ **Caution.** Not balancing the busy and queue calls causes unpredictable counter values.

If you are using an array of counters, specify the index value of the counter as the fourth parameter. By default, the index value is zero to indicate the only (or first) value associated with the counter name.

If the counter specified is not part of a currently active measurement, MEASCOUNTERBUMP returns the literal value ERR^UDCNOTPRESENT in *error*. Always check for this error.

For time-critical applications, to avoid unnecessary calls to the MEASCOUNTERBUMP procedure, set a flag in your application to indicate that a counter is not active. Testing the flag before you call MEASCOUNTERBUMP costs fewer CPU cycles than calling MEASCOUNTERBUMP and finding that the counter is not active.

Sample TAL Program

The TAL program in Example 5-1 bumps three counters:

- TRANSACTIONS, an accumulating counter array containing two elements
- TIMEPER, a busy counter
- WAITING, a queue counter

This sample program assumes that the measurement is already running.

Example 5-1. TAL Application Containing User-Defined Counters (page 1 of 3)

```
?PAGE "GLOBAL DECLARATIONS"
?SYMBOLS, INSPECT
?NOMAP, NOLMAP, NOCODE

! error variable
int error := 0;
int .array[0:2];
string .sarray := @array '<<' 1;

! terminal name and number
int term[0:11];
int term^number;

! message variable
int .msg[0:40*["  "]];
string .smsg := @msg '<<' 1;

?NOLIST
?SOURCE $SYSTEM.SYS20.MEASDECS  ! check your subvolume name
```

Example 5-1. TAL Application Containing User-Defined Counters (page 2 of 3)

```
?SOURCE $SYSTEM.SYSTEM.EXTDECS (MYTERM, OPEN, DEBUG, WRITE,
?                NUMOUT, DELAY, CLOSE, MEASCOUNTERBUMP,
?                MEASCOUNTERBUMPINIT)
?LIST
?PAGE "MAIN PROCEDURE -- BILLING"
PROC billing MAIN;

BEGIN

   ! counter names and offsets
   int .transactions[0:7] := ["TRANSACTIONS    "];
   int .timeper[0:7] := ["TIMEPER         "];
   int .waiting[0:7] := ["WAITING         "];
   int transactions^offset;
   int timeper^offset;
   int waiting^offset;

   subproc printmsg;
   ! displays a message on the terminal
   begin
        CALL WRITE (term^number, msg, 80);
        IF <> THEN CALL DEBUG;
        smsg ':=' 80*[" "];
   end; !subproc

subproc checkiniterr;
   ! checks for errors returned by MEASCOUNTERBUMPINIT
   ! note, this sample program assumes an active measurement
   begin
   IF error AND (error <> ERR^UDCNOTPRESENT) THEN
     BEGIN
        CALL NUMOUT (array, error, 10, 5);
        smsg ':=' "MEASCOUNTERBUMPINIT error: " &
                                        sarray FOR 6 bytes;
        call printmsg;
     END;
   end; !subproc

  subproc checkerr;
   ! checks for errors returned by MEASCOUNTERBUMP
   ! note, this sample program assumes an active measurement
   begin
   IF error AND (error <> ERR^UDCNOTPRESENT) THEN
     BEGIN
        CALL NUMOUT (array, error, 10, 5);
        smsg ':=' "MEASCOUNTERBUMP error: " &
                                        sarray FOR 6 bytes;
        call printmsg;
     END;
   end; !subproc

    ! *******************
   ! begin main procedure
   ! *******************
```

Example 5-1. TAL Application Containing User-Defined Counters (page 3 of 3)

```
    ! open the terminal
   CALL MYTERM (term);
   CALL OPEN (term, term^number);
   IF <>  THEN  CALL DEBUG;

   ! determine location of each counter
   error := MEASCOUNTERBUMPINIT (transactions,
                                transactions^offset);
   call checkiniterr;

     error := MEASCOUNTERBUMPINIT (timeper, timeper^offset);
   call checkiniterr;

   error := MEASCOUNTERBUMPINIT (waiting, waiting^offset);
   call checkiniterr;

  ! increment both counters of the TRANSACTIONS array
   error := MEASCOUNTERBUMP (transactions^offset, INC);
   call checkerr;
   error := MEASCOUNTERBUMP (transactions^offset, INC,,1);
   call checkerr;

   ! start the TIMEPER busy counter
   error := MEASCOUNTERBUMP (timeper^offset, SETBUSY);
   call checkerr;

   ! increment the WAITING queue counter
   error := MEASCOUNTERBUMP (waiting^offset, INCQUEUE);
   call checkerr;
             .
             .  application code
             .
   ! stop the TIMEPER busy counter
   error := MEASCOUNTERBUMP (timeper^offset, RESETBUSY);
   call checkerr;

   ! decrement the WAITING queue counter
   error := MEASCOUNTERBUMP (waiting^offset, DECQUEUE);
   call checkerr;

   ! close the terminal
   CALL CLOSE (term^number);

END;
```

The array size for transactions (that is, the number of elements) is defined by the measurement configuration. For the configuration to measure this instrumented code, see Section 6, Creating a Custom Measurement Application.

Task 2: Measure the Application

Before starting a measurement, you must first create a measurement configuration that names the application to measure and the user-defined counters within the application to use. Each record written to the measurement data file contains information about a process that is running the application and the values of the user-defined counters bumped by that application.

Step 1: Specify the Processes Running the Application

To specify the processes running the application, use the ADD USERDEF and DELETE USERDEF commands. To display the application processes currently in the configuration, use the INFO USERDEF command.

The syntax for the ADD USERDEF command is:

```
ADD USERDEF process-spec
```

where *process-spec* identifies any process running in the application. *process-spec* can be any of:

```
{ *                                                }
{ cpu, pin                                         }
{ $process-name [ cpu, pin ]                       }
{ [ [ $device.]subvolume.]filename [ cpu, pin    ] }
```

For a complete description of the USERDEF entity specification, see the *Measure Reference Manual*.

Step 2: Specify the User-Defined Counters in the Application to Measure

To specify the user-defined counters in the application to be used to collect the application data, use the ADD COUNTER and DELETE COUNTER commands. To display the user-defined counters in the current configuration, use the INFO COUNTER command.

The syntax for the ADD COUNTER command is:

```
ADD COUNTER name, PROCESS process-spec, type [, ARRAY n ]
```

name

is the name of a user-defined counter in the application you plan to measure.

process-spec

is a process specification defined with a previous ADD USERDEF command. You must specify it in this ADD COUNTER command because the user-defined counter is associated with the processes that run the application containing the counter.

type

is a counter type: ACCUM, BUSY, or QUEUE. The type depends on the bump action the application uses to bump the counter. Specify *type* in the ADD COUNTER command to allocate the proper amount of space for the counter.

If the application treats the specified user-defined counter as an array of counters, specify *n* as the number of counters associated with the counter name.

To create a configuration that measures the BILLING application using the TRANSACTIONS counter:

```
8+ ADD USERDEF BILLING
9+ ADD COUNTER TRANSACTIONS, PROCESS BILLING, ACCUM, ARRAY 2
```

Considerations

- You must execute the ADD USERDEF command before its associated ADD COUNTER command.
- The process specification in the ADD COUNTER command must be identical to the process specification in a previous ADD USERDEF command.
- The process specification in the INFO COUNTER display does not reflect the USERDEF processes in the measurement configuration. Therefore, to restore the current configuration, you must save both the INFO USERDEF and INFO COUNTER displays.
- You cannot run concurrent measurements on an application containing user-defined counters. There can be only one measurement per process running the instrumented code. If you try to measure user-defined counters in an application with active counters, MEASCOM displays an error message when you next enter a LIST USERDEF command if ZERO-REPORTS is set to INCLUDE.
- To measure an application, the executing process must be measured. Therefore, a PROCESS entity is added automatically for any USERDEF entity in the measurement configuration. In this example, Measure automatically adds the PROCESS entity in the measurement data file. It is not part of the measurement configuration.

  ```
  12+ INFO MEASUREMENT DATA
  Add measurement $WORK.PERF.DATA
  From  8 Nov 1990, 14:49:27,  To  8 Nov 2002, 14:50:03
  Process                  1 Entities              71 Words
  Userdef                  1 Entities              76 Words
  -- Add Userdef $WORK.APPL.BILLING
  -- Add counter TRANSACTIONS, process $WORK.APPL.BILLING,
       accum, array 2
  -- Add counter TIMEPER, process $WORK.APPL.BILLING, busy
  -- Add counter WAITING, process $WORK.APPL.BILLING, queue
  ```

- To examine the application information, use the LIST USERDEF or LISTACTIVE USERDEF command. Each USERDEF report contains information about the executing process and the values of the user-defined counters bumped by the application.

- These reports were generated by measuring the execution of the application in Example 5-1 on page 5-3. The application bumps each type of user-defined counter. The first report is displayed with the RATE ON report attribute, the second with the RATE OFF report attribute.

```
13+ LIST USERDEF *
Process              Program $WORK.APPL.BILLING
CPU 4 Pin 45  Priority 140  Userid 255,255  Creatorid 255,255
Local System \BUYER From 8 Nov 1990, 14:49:27 For 3.5 Seconds

TRANSACTIONS            0.28                   Accum    0
TRANSACTIONS            0.28                   Accum    1
TIMEPER                87.48 %                 Busy     0
WAITING                 0.87 #   Max     1 #   Queue    0
14+ SET REPORT RATE OFF
15+ LIST USERDEF *
Process              Program $WORK.APPL.BILLING
CPU 4 Pin 45  Priority 140  Userid 255,255  Creatorid 255,255
Local System \BUYER From 8 Nov 1990, 14:49:27 For 3.5 Seconds

TRANSACTIONS               1 #                 Accum    0
TRANSACTIONS               1 #                 Accum    1
TIMEPER                 3.08 s                 Busy     0
WAITING                 3.08 s   Max     1 #   Queue    0
```

- For each user-defined counter, MEASCOM displays the name, value, type, and index value. For queue counters, MEASCOM also displays the maximum queue length.

- You can use the BY and IF clauses of the LIST and LISTALL commands on USERDEF reports. For descriptions of the BY and IF clauses, see Overriding REPORT Attributes on page 4-5.

Sample COBOL Application

To modify a COBOL program to establish a programmatic interface to the Measure subsystem and update user-defined counters:

1. Create a TAL procedure named MEAS^BUMP by compiling the sample TAL routine shown in Example 5-2.
2. Place the object in a user-named file (*measlib*).

Example 5-2. TAL Source of the MEAS^BUMP Procedure

```
?SYMBOLS,INSPECT
?NOMAP, NOLMAP, NOCODE

INT ERROR :=0;

?SOURCE $SYSTEM.SYSTEM.EXTDECS0 (MEASCOUNTERBUMPINIT,
                                           MEASCOUNTERBUMP)
INT  PROC MEAS^BUMP (name,bumptype,addvalue,index)
                                           EXTENSIBLE;
STRING .NAME;
INT    bumptype, addvalue, index;
BEGIN
INT .INAME := @Name '>>' 1;
INT OFFSET;

IF NOT $PARAM(bumptype) THEN
  bumptype := 0;

IF NOT $PARAM(addvalue) THEN
  addvalue := 0;

IF NOT $PARAM(index) THEN
  index := 0;

ERROR := MEASCOUNTERBUMPINIT (iname, offset);
If error <> 0 then return error;
RETURN   MEASCOUNTERBUMP (offset,bumptype,addvalue,index);

END;
```

The sample COBOL program in Example 5-3 on page 5-10:

1. Adds a ?SEARCH directive to identify the location of the *measlib* file
2. Adds the working storage Measure literals (WS-MEASURE-LITS) to the WORKING STORAGE SECTION of the program
3. Adds a working-storage entry (WS-*id*-NAME) to uniquely identify each user-defined counter
4. For every ADD-type counter, adds an extra working-storage entry (WS-*id*-COUNT) to hold the value to be added to the counter

The counters are ready to use after they are defined.

The MEAS-NUMS section of the program contains the code to initialize and modify the counters; for example:

```
ENTER TAL "MEAS^BUMP" USING WS-id-NAME,action,count,index
                      GIVING MEAS-ERROR
```

where:

action

is an item from WS-MEASURE-LITS.

count

is WS-*id*-COUNT (required only if *action* is MEAS-ADD).

index

is a numeric value or WORKING-STORAGE entry that contains the index value for an array of counters.

MEAS^BUMP determines the counter’s offset with the Measure internal user-defined record and updates the counter values.

The returned value is not tested because the application program does not depend on the Measure subsystem and usually is not required to report or recover from Measure subsystem errors. However, for debugging purposes, the error codes 3221 (invalid bump type) and 3222 (invalid index) might be useful. All possible return codes are listed in $SYSTEM.SYS*nn*.MEASDECS.

Example 5-3. COBOL Application Containing User-Defined Counters (page 1 of 6)

```
?SEARCH MEASLIB, $SYSTEM.SYSTEM.COBOLLIB
?INSPECT
?SYMBOLS
*
 IDENTIFICATION DIVISION.
*
 PROGRAM-ID.     MYSERVER.
 AUTHOR.
 DATE-WRITTEN.   29 OCTOBER 1987.
 DATE-COMPILED.
*
*
 ENVIRONMENT DIVISION.
*
CONFIGURATION SECTION.
 SOURCE-COMPUTER. TNSII.
 OBJECT-COMPUTER. TNSII.
*
 INPUT-OUTPUT SECTION.
*
```

Example 5-3. COBOL Application Containing User-Defined Counters (page 2 of 6)

```
  FILE-CONTROL.
     SELECT MESSAGE-FILE
         ASSIGN              TO $RECEIVE.
     SELECT REPLY-FILE
         ASSIGN              TO $RECEIVE.
     SELECT PIN-FILE
         ASSIGN              TO PINFILE
         ORGANIZATION        IS RELATIVE
         ACCESS MODE         IS RANDOM
         RELATIVE KEY        IS REL-KEY
         ALTERNATE RECORD KEY IS CUST-NO-A
         ALTERNATE RECORD KEY IS PIN-NO    WITH DUPLICATES
         FILE STATUS         IS FILE-STATUS.

   RECEIVE-CONTROL.
         TABLE OCCURS 10 TIMES
         REPLY CONTAINS REPLY-FILE RECORD.
/
*
 DATA DIVISION.
 FILE SECTION.
 FD MESSAGE-FILE
     LABEL RECORDS ARE OMITTED
     RECORD CONTAINS 1 TO 41 CHARACTERS.

     COPY CUST-NOS                 OF COPYLIB.

 FD REPLY-FILE
     LABEL RECORDS ARE OMITTED
     RECORD CONTAINS 1 TO 142 CHARACTERS.

     COPY SEC-NAR-REPLY            OF COPYLIB.

 FD PIN-FILE
     LABEL RECORDS ARE OMITTED.

     COPY PIN-RECORD               OF COPYLIB.

/
*
 WORKING-STORAGE SECTION.
*
     COPY JOB-STATE                OF COPYLIB.
     COPY WS-ERROR-LINE            OF COPYLIB.
 01  FILE-STATUS.
     03  fstat-1                     PIC 9.
     03  fstat-2                     PIC 9.
 01  REL-KEY                       PIC 99  VALUE 1.
*
*  This is the WORKING-STORAGE required to support Measure
*  user-defined counters.
*
```

Example 5-3. COBOL Application Containing User-Defined Counters (page 3 of 6)

```
01  WS-MEASURE-LITS.
     03   MEAS-INC                    PIC 9 VALUE  1.
     03   MEAS-ADD                    PIC 9 VALUE  2.
     03   MEAS-SETBUSY                PIC 9 VALUE  3.
     03   MEAS-RESETBUSY              PIC 9 VALUE  4.
     03   MEAS-INCQUEUE               PIC 9 VALUE  5.
     03   MEAS-DECQUEUE               PIC 9 VALUE  6.
     03   MEAS-ERROR                  PIC 9(4).
*
*    WS-MEASURE-LITS - are numeric literals that signify
*                      the action required from MEAS^BUMP.
*                      The numeric literals are equivalent
*                      to those given in
*                      $SYSTEM.SYSnn.MEASDECS,
*                      for use by TAL programmers.
*
*    MEAS-ERROR      - is a numeric field required to store
*                      status information returned by the
*                      calls to MEAS^BUMP and
*                      MEAS^BUMP^INIT.  The value is tested
*                      to determine whether the counter is
*                      under measurement.
 01  WS-MEASURE-COUNTERS.

     03  WS-COUNTS-NAME          PIC X(16) VALUE "COUNTS".
     03  WS-BUSY-NAME            PIC X(16) VALUE "BUSY-ST".
     03  WS-TOTAL-NAME           PIC X(16) VALUE "TOT-CNT".
     03  WS-MEAS-TOTAL-TRANS     PIC 9(4) COMP VALUE 0.

*
*   WS-id-NAME    - is an alphanumeric literal that
*                   names the Measure counter and the
*                   literal; it appears in the generated
*                   report.  The entry defines either a
*                   single counter or an array of counters.
*                   Specify the counter name in the same
*                   way for either case.

/
*
 PROCEDURE DIVISION.
 DECLARATIVES.
 FILE-RECOVERY SECTION.
 USE AFTER STANDARD EXCEPTION PROCEDURE ON MESSAGE-FILE
                                           REPLY-FILE
                                           PIN-FILE.
```

Example 5-3. COBOL Application Containing User-Defined Counters (page 4 of 6)

```
 A-00-ERROR.
*    MOVE 1 TO fstat-1.
 END DECLARATIVES.
 MAIN-PROC SECTION.
*
 B-00.
   PERFORM OPENERS.
   PERFORM READ-MESSAGE   UNTIL  JOB-STATE = "T".
   PERFORM CLOSERS.
 B-99.
*
   STOP RUN.
/
*
 OPENERS SECTION.
*
 C-00.
   OPEN INPUT  MESSAGE-FILE.
   OPEN OUTPUT REPLY-FILE.
   OPEN INPUT  PIN-FILE SHARED.

 C-99.
   EXIT.

/
*
 READ-MESSAGE SECTION.
 D-00.
   READ MESSAGE-FILE AT END MOVE "T" TO JOB-STATE.
*
* For Measure on $RECEIVE access; that is, a busy counter.
*
   PERFORM MEAS-SB.

   IF JOB-STATE  = "T" GO TO D-99.
   PERFORM VERIFY-PIN.
   ADD 1 TO WS-MEAS-TOTAL-TRANS.
*
 D-99.
   EXIT.

/
*
 VERIFY-PIN SECTION.
*
 F-00.
   MOVE CUST-NO-A    OF MESSAGE-FILE
                     TO CUST-NO-A       OF PIN-RECORD.
   MOVE 0            TO REPLY-CODE      OF SEC-NAR-REPLY.
   MOVE SPACES       TO REPLY-MSG       OF SEC-NAR-REPLY.
   READ PIN-FILE KEY IS CUST-NO-A       OF PIN-RECORD.
```

Example 5-3. COBOL Application Containing User-Defined Counters (page 5 of 6)

```
   IF fstat-1 NOT = 0
*              I.E. RECORD NOT FOUND
     MOVE "*** CUSTOMER NOT FOUND ON FILE              ***"
               TO REPLY-MSG  OF SEC-NAR-REPLY
     MOVE 0    TO REPLY-CODE OF SEC-NAR-REPLY
     WRITE SEC-NAR-REPLY
     PERFORM MEAS-INCUST
     GO TO F-99.
*              I.E. RECORD FOUND
   IF PIN-NO OF MESSAGE-FILE  = PIN-NO OF PIN-RECORD
     MOVE "*** VERIFICATION COMPLETED  ***"
               TO REPLY-MSG  OF SEC-NAR-REPLY
     MOVE 0    TO REPLY-CODE OF SEC-NAR-REPLY
     WRITE SEC-NAR-REPLY
     PERFORM MEAS-SUC
    ELSE
     MOVE "*** INVALID PIN NUMBER                      ***"
               TO REPLY-MSG  OF SEC-NAR-REPLY
     MOVE 0    TO REPLY-CODE OF SEC-NAR-REPLY
     WRITE SEC-NAR-REPLY
     PERFORM MEAS-INP.
*
*
 F-99.
   PERFORM MEAS-RB.

 F-EXIT.
   EXIT.

/
*
 MEAS-NUMS SECTION.
*
* Pinched from MEASSRC
* Measure control

 MEAS-SB.
*  setbusy  (3) for Measure control

   ENTER TAL "MEAS^BUMP" USING WS-BUSY-NAME, MEAS-SETBUSY
                        GIVING MEAS-ERROR.

 MEAS-RB.
*  resetbusy (4) for Measure control

   ENTER TAL "MEAS^BUMP" USING WS-BUSY-NAME, MEAS-RESETBUSY
                        GIVING MEAS-ERROR.

 MEAS-INCUST.
*  user-created counter for INVALID CUSTOMER ID

   ENTER TAL "MEAS^BUMP" USING WS-COUNTS-NAME, MEAS-INC,, 0
                      GIVING MEAS-ERROR.
```

Example 5-3. COBOL Application Containing User-Defined Counters (page 6 of 6)

```
 MEAS-INP.
*  user-created counter for INVALID PASSWORD

   ENTER TAL "MEAS^BUMP" USING WS-COUNTS-NAME, MEAS-INC,, 1
                         GIVING MEAS-ERROR.

 MEAS-SUC.
*  user-created counter for SUCCESSFUL PASSWORD

   ENTER TAL "MEAS^BUMP" USING WS-COUNTS-NAME, MEAS-INC,, 2
                         GIVING MEAS-ERROR.

 MEAS-CNT.
*   user-created counter for NUMBER OF TOTAL TXNS

       ENTER TAL "MEAS^BUMP" USING WS-TOTAL-NAME, MEAS-ADD,
                                   WS-MEAS-TOTAL-TRANS
                             GIVING MEAS-ERROR.

 MEAS-EXIT.
    EXIT.

/
*
 CLOSERS SECTION.
*
 G-00.
*
*  In this sample application, updating is done after each
*  session to reduce overhead.  Updating can be done after
*  each transaction if necessary, but overhead is greater.
*
   CLOSE MESSAGE-FILE REPLY-FILE PIN-FILE.

   PERFORM MEAS-CNT.
   PERFORM MEAS-RB.
*
 G-99.
   EXIT.

***********************************************************
*                                                         *
*             E N D   O F   P R O G R A M                 *
*                                                         *
***********************************************************
```

6 Creating a Custom Measurement Application

You can write application programs that control the Measure subsystem and access measurement data. Your application must pass calls to the Measure procedures, which provide a programmatic equivalent of the MEASCOM command functions.

For example, you can design custom performance tools that:

- Perform specialized analyses of Measure data
- Continuously collect selected performance information and present it for problem analysis and correction
- Dynamically balance the load on CPUs by monitoring the CPUs and creating new processes on those least loaded
- Collect capacity-planning information, analyze it, produce reports, and flag critical performance problems

This section describes how to use the callable procedures to:

- Start and stop the Measure subsystem
- Start and stop a measurement
- Read counter records from a data file and from active counters
- Check the status of the subsystem, a measurement, and a measurement configuration

Table 6-1 on page 6-2 lists the callable procedures. For more information, see the *Measure Reference Manual*.

Preparing Your Program and Defining the Configuration

Preparing your program is a four-step process:

1. Read declaration files into the source code global declarations.
2. Allocate space in your program's global data area for the Measure control block.
3. Define the entities to be measured.
4. Declare and initialize the records of the configuration table.

Table 6-1. Measure Callable Procedures (page 1 of 2)

Procedure	Function
MEAS_ADJUSTZMSRECORD_	Adjusts ZMS-style structure records to the MEASDDLS format with which an application was compiled.
MEAS_CODERANGENAME_DEMANGLE_	Supports demangling of procedure (code-range) names.
MEASCLOSE	Terminates access to a data file.
MEASCONFIGURE	Defines the entities to be measured.
MEASCONTROL	Starts and stops a measurement.
MEASCOUNTERBUMP	Bumps a user-defined counter.
MEASCOUNTERBUMPINIT	Initializes and returns the offset of a user-defined counter.
MEASGETVERSION	Returns the PVU under which a data file was created (D25, D30, and so on).
MEASLISTCONFIG	Returns system configuration information supplied by the MEASCTL process.
MEASLISTENAME	Translates Guardian file names of MIDs to corresponding external format ANSI SQL names or OSS pathnames
MEASLISTEXTNAMES	Makes Measure list structured OSS and ANSI SQL name information to the EXTNAMES file
MEASLISTGNAME	Translates OSS file pathnames to Guardian file names. Returns the MID (PATHID and CRVSN) content for entity descriptor construction. Also returns the contents of an OSS directory
MEASLISTOSSNAME	Returns structured OSS file pathname translation information to the file OSSNAMES
MEASLISTPNAME	Translates Guardian file name or OSS pathid to its OSS file pathname equivalent
MEASMONCONTROL	Starts or stops the Measure subsystem.
MEASMONSTATUS	Returns current or configured measurement data file names.
MEASINFO	Returns measurement configuration information from a data file. Unlike MEASREADCONF, MEASINFO does not require that the data file be open. Use MEASINFO when you do not require retrieval of data records.
MEASOPEN	Creates a MEASFH process to initialize or read a data file.
MEASREAD	Calls MEASREAD_DIFF_ to return counter records from a measurement data file.
MEASREADACTIVE	Reads data from active counter records, for buffers up to 32KB.

Table 6-1. Measure Callable Procedures (page 2 of 2)

Procedure	Function
MEAS_READACTIVE_	Reads data from active counter records, for buffers larger than 32KB.
MEASREADCONF	Returns measurement configuration and resource use information from a data file. Requires that the file be opened (using MEASOPEN).
MEASREAD_DIFF_	Returns counter records from a measurement data file.
MEASSTATUS	Returns resource use information for an active measurement.
MEASWRITE_DIFF_	Writes counter records from a measurement data file to a structured file.

Reading Declaration Files

Use the TAL compiler ?SOURCE command to read these declaration files into the source code global declarations:

Declaration File	Description
$SYSTEM.SYSTEM.EXTDECS0	Contains the external declarations for the Measure procedures. Each Measure procedure used in your program must be specified in the ?SOURCE command.
$SYSTEM.SYSnn.MEASDECS	Contains template structures for the Measure control block, configuration table, and entity descriptors. It also declares literal identifiers for the error message codes returned by the procedures and for constants used in the configuration table and entity descriptors.

Allocating Space for the Measure Control Block

Allocate space in the program's global data area for the Measure control block (*meascb*). You must perform this allocation unless you use only the MEASCLOSE, MEASOPEN, MEASREAD or MEASREAD_DIFF_, and MEASREADCONF procedures in a program.

You need only one Measure control block per process, regardless of the number of measurements the process makes.

MEASCB^DEF in the MEASDECS file is the template structure for the control block. This referral structure allocates space for the control block:

```
STRUCT  .MEASCB(MEASCB^DEF)
```

Before you call the first procedure that uses the control block, you must initialize each element in the block to -1:

```
MEASCB.FIRSTWORD ':=' -1 & MEASCB.FIRSTWORD
                           FOR (($LEN(MEASCB)>>1) - 1);
```

The control block is used to store data for procedure calls. Do not modify the contents of the block for subsequent Measure procedure calls.

Defining Entities

The entity descriptors define what is to be measured.

The MEASDECS file declares template structures for the entity descriptors. Table 6-2 shows which template structure to use for each entity type.

Table 6-2. MEASDECS Entity Descriptors (page 1 of 2)

Section	Descriptor Template Structure (Systems Running D-Series RVUs)	Descriptor Template Structure (Systems Running G-Series RVUs)	Type Literal	Numeric Identifier
CONTAB header	CONTAB^HDR	CONTAB^HDR	CONTAB^T	50
CPU	CPU^DESC	CPU^DESC	CPU^T	1
PROCESS	PROCESS^DESC	PROCESS^DESC PROCESS^OSS^DESC	PROCESS^T	2
PROCESSH	PROCESSH^DESC	PROCESSH^DESC CODE^SPACE^DESC PROCESSH^OSS^DESC CODE^SPACE^OSS^DESC	PROCESSH^T	3
USERDEF	USERDEF^DESC	USERDEF^DESC COUNTER^DESC USERDEF^OSS^DESC	USERDEF^T	4
FILE	FILE^OPEN^DESC	FILE^OPEN^DESC FILE^OPEN^DESC^D10 FILE^OPEN^OSS^DESC FILE^OPEN^OSS^DESC^D10	FILOP^T	5
DISCOPEN	FILE^OPEN^DESC	FILE^OPEN^DESC FILE^OPEN^OSS^DESC	DFILOP^T	6
DISC	DEVICE^DESC	DEVICE^SVNET^DESC DEVICE^SVNET^DESC^G05	DISC^T	7
DEVICE	DEVICE^DESC	DEVICE^SVNET^DESC DEVICE^SVNET^DESC^G05	IODEV^T	8
LINE	DEVICE^DESC	WAN^DESC	LINE^T	9
NETLINE	DEVICE^DESC	WAN^DESC	NETLINE^T	10
SYSTEM	SYSTEM^DESC	SYSTEM^DESC	REMSYS^T	11
CLUSTER	SYSTEM^DESC	Not applicable	CLUSTER^T	12

Table 6-2. MEASDECS Entity Descriptors (page 2 of 2)

Section	Descriptor Template Structure (Systems Running D-Series RVUs)	Descriptor Template Structure (Systems Running G-Series RVUs)	Type Literal	Numeric Identifier
TERMINAL	DEVICE^DESC	WAN^DESC	TERM^T	13
TMF	CPU^DESC	CPU^DESC	TMF^T	14
SQLPROC	SQLPROC^DESC	SQLPROC^DESC SQLPROC^OSS^DESC	SQLPROC^T	15
SQLSTMT	SQLSTMT^DESC	SQLSTMT^DESC SQLSTMT^OSS^DESC	SQLSTMT^T	16
OPDISK	OPDISK^DESC	Not applicable	OPDISK^T	17
CONTROLLER	CTRL^DESC	Not applicable	CTRL^T	18
SERVERNET	Not applicable	SVNET^DESC	SVNET^T	18
DISKFILE	DISKFILE^DESC	DISKFILE^DESC DISKFILE^OSS^DESC	DISKFILE^T	19
OSSCPU	Not applicable	CPU^DESC	OSSCPU^T	20
OSSNS	Not applicable	OSSNS^DESC	OSSNS^T	21
Maximum value			MAX^T	24
CONTAB trailer	CONTAB^TRAILER	CONTAB^TRAILER	CONTAB^TRAILER^T	51

Each entity descriptor contains these fields:

type	A number that identifies the type of entity being defined. The MEASDECS file declares some literal identifiers for these numbers. For numeric identifiers and corresponding literals, see Table 6-2.
len	The length, in bytes, of the descriptor.
cpu^number	The number of the CPU in which the entity resides.

The remaining fields in a descriptor vary according to entity type. In many fields, including the *cpu^number* field, you can use wild-card values to describe a set of entities. For example, in a CPU entity descriptor, you can use the literal ALL in the *cpu^number* field to specify all CPUs on the system. You can also use the wild-card character, an asterisk (*), in name fields. (However, when Measure reads from active counters, the entity descriptor you pass to MEASREADACTIVE must specify only one entity.)

All disk file, system, device, and process names must be in local internal name format. If you use an asterisk in a name field or subfield, you must include the appropriate leading character (\, $, or #) and pad the field with blanks. For an exact definition of each entity descriptor and its allowable field values, see the description of the MEASCONFIGURE procedure in the *Measure Reference Manual*.

Preparing the Configuration Table

The configuration table defines which entities are in a measurement. You pass the configuration table to MEASCONFIGURE before you start a measurement. After a measurement is configured, you can read its configuration table by calling MEASREADCONF or MEASINFO.

The configuration table consists of three parts:

- A header record
- The entity descriptor sections
- A trailer record

The header and trailer records have fixed lengths. The entity descriptor sections can vary in length, so the table as a whole is variable length.

Header Record

Use the template structure CONTAB^HDR to define the header record. The header record consists of these elements:

Element	**Description**
Type	The numeric identifier 50 or the string literal CONTAB^T (declared in the MEASDECS file). This field identifies this record as the header record.
Len	The length of the entire configuration table, in bytes.
Sections	An array of offsets in bytes that point to each entity type's descriptor section within the table. The first word of the array is always zero. The following offsets are ordered by entity type number and are indexed to the beginning of the configuration table. If an entity type is not being measured, its offset is zero.
Maxents	The first element in the sections array (that is, SECTIONS[0]). It identifies the header format.

Note. If you have older custom programs that do not use the current header format, HP recommends that you recompile those programs.

Entity Descriptor Sections

The body of the configuration table is an array of entity descriptors. The entity descriptors are grouped into sections according to entity type, with the sections ordered by their numeric identifiers. All CPU descriptors (type number 1) appear first, followed by all PROCESS descriptors (type number 2), then all PROCESSH descriptors (type number 3), and so on. If an entity type is not being measured, do not include a section for that type.

To use a single descriptor to define a set of entities to measure, use wild-card values in fields. After you define a set of entities, you can exclude individual entities from the set and ultimately from the measurement. To exclude an entity, add a specification for that entity to the configuration table, and set the *type* field to a negative value. You can specify a negative value for the *type* string literal or the numeric identifier.

For example, to measure all CPUs except CPU 6:

1. Add a CPU specification in which *type* is set to CPU^T and *cpu^number* is set to ALL.
2. Add a second CPU specification in which *type* is set to -CPU^T (the negative literal that corresponds to CPU^T) and *cpu^number* is set to 6.

To do the same thing using numeric type identifiers instead of string literals:

1. Add a specification in which *type* is set to 1 (the numeric identifier for CPU) and *cpu^number* is ALL (or the equivalent numeric literal, -1).
2. Add a second specification in which bit 0 of the *type* field is set to 1. This sets the exclude flag and is equivalent to setting a negative value.
3. Set bits 1 through 15 of the *type* field to 1, the numeric type identifier for CPU.
4. Set the *cpu^number* field to 6.

Trailer Record

Use the template structure CONTAB^TRAILER to define the trailer record. The trailer record consists of these elements:

- *Type*—the numeric identifier 51, or the string literal CONTAB^TRAILER^T (declared in the MEASDECS file). This field identifies this record as the trailer record.
- *Len*—the length of the trailer record, in bytes.

Use the template structure CONTAB^TRAILER to define the trailer record and to assign the literal CONTAB^TRAILER^T to the *type* field.

Starting and Stopping the Measure Subsystem

Use the MEASMONCONTROL procedure to start and stop the Measure subsystem:

- To start the subsystem, call the procedure with a nonzero *start* parameter.

  To start the subsystem, a process must be a member of the super-user group.
- To stop the subsystem, call the procedure with a zero-value *start* parameter.

  Before you stop the subsystem, to determine if any measurements are active, call the MEASMONSTATUS procedure.

Example 6-1 on page 6-8 uses MEASMONCONTROL to demonstrate all of these actions.

Example 6-1. Starting and Stopping the Subsystem

```
DEFINE  WLEN(S) = $LEN(S)>>1#;

LITERAL STOP^SUBSYS = 0,
        START^SUBSYS = -1,
        MAX^NUM^MEASUREMENTS = 64;

! Structures for the Measure control block and for the
! array of measurement names returned by MEASMONSTATUS.

STRUCT  .MEASCB(MEASCB^DEF);

STRUCT  .MEASNAMES[0:MAX^NUM^MEASUREMENTS-1];
  BEGIN
    INT FNAME[0:11];
  END;

INT MEASUREMENTS,
        ERROR;
  .
  .
  .

! Start the subsystem.  The call to MEASMONSTATUS is the
! first Measure procedure call, so initialize the control
! block first.  Then check the subsystem status by calling
! MEASMONSTATUS.  If no error is returned, the subsystem is
! running.  If error ERR^MEASMON is returned, the subsystem
! hasn't been started, so start it.

MEASCB.FIRSTWORD ':=' -1 & MEASCB.FIRSTWORD FOR
                           (WLEN(MEASCB) - 1);
IF ERROR := MEASMONSTATUS(MEASCB, MEASUREMENTS, MEASNAMES)
  THEN
    BEGIN
      IF ERROR = ERR^NOMEASMON
        THEN
          IF ERROR := MEASMONCONTROL(MEASCB,TRUE) !start it
             THEN ... ! Handle error on start attempt
        ELSE ... ! Handle error on status attempt

! Stop the subsystem.  Call MEASMONSTATUS to see if any
! measurements are active (nonzero measurements parameter
! returned).  If some are, resolve appropriately (for
! example, notify operator), and do not stop the subsystem.
! If no measurements are active (zero in returned
! measurements parameter), stop the subsystem.

IF ERROR := MEASMONSTATUS(MEASCB,MEASUREMENTS,MEASNAMES)
  THEN ... ! handle error
IF MEASUREMENTS
  THEN ... ! handle case of active measurements
  ELSE     ! shut down the subsystem
    IF ERROR := MEASMONCONTROL(MEASCB,FALSE)
      THEN ... ! handle error
```

Starting and Stopping a Measurement

To start a measurement:

1. Call MEASOPEN and pass it the data file plus a nonzero `write` parameter.
2. Call MEASCONFIGURE and pass it the data file number returned by MEASOPEN plus the location of the configuration table.
3. Call MEASCONTROL and pass it the measurement number returned by MEASCONFIGURE plus a specific start time and (optionally) a stop time.
4. If you will read data only from active counters, call MEASCLOSE to close the data file and delete the MEASFH process to release resources held by MEASFH.

Step 1: Call MEASOPEN

When you call MEASOPEN, you must pass it the data file name and a nonzero *`write`* parameter. Specifying read access when you make this call lets you later read from the data file without calling MEASOPEN again.

MEASOPEN creates a MEASFH process and sends it the data file name. If the file exists, MEASFH initializes the file. Any existing data in the file is deleted. If the data file does not exist, MEASFH creates it.

An existing file passed to the MEASOPEN procedure can be a local tape file or an unstructured local disk file with a file code of 175. (For data analysis, the data file can be local or remote, but it must be a disk file.) If MEASOPEN creates the data file, it creates an unstructured disk file, code 175, with a primary extent size of 30 pages and a secondary extent size of 30 pages. Maximum extent size for the data file is set to 256, which gives a maximum file size of 15 megabytes. If you need a file larger than 15 megabytes, create the file yourself before you start the measurement.

When you call MEASOPEN to start a measurement on an existing file, the file cannot be already open from a previous MEASOPEN call. If it is, MEASOPEN returns error ERR^FILEINUSE (code 3255). Call MEASCLOSE to close the file, then try to start the measurement again.

To specify a volume for MEASFH swap files other than the volume on which the data file resides, use the MEASOPEN option SWAPVOL. SWAPVOL is an array that contains the name of the volume for MEASFH to use for swap files when processing the data file.

Step 2: Call MEASCONFIGURE

You must call MEASCONFIGURE and pass it the data file number returned by MEASOPEN and the location of the configuration table.

MEASCONFIGURE passes the configuration table to MEASFH. MEASFH validates the format of the table and writes a copy of it to the data file.

Step 3: Call MEASCONTROL

You must call MEASCONTROL and pass it the measurement number returned by MEASCONFIGURE and a specific start time in the *starttime* parameter. You can also specify a stop time for the measurement in the first or a subsequent call to MEASCONTROL.

Specify the current time so that the measurement starts immediately. If you omit the *starttime* parameter or give it a value of -1, the measurement does not start until you call MEASCONTROL again and pass a specific start time.

You can specify a measurement interval when you call MEASCONTROL. After the measurement starts, you cannot change the interval.

To stop a measurement, specify a stop time in one of:

- The MEASCONTROL call used to start the measurement
- A subsequent MEASCONTROL call

If you do not specify a stop time when you start a measurement, the measurement runs until you call MEASCONTROL again and pass it a stop time. You must omit the *starttime* parameter in this call (or give it a value of -1). If you specify the current time in the *stoptime* parameter, the measurement stops immediately.

Step 4: Call MEASCLOSE (Optional)

To close a data file and free MEASFH resources, call MEASCLOSE. The measurement continues even though the user process closes the data file.

Example 6-2 on page 6-11:

1. Defines a measurement configuration that measures all CPUs on a system except CPU 0
2. Creates and opens a data file with a call to MEASOPEN
3. Passes the configuration table to MEASCONFIGURE
4. Starts the measurement immediately by a call to MEASCONTROL
5. Later stops the measurement with another call to MEASCONTROL

Example 6-2. Starting and Stopping a Measurement (page 1 of 2)

```
STRUCT     .CONTAB;   ! structure for configuration table
  BEGIN
    STRUCT HEADER(CONTAB^HDR);
    STRUCT CPU(CPU^DESC)[0:1];
    STRUCT TRAILER(CONTAB^TRAILER);
  END;

STRUCT     .MEASCB(MEASCB^DEF);

INT        DFNAME[0:11] := ["$SYSTEM MEASURE CPUDATA "],
           DFNUM,     ! returned by MEASOPEN
           SWAPVOL[0:3] := ["$SWAP   "],
           MEASNUM,   ! returned by MEASCONFIGURE
           ERROR;
  .
  .
  .
! Initialize the configuration table header record.

CONTAB.HEADER.TYPE := CONTAB^T;
CONTAB.HEADER.LEN := $LEN(CONTAB);
CONTAB.HEADER.SECTIONS ':=' 0 &
    CONTAB.HEADER.SECTIONS FOR MAX^T;
CONTAB.HEADER.SECTIONS[CPU^T] := $OFFSET(CONTAB.CPU);

! Initialize the first CPU descriptor to measure all CPUs.

CONTAB.CPU.TYPE := CPU^T;
CONTAB.CPU.LEN := $LEN(CPU^DESC);
CONTAB.CPU.CPU^NUMBER := ALL;

! Initialize the second CPU descriptor to remove CPU 0
! from the measurement configuration.  The first two
! assignment statements use these DEFINEs declared in the
! file MEASDECS:

DEFINE DESCTYPE = TYPE.<1:15>#;
DEFINE EXCLUDE = TYPE.<0>#;

CONTAB.CPU[1].EXCLUDE := 1;   ! set exclude flag bit
CONTAB.CPU[1].DESCTYPE := CPU^T;
CONTAB.CPU[1].LEN := $LEN(CPU^DESC);
CONTAB.CPU[1].CPU^NUMBER := 0;
```

Example 6-2. Starting and Stopping a Measurement (page 2 of 2)

```
! Initialize the trailer record.

CONTAB.TRAILER.TYPE := CONTAB^TRAILER^T;
CONTAB.TRAILER.LEN := $LEN(CONTAB.TRAILER);

! Start the measurement.  First open the data file with
! MEASOPEN:  request both read and write access and
! specify an alternate volume for MEASFH swap files.
! Next, call MEASCONFIGURE to pass CONTAB.  Finally,
! call MEASCONTROL and pass it the current time in
! starttime.

MEASCB.FIRSTWORD ':=' -1 & MEASCB.FIRSTWORD FOR
                           (WLEN(MEASCB) - 1);

IF ERROR := MEASOPEN(DFNAME,DFNUM,TRUE,TRUE,SWAPVOL)
  THEN ... ! handle error
IF ERROR := MEASCONFIGURE(MEASCB,DFNUM,MEASNUM,CONTAB)
  THEN ... ! handle error
IF ERROR := MEASCONTROL(MEASCB,MEASNUM,
                      CONVERTTIMESTAMP(JULIANTIMESTAMP,0))
  THEN ... ! handle error
  .
  .
  .

! Stop the measurement.  Call MEASCONTROL and pass it
! a -1 in starttime and the current time in stoptime.

IF ERROR := MEASCONTROL(MEASCB, MEASNUM, -1F,
                      CONVERTTIMESTAMP(JULIANTIMESTAMP,0))
  THEN... ! handle error
```

Reading Counter Records

To read counter records, use any of these procedures:

Procedure	Returns...
MEASREAD	Counter records built from counter values in the data file
MEASREAD_DIFF_	Counter records built from counter values in the data file
MEASREADACTIVE	Counter records built from active counter values

Note. The MEASREAD_DIFF_ procedure, an improvement on the MEASREAD procedure, lets you specify a time window (both FROM and TO times) in the same procedure call. You can use MEASREAD_DIFF_ with D00 or later PVUs of MEASFH. If you are using earlier PVUs of MEASFH, you must use MEASREAD. For detailed descriptions of MEASREAD and MEASREAD_DIFF_, see the *Measure Reference Manual*.

- You must pass an entity descriptor to MEASREAD, MEASREAD_DIFF_, or MEASREADACTIVE that specifies the entities whose records you want to read. Because you can pass only one entity descriptor, you can read records belonging to only one entity type in each call. If the measurement configuration includes more than one entity type, you must issue a separate MEASREAD or MEASREAD_DIFF_ call for each entity type whose records you want returned.
- You can use wild-card values in the entity descriptor passed to the MEASREAD and MEASREAD_DIFF_ procedures. For the entities included in the descriptor, MEASREAD and MEASREAD_DIFF_ return the most recent records or the records closest to the time you specify.
- The entity descriptor passed to the MEASREADACTIVE procedure must describe a single entity. Wild-card values in some fields, such as CPU, are not allowed.
- In general, you receive one record for each entity in your entity descriptor. For example, if your entity descriptor describes two entities, you receive two records, provided that records for both entities exist. In the case of USERDEF and PROCESSH entities, however, you can receive many records per entity.
- Each USERDEF counter has its own counter record; the same is true of a PROCESSH code range. Because you cannot pass counter descriptors or code-range descriptors to MEASREAD, MEASREAD_DIFF_, or MEASREADACTIVE, you cannot limit the records returned. MEASREAD and MEASREAD_DIFF_ return all counter records associated with a PROCESSH or USERDEF entity. MEASREADACTIVE returns all counter records associated with a USERDEF entity. (MEASREADACTIVE cannot read PROCESSH records.) You can select individual records from the returned records by checking the counter name and index fields of USERDEF records and the code-space and code-range fields of PROCESSH records.

- The *Measure Reference Manual* shows the counter record format for each entity type using DDL RECORD statements. To create a source file containing TAL template structures for each record, use this DDL command:

  ```
  DDL /IN MEASDDLS/TAL filename, TALBOUND 0
  ```

  where MEASDDLS is the DDL source file provided with Measure and `filename` is the TAL output file.

- The first field in a counter record is an error field. If this field is nonzero, counter space could not be allocated for the entity, and thus valid information cannot be returned in the counter value fields. The entire counter record is still returned, with valid information in the measurement and entity identification fields. For descriptions of the possible error values for each entity type, see the *Measure Reference Manual*.

Reading Counter Records From a Data File

To retrieve records from a data file:

1. If you have not obtained read access to the data file, call MEASOPEN to do so. Do not specify write access on this call. Doing so deletes the data in the file.
2. Call MEASREAD or MEASREAD_DIFF_ and pass it the entity descriptor and the size and location of the buffer to receive the returned records.
3. Check the *`bytesret`* parameter returned by MEASREAD or MEASREAD_DIFF_. If it is zero, the procedure could not find a record meeting the specifications.

Check the *`firstcall`* parameter returned by the MEASREAD or MEASREAD_DIFF_ procedure. If the specified buffer is too small to hold all the counter records, the procedure returns as many complete records as it can and returns a nonzero value in *`firstcall`*. To read the remaining records, call the read procedure again and pass it the value returned in *`firstcall`*. Continue calling the read procedure until it returns a zero in *`firstcall`*, indicating that all records have been read.

To access a remote data file, use the *`measfh`* parameter of the MEASOPEN procedure to start a MEASFH process on the remote system. The MEASFH process must run on the same system that contains either MEASCOM or the data file. If you used the optional MEASOPEN SWAPVOL parameter to specify an alternate volume for MEASFH swap files, the volume you specified must belong to the system on which MEASFH is running.

You can allocate space in an extended segment for the counter records and specify a buffer size as large as 32000 bytes (the maximum data transfer permitted by the file system) in a MEASREAD or MEASREAD_DIFF_ call. However, the read procedure might have more than 32000 bytes to return. If so, the procedure returns a nonzero value in *`firstcall`*, and you must continue to call MEASREAD or MEASREAD_DIFF_ until it returns a zero in *`firstcall`*. Each time you call the read procedure, specify a buffer size equal to 32000 or the space remaining in the extended segment, whichever is smaller.

By default, both MEASREAD and MEASREAD_DIFF_ return the most recent record for an entity. This record is written at measurement, at entity stop time, or at the last collection interval. Both procedures have optional parameters you can use to specify a target time for records:

- With the MEASREAD procedure, the *`nomtime`* parameter specifies a target time. Instead of returning the most recent record for an entity, MEASREAD returns the record whose TO-TIMESTAMP field is closest to the specified target time.

  The *`timetol`* parameter establishes a time window in which the record must reside. The window is the interval between *`nomtime`* - *`timetol`* and *`nomtime`* + *`timetol`*. If MEASREAD does not find a record whose TO-TIMESTAMP resides within the window, it does not return a record for that entity. If it finds more than one record, it returns the record whose TO-TIMESTAMP is closest to *`nomtime`*.

  If you use the *`timetol`* parameter, remember that some entities do not exist for an entire measurement interval. If you use *`timetol`* to specify a window that is less than the measurement interval, you might unintentionally exclude transient entity records. For example, if you have a measurement interval of 30 seconds, a transient entity can start and stop during the first 10 seconds of this interval. If you specify a *`timetol`* of 15 seconds, you will not receive this entity record.

- With the MEASREAD_DIFF_ procedure, the *`to^time`* and *`from^time`* parameters specify a target time window for the records. The *`timetol`* parameter specifies the tolerance on either side of the time window.

  Instead of returning the most recent record for an entity, MEASREAD_DIFF_ returns the records at *`to^time`* subtracted from the records at *`from^time`*. If MEASREAD_DIFF_ does not find a record within the window, it does not return a record for that entity. If it finds more than one record that could satisfy the *`to^time`* or *`from^time`* specification, it returns the record closest to the respective end point (either *`to^time`* or *`from^time`*). For more information, see the MEASREAD_DIFF_ procedure in the *Measure Reference Manual*.

Example 6-3 on page 6-16 shows how to use MEASREAD to open a data file and read the most recent records for all DISC entities into an extended segment; 65536 bytes are allocated in the extended segment for the returned records. (In many cases, you might need more space.)

Example 6-3. Reading Records From a Data File (page 1 of 2)

```
DEFINE    RECORD^LENGTH = $DBL($LEN(DISC^DEF))#;
LITERAL   TRUE = -1,
          FALSE = 0;
INT  .EXT RECORD^BUF[0:32767], ! space in extended segment
     .EXT READ^PTR,  ! buffer pointer
     .EXT RECORD^PTR(DISC^DEF)   !record pointer
          DFNAME[0:11] := ["$SYSTEM MEASURE DISCDATA"],
          DFNUM,
          BUFSIZE,
          BYTESRET := 0,
          ERROR,
          NUM^OF^RECORDS,
          I;
INT(32)   SPACE^AVAILABLE := 65536D,
          TOTAL^BYTES := 0D;
FIXED     FIRSTCALL := 0F; ! initialize firstcall to zero
STRUCT    DESCRIPTOR(DEVICE^DESC);
! Open the data file for read access only.

IF ERROR := MEASOPEN(DFNAME,DFNUM,FALSE,TRUE)
THEN ... ! handle error
! Initialize the descriptor to describe all disks and
! initialize buffer pointer.

DESCRIPTOR.TYPE := DISC^T;
DESCRIPTOR.LEN := $LEN(DEVICE^DESC);
DESCRIPTOR.CPU^NUMBER := ALL;
DESCRIPTOR.CHNL := ALL;
DESCRIPTOR.CTL := ALL;
DESCRIPTOR.UNIT := ALL;
DESCRIPTOR.DEVICE^NAME ':=' "$*                      ";
@READ^PTR := @RECORD^BUF;
! Read in counter records until firstcall is zero.  In a
! DO loop, first calculate the unused space in the extended
! segment, using the bytesret parameter returned by the
! last MEASREAD call.  Set the buffer size to 32000 or to
! the unused space, whichever is smaller.  Then call
! MEASREAD.  Finally, adjust the buffer pointer and
! increment the total byte count by the number returned
! in bytesret.
```

Example 6-3. Reading Records From a Data File (page 2 of 2)

```
DO
  BEGIN
    SPACE^AVAILABLE := SPACE^AVAILABLE - $DBL(BYTESRET);
    IF SPACE^AVAILABLE < RECORD^LENGTH
      THEN ... ! not enough space for one record
    IF SPACE^AVAILABLE > 32000D
      THEN BUFSIZE := 32000
      ELSE BUFSIZE := $INT(SPACE^AVAILABLE);
    IF ERROR := MEASREAD(DFNUM,DESCRIPTOR,READ^PTR,
                         BUFSIZE,BYTESRET,FIRSTCALL)
      THEN ... ! handle error
    IF NOT BYTESRET
      THEN ... ! no records returned
    TOTAL^BYTES := TOTAL^BYTES + $DBL(BYTESRET);
    @READ^PTR := @READ^PTR + $DBL(BYTESRET);
  END
UNTIL FIRSTCALL = 0F OR ERROR <> 0;
! Begin FOR loop to validate returned records.
! Assumes number of records is less than 32767.

NUM^OF^RECORDS := $INT(TOTAL^BYTES/RECORD^LENGTH);
FOR I := 0 TO NUM^OF^RECORDS - 1 DO
  BEGIN
    IF RECORD^PTR[I].ERROR THEN ... !invalid counter values
  .
  .
  .
```

After the records are received, they are validated by checking the ERROR field. An extended structure pointer based on the template structure DISC^DEF (not shown), which defines a DISC record, is used for validation.

Reading Active Counters

While an entity is being measured, you can obtain a counter record for it that contains the current values in its active counters. The active counter records are maintained by MEASCTL, not MEASFH, so you do not need to open the measurement data file to read an active counter record. You cannot read active counter records for DISCOPEN, DISKFILE, or PROCESSH entities. You must use MEASREAD or MEASREAD_DIFF_ to read these records.

To read an active counter record, call MEASREADACTIVE (for buffers up to 32 KB) or MEAS_READACTIVE_ (for buffers larger than 32 KB) and pass it an entity descriptor and the size and location of the buffer. The entity descriptor must specify a single entity. Besides the *type* and *len* fields, these fields in these descriptors must contain specific values (wild-card values are not permitted):

- For systems running D-series or G-series RVUs:
  - The *cpu^number* field in the CPU and TMF descriptors

  - The *opener^cpu*, *opener^pin*, and *file^number* fields in the FILE descriptor
  - The *cpu^number* and *pin* fields in the PROCESS and USERDEF descriptors
  - The *lh^cpu* and *system^number* fields in the SYSTEM and CLUSTER descriptors
- For systems running D-series RVUs:
  - The *cpu^number*, *channel-num*, *ctrl*, and *unit* fields in the CONTROLLER, DISC, DEVICE, LINE, NETLINE, and TERMINAL descriptors
- For systems running G-series RVUs:
  - The *cpu^num*, *servernet*, *group*, *module*, and *slot* fields in the DISC and DEVICE descriptors
  - The *cpu^number*, *trackid*, *clip*, and *line* fields in the LINE, NETLINE, and TERMINAL descriptors
  - The *cpu^number*, *group*, *module*, *SvNet-node-number*, *slot,* and *remote-CPU* fields in the SERVERNET descriptor

MEASREADACTIVE and MEAS_READACTIVE_ ignore the remaining fields in the descriptors. They can contain any valid value.

You must pass MEASREADACTIVE or MEAS_READACTIVE_ the measurement number. If you do not have this number (which is returned by MEASCONFIGURE), obtain it by calling MEASMONSTATUS and looking up the measurement data file name in the array returned by the procedure. The data file index into the array is the measurement number. (See Checking the Status of the Subsystem or a Measurement on page 6-19.)

Because you specify a single entity in the call, MEASREADACTIVE and MEAS_READACTIVE_ usually return one counter record. The exception is for a USERDEF entity. Each user-defined counter has its own record, and MEASREADACTIVE and MEAS_READACTIVE_ return all the counter records associated with a USERDEF instance if there is room. The largest *bufsize* value MEASREADACTIVE supports is 32767 bytes. To configure measurements for which returning all counters for a USERDEF instance requires more space, use MEAS_READACTIVE_.

When you start a measurement, MEASCTL initializes any new counters to zero (except CPU counters, which are initialized at system load). If the counters already exist because another measurement is using them, however, MEASCTL adopts them for the new measurement. For this reason, the counter fields in an active counter record the activity of the entity because the counters were initialized by the first measurement using them. The FROM-TIMESTAMP field in an active counter record contains the time and date when the counters were initialized.

Example 6-4 on page 6-19 shows how to call MEASREADACTIVE to read active CPU counter records one at a time into a buffer. The buffer is defined as an array of referral structures based on the template structure CPU^DEF (not shown), which defines a CPU counter record.

Example 6-4. Reading Active Counters

```
LITERAL   MAX^CPU = 15;

STRUCT    .CPU(CPU^DEF)[0:MAX^CPU]; ! counter record buffer
STRUCT    .DESCRIPTOR(CPU^DESC);    ! CPU descriptor

INT       BYTESRET,
          ERROR[0:MAX^CPU],
          I;
  .
  .
  .
! Initialize first two words of CPU descriptor.

DESCRIPTOR.TYPE := CPU^T;
DESCRIPTOR.LEN := $LEN(CPU.DESC);

! Read in CPU records.

FOR I := 0 TO MAX^CPU DO
  BEGIN
    DESCRIPTOR.CPU^NUMBER := I;
    ERROR[I] := MEASREADACTIVE(MEASCB,MEASNUM,DESCRIPTOR,
                          CPU[I],$LEN(CPU^DEF),BYTESRET);
  END;
```

Checking the Status of the Subsystem or a Measurement

Two procedure calls are used to check the status of the Measure subsystem and an active measurement.

The MEASMONSTATUS procedure call returns this information about the Measure subsystem:

- The number of measurements currently active.
- The names of the currently active or configured measurements. The names are ordered by measurement number. Therefore, you can use MEASMONSTATUS to obtain the measurement number of an active measurement.

A call to MEASCONFIGURE adds a measurement to the list of active measurements. Therefore, a measurement that is configured but not started appears in the array returned by MEASMONSTATUS. A measurement remains on the list of active measurements until a call to MEASCONTROL stops it.

The MEASSTATUS procedure call returns this information about an active measurement:

- The CPUs being measured. If any entity is being measured in a CPU and a MEASCTL process is running in that CPU, the bit corresponding to the CPU number is set in the parameter of the returned CPU.
- The measurement start, stop, and interval times. If the measurement was started without a stop or interval time, MEASSTATUS returns -1 in the *stoptime* or *interval* parameter.
- The number of entities of each type that are being measured. This information is returned in an array of MAX^T + 1 words. The first word is always zero. The remaining words are ordered by numeric identifier (1 for CPU, and so on).
- The counter space, in words, used by each entity type. This information is returned in an array of MAX^T + 1 words and is the same format as the *entities* array.

Example 6-5 calls MEASMONSTATUS and searches the array of data file names returned by MEASMONSTATUS for the file called CPU0DATA. If the file is found, MEASMONSTATUS assigns the index into the array to the variable *measnum*. It then calls MEASSTATUS, passing it the measurement number in *measnum*, and checks the *entities* array returned by MEASSTATUS to determine whether the measurement is measuring PROCESS entities.

Example 6-5. Using MEASMONSTATUS and MEASSTATUS (page 1 of 2)

```
LITERAL      MAX^NUM^MEASUREMENTS = 64;

STRUCT       .MEASNAMES[0:MAX^NUM^MEASUREMENTS-1];
  BEGIN
    INT      FNAME[0:11];
  END;

INT          MEASUREMENTS,
             ERROR,
             CPUS,
             MEASNUM;

FIXED        STARTTIME,
             STOPTIME,
             INTERVAL;

INT(32)      .ENTITIES[0:MAX^T],
             .CTRSPACE[0:MAX^T];
   .
   .
   .
! Call MEASMONSTATUS to get the measurement number.
! If you have already used the Measure control block
! (MEASCB) do not initialize it again.
```

Example 6-5. Using MEASMONSTATUS and MEASSTATUS (page 2 of 2)

```
IF ERROR := MEASMONSTATUS(MEASCB,MEASUREMENTS,MEASNAMES)
   THEN ... ! handle error
IF NOT MEASUREMENTS
   THEN ... ! no active measurements; otherwise continue
MEASNUM := 0;
WHILE (MEASNUM < MAX^NUM^MEASUREMENTS) AND
      (MEASNAMES[MEASNUM].FNAME[8] <> "CPU0DATA")
DO MEASNUM := MEASNUM + 1;
IF MEASNUM >= MAX^NUM^MEASUREMENTS
  THEN ... ! Measurement not active; otherwise continue

! Pass measnum found above to MEASSTATUS and examine the
! returned entities array.

IF ERROR :=
MEASSTATUS(MEASCB,MEASNUM,CPUS,STARTTIME,STOPTIME,
INTERVAL, ENTITIES, CTRSPACE)   THEN ... ! handle error
IF ENTITIES[PROCESS^T] > 0D
  THEN ... ! PROCESS entities are being measured
```

Reading the Measurement Configuration

You can get this information on the configuration and resource use of a measurement by calling MEASREADCONF:

- The measurement configuration table
- The measurement start, stop, and interval time
- The maximum number of entities of each type that were measured concurrently during the life of the measurement (returned for inactive measurements only)
- The maximum counter space used by each entity type during the life of the measurement (returned for inactive measurements only)

MEASREADCONF reads the configuration table from the data file. Therefore, before you call MEASREADCONF, you must call MEASOPEN to get read access to the data file.

When you call MEASREADCONF, you must specify a buffer to receive the configuration table. If the buffer you specify is too small for the entire configuration table, MEASREADCONF returns error ERR^BUFTOOSMALL (code 3204), and no information is returned. To avoid this error, you can allocate a buffer as large as 32000 bytes in an extended segment for the configuration table.

Example 6-6 on page 6-22 restarts a measurement. It uses MEASREADCONF to get the configuration table from the data file and then passes the table to MEASCONFIGURE.

Example 6-6. Restarting a Measurement (page 1 of 2)

```
LITERAL    BUFSIZE = 32000,
           TRUE = -1,
           FALSE = 0,
           MAX^NUM^MEASUREMENTS = 64;

STRUCT     .MEASNAMES[0:MAX^NUM^MEASUREMENTS-1];
  BEGIN
    INT    FNAME[0:11];
  END;

INT  .EXT  CONTAB[0:15999],  !use space in extended segment
           MEASUREMENTS,
           MEASNUM,
           FILENAME[0:11] := ["$SYSTEM MEASURE CPUDATA "],
           DFNUM,
           BYTESRET;

FIXED      STARTTIME,
           STOPTIME,
           INTERVAL;

INT(32)    .ENTITIES[0:MAX^T],
           .CTRSPACE[0:MAX^T];
  .
  .
  .
! Determine if measurement is already active.
! If you have already used the Measure control block
! (MEASCB), do not initialize it again here.

IF ERROR := MEASMONSTATUS(MEASCB,MEASUREMENTS,MEASNAMES)
  THEN ... ! handle error
MEASNUM := 0;
WHILE (MEASNUM < MAX^NUM^MEASUREMENTS) AND
      (MEASNAMES[MEASNUM].FNAME <> FILENAME FOR 12)
DO MEASNUM := MEASNUM + 1;
IF MEASNUM < MAX^NUM^MEASUREMENT
  THEN ... ! Measurement already active; don't continue

! Open the data file (read access only) and read the
! configuration table into an extended segment.  Because a
! measurement cannot be started on an open data file, close
! the data file.
```

Example 6-6. Restarting a Measurement (page 2 of 2)

```
IF ERROR := MEASOPEN(FILENAME,DFNUM,FALSE,TRUE)
  THEN ... ! handle error
IF ERROR := MEASREADCONF(DFNUM,CONTAB,BUFSIZE,BYTESRET,
                         STARTTIME,STOPTIME,INTERVAL,
                         ENTITIES,CTRSPACE)
  THEN ... ! handle error
IF ERROR := MEASCLOSE(DFNUM)
  THEN ... ! handle error

Now start measurement.  Reopen the data file, specifying
! both read and write access.  Pass the configuration table
! to MEASCONFIGURE and start the measurement immediately,
! using the measurement interval returned by MEASREADCONF.

IF ERROR := MEASOPEN(FILENAME,DFNUM,TRUE,TRUE)
  THEN ... ! handle error
IF ERROR := MEASCONFIGURE(MEASCB,DFNUM,MEASNUM,CONTAB)
  THEN ... ! handle error
IF ERROR := MEASCONTROL(MEASCB,MEASNUM, CONVERTTIMESTAMP
                       (JULIANTIMESTAMP,0), -1F, INTERVAL);
  THEN ... ! handle error
```

Modifying D-Series Applications for G-Series Systems

You might have to make changes in custom measurement applications after migrating from a D-series RVU to a G-series RVU.

D-series RVUs use channel, controller, and unit numbers to identify specific storage devices (DEVICE, DISC) and communication devices (LINE, NETLINE, TERMINAL). In G-series RVUs, instead of channel, controller, and unit numbers, you must specify group, module, and slot numbers for storage devices or track ID, CLIP, and line numbers for communication devices. Therefore, you must change any application that identifies an entity by channel, controller, or unit.

Note. OPDISK entities are also identified by channel, controller, and unit numbers. Optical disks are currently supported only in D-series RVUs.

For example, if you have a D-series application that calls the MEASREADACTIVE procedure and passes DISC identifiers to it, you must change that application. MEASREADACTIVE requires the procedure call to identify a single entity, so the call includes channel, controller, and unit identifiers. For a G-series disk, you must change the identifiers to group, module, and slot.

You do not have to change applications in which channel, controller, and unit are specified as -1 (ALL). For example, an application that calls the MEASREAD_DIFF_ procedure and passes DISC identifiers of -1 for channel, controller, and unit can be used to measure all DISC entities on a system running a G-series RVU.

The CTRL^DESC descriptor, which is used in D-series PVUs to identify CONTROLLER entities, does not exist in G-series PVUs. It is replaced by the SVNET^DESC descriptor, which identifies ServerNet addressable controllers (SACs). However, you need not modify applications that include the CTRL^DESC descriptor if the descriptor specifies all controllers or specifies only a CPU number. On systems running G-series RVUs, the application measures all SACs. To identify a specific SAC, you must use the SVNET^DESC descriptor and specify the SAC identifiers.

Applications that access Measure data files might need to be changed because of a change for G-series RVUs in ERROR field values. The ERROR field can now return a value of -1, indicating that there is data in the record but one or more fields overflowed. If an application tests for the presence of data by checking for a 0 (zero) value, you must change the application to check for either 0 or -1.

7 Balancing and Tuning a System

Tuning a system lets you take full advantage of its capabilities to provide optimum performance to all users. This requires detailed knowledge of the system with which you are working: its hardware, applications, daily use, peak use, and so on.

The first step in system tuning often is balancing the system. In a transaction processing environment, this means spreading the workload evenly across all hardware and software resources that make up the system; for example, distributing CPU usage evenly across all CPUs, distributing disk usage evenly across all disk processes, distributing line usage evenly across all lines, and so on.

A system workload consists of the processes executing on that system. A workload might not be static but instead be subject to periods of peak use over various times of the day, week, month, or year. Each process uses a different mix of system resources. Therefore, when you move a process to balance one resource (for example, CPU use), you must be aware of the effect of that process on other system resources, particularly during peak-period system usage.

This section describes basic steps for balancing and tuning a NonStop system using Measure data:

The steps in this section generally work well in a transaction processing environment. If you are not working in such an environment, consult a performance analyst before using these guidelines.

For more information on performance products and analysis techniques for NonStop systems, see Related Reading on page xii.

Balancing a System

Balancing a system starts when you configure the system hardware. You should distribute disks, terminals, and other devices evenly across all CPUs. Balancing lets you distribute the primary I/O processes for those devices (especially disk processes) across all CPUs. The eventual goal is to distribute workload, so consider which I/O processes carry the heaviest loads and try to keep them on different CPUs.

When you bring up the system, start the Measure subsystem and a system measurement. Use data from this measurement to check the load on each CPU and the balance of the workload across all resources. It can help you identify trends as your workload and your resource needs change. It can also help you identify trends in system activity, such as when the system is heavily or lightly used, the resources that are most heavily or most lightly used, and so on.

Because multiple measurements can run concurrently, the system measurement can continuously collect performance data for use in performance analyses and capacity planning without seriously affecting any other measurements that are configured.

Adding these commands to the system start-up file automatically starts the Measure subsystem and a continuously running system measurement each time you start the system. This configuration is effective for an initial evaluation of system performance.

```
MEASCOM
MEASSUBSYS
ADD CPU *
ADD PROCESS *
ADD DISC *
ADD DEVICES *
ADD LINE *
ADD NETLINE *
ADD CLUSTER *
ADD SYSTEM *
START MEASUREMENT datafile, INTERVAL 30 MIN
```

After running this measurement for 24 hours, check the size of the measurement data file. If it is growing too rapidly, modify the measurement configuration and run it again.

Tuning a System

Because tuning a system can easily create as many problems as it solves, start tuning activities only if system users are not satisfied with system performance. When you are notified of a performance problem, the measurement data file from the ongoing system measurement gives you an excellent starting point for investigating the problem.

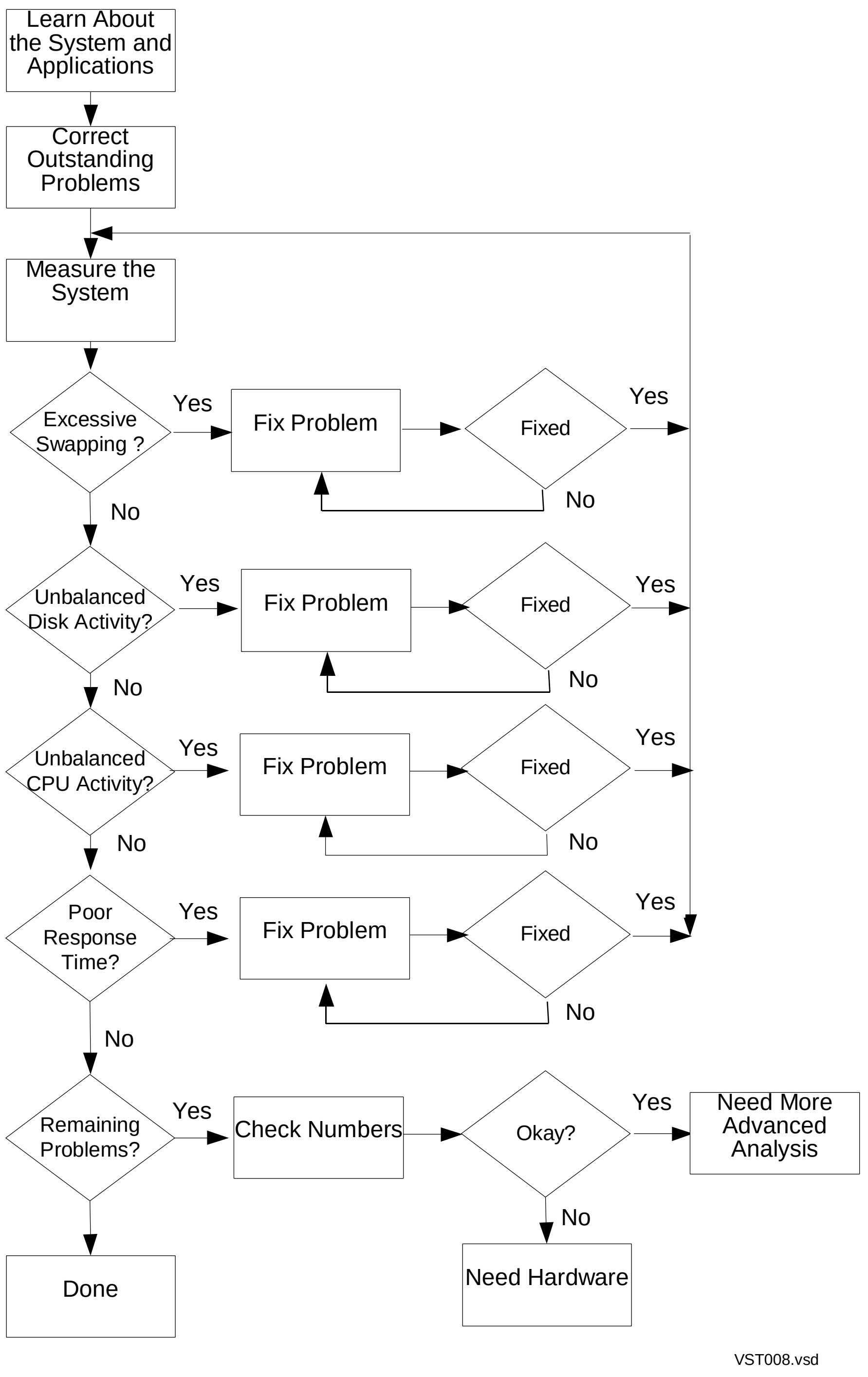
Figure 7-1 on page 7-4 shows typical steps used to tune a NonStop system. As you work on the system:

- Allow enough time
- Make one change at a time
- Check the effect before making another change

Each change you make to the system affects a number of different system resources, not just the resource on which you are working. You need to measure and understand the systemwide effect of one change before trying a second.

At some sites, analyzing performance and modifying the system are the domain of separate groups. If this is the case at your site, warn the appropriate groups that you will be making numerous requests in the coming days as you work on improving system performance.

Figure 7-1. Tuning Flow Chart



Learning About the System and Its Applications

You must know your system so you can understand its limitations. You must learn about each major application to determine how easily you can distribute its workload. Knowing your system and the applications that run on it can help you anticipate problems before they arise, identify problems more quickly and accurately when they arise, and apply the best solutions to resolve those problems.

To become familiar with the system and its major applications, draw detailed diagrams of each. Figure 7-2 on page 7-6 and Figure 7-3 on page 7-7 show sample diagrams of systems running a D-series RVU and a G-series RVU, respectively.

The system diagram should contain this information, most of which can be found in the SYSGEN files:

- CPU numbers, CPU types, and memory configuration.
- Disks on each CPU. Designate which CPU contains the primary disk process and which the backup, and indicate whether the disk is mirrored and which controllers have disks configured.
- Other I/O devices. Designate which CPU contains the primary I/O process and which the backup. For terminals, include the name of each controller and the number of terminals under its control.
- Communication lines on each CPU, including line names.
- Network connections on each CPU, including line names and whether the connection is Expand or FOX.

Figure 7-2. Sample System Diagram: D-Series RVU

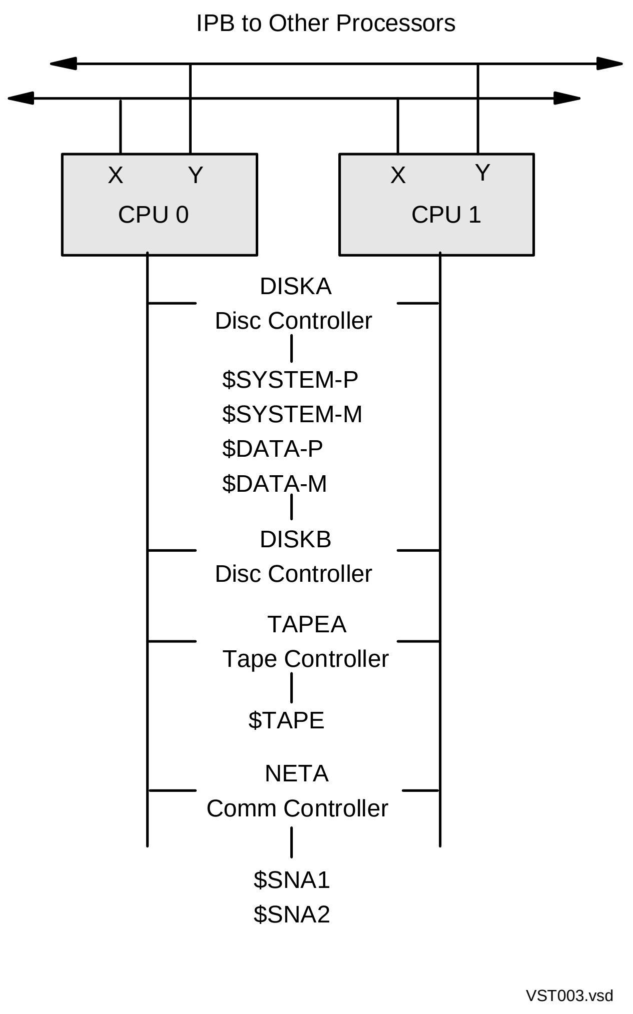
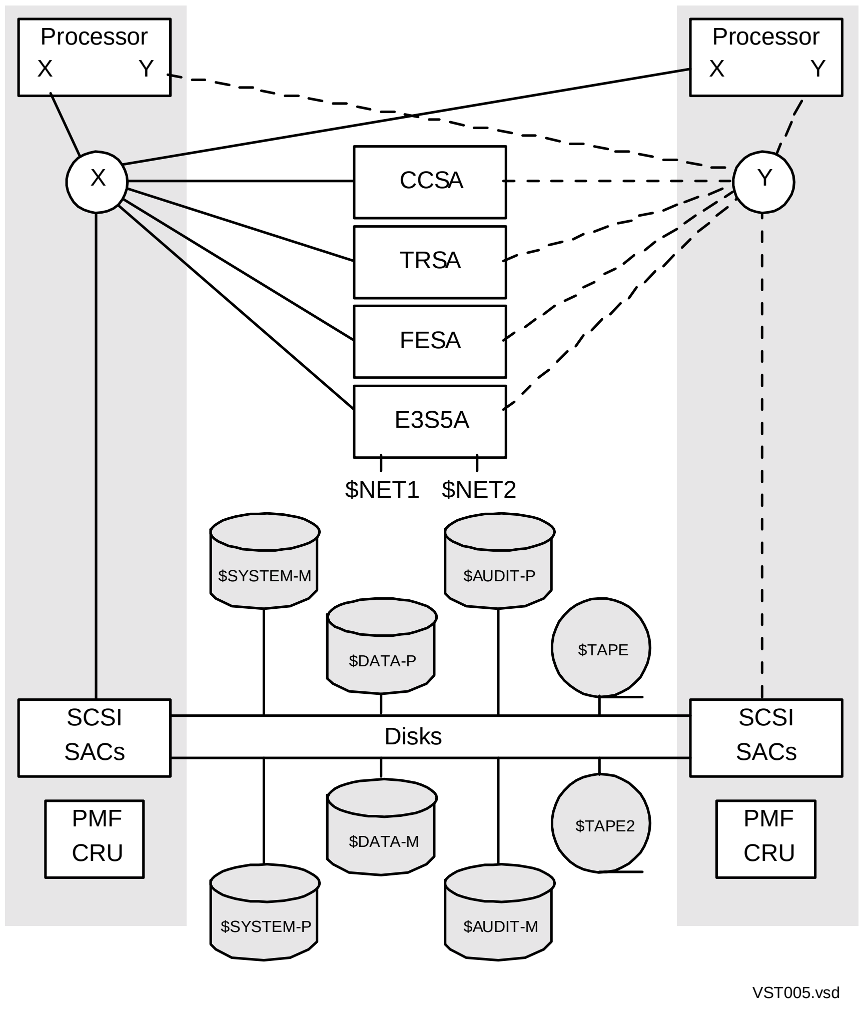


Figure 7-3. Sample System Diagram: G-Series RVU

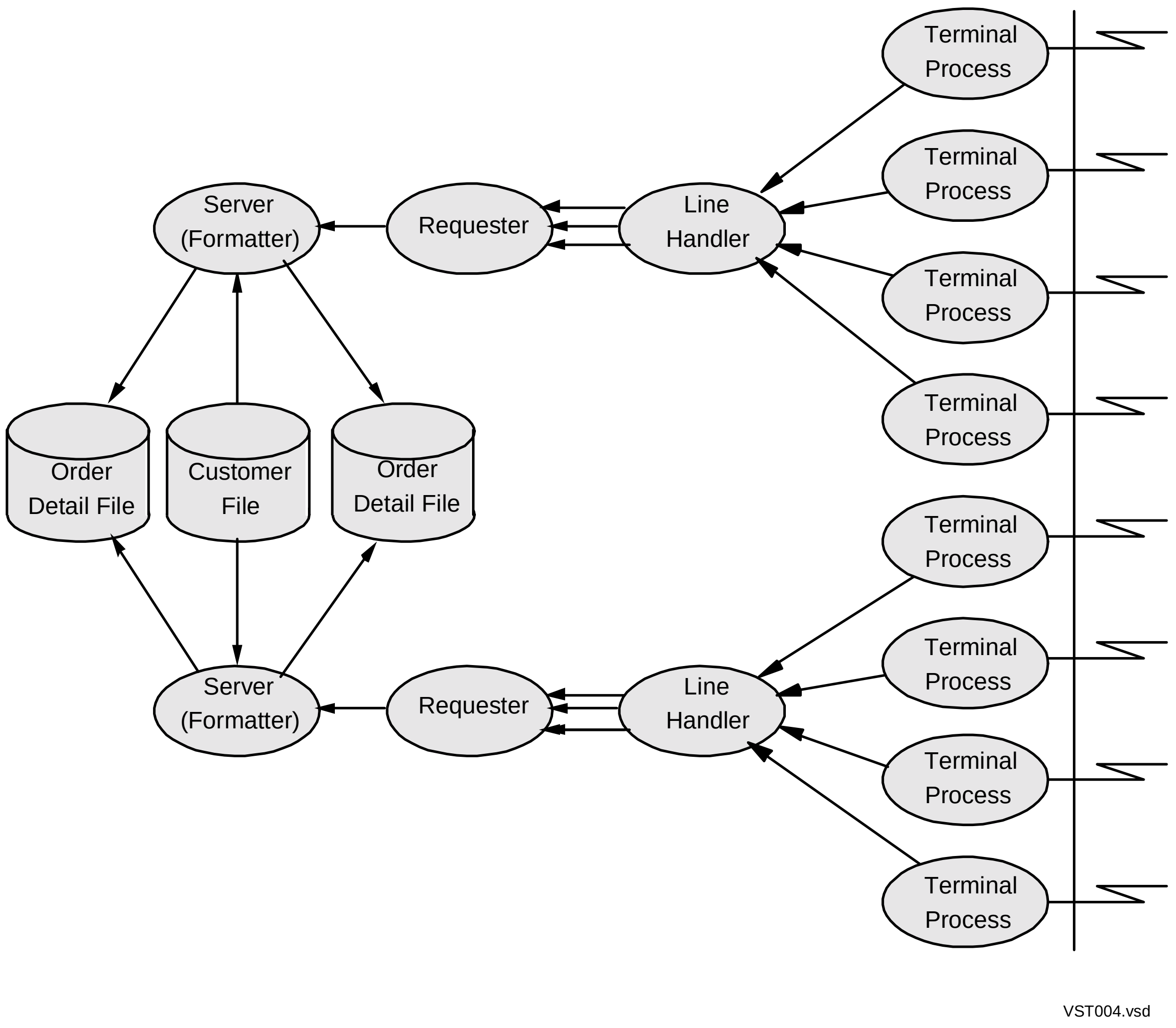


You should also develop a diagram of each major application running on the system. Each application diagram should reflect these system elements:

- Requesters
- Servers
- Data files, including how the file is structured and accessed, where the file is located, whether it can be moved, and whether it can be partitioned
- Paths between requesters, servers, and files. Include the approximate number of messages over each path.

Figure 7-4 on page 7-8 shows a sample application diagram.

Figure 7-4. Sample Application Diagram



These attributes make an application easier to tune. That is, the more of these attributes an application has, the easier it is to spread its workload across the system:

- Any application process can be moved to any CPU.
- Copies can be made of any application process, and the workload for that type of process can be spread evenly across all its running copies.
- Any file, whether a database file or program object file, can be moved to any disk drive.
- All transaction message-path lengths are kept to a minimum. Ideally, an application server should never request service from another application server. Run servers in parallel, not in series. When you arrange server tasks in a series of sequential

server processes, you lose any chance of parallel processing and create an unmanageable sequence of queues.

- Transient activity (process creations and deletions, and file opens and closes) is kept to a minimum.

If an application has few of these attributes, it is difficult to spread its workload across the system. You might have to balance the system around the application.

Pathway processes, the PATHMON-managed TCPs, and servers that make up a Pathway application can have all the listed features. A well-designed Pathway application is easy to tune.

Correcting Outstanding Problems

Check the system console and any error logs your site uses to locate problems in the system or bugs in the applications. Correct these before attempting to make other changes. Resolving a major problem in hardware or software can change the performance of the system, so eliminate these problems before proceeding.

Measuring the System

Take a measurement that gives a comprehensive picture of the system and its major applications. If possible, measure the system for a full day or more to make sure peak periods are measured. Using this data, you can find the busiest hour of the day. This peak hour is most critical for system tuning. For the longer term, it is also important to learn about peak periods during a week and at the end of the month.

To create this comprehensive picture of the system, HP recommends this measurement:

```
2+ ADD CPU *
3+ ADD DISC *
4+ ADD DEVICE *
5+ ADD LINE *
6+ ADD NETLINE *
7+ ADD CLUSTER *
8+ ADD SYSTEM *
9+ ADD PROCESS *
10+ ADD CONTROLLER *
11+ ADD DISCOPEN application data files
12+ ADD FILE application data files
13+ START MEASUREMENT file, INTERVAL 30 MIN
```

This example is valid for both D-series and G-series RVUs. Although the CONTROLLER entity type is replaced by SERVERNET in G-series PVUs, the command ADD CONTROLLER * is accepted in G-series PVUs as equivalent to ADD SERVERNET *. Either command measures all ServerNet addressable controllers (SACs). To measure specific SACs, you must use the ADD SERVERNET command and specify the SACs to measure.

Checking and Tuning Problem Areas

This subsection explains how to check and tune or balance common problem areas found in the system. When attempting to tune a system, start with the highest-level or most global external change. If the system problem is not resolved, progress to more complex or lower-level tuning tasks. This strategy is, in order of priority:

1. Balance memory consumption and minimize swapping.
2. Balance disk activity.
3. Balance CPU activity, which includes:
    - Balance other I/O process activity.
    - Balance user process activity.

Step 1: Balance Memory Consumption and Minimize Swapping

A page fault occurs when a process requires a page not currently in memory. In response, the memory manager brings the required page into memory, causing one or two swaps. A high swap rate in a CPU indicates a problem. Possible causes include:

- The CPU does not have enough memory.
- The CPU has too many primary disks primaried in the CPU.
- The disks have too much cache configured.
- Too many pages are locked into memory.
- There are too many process creations or deletions.

Applications also can cause excessive swapping by repetitively allocating and deallocating extended data segments in their program logic.

Swapping causes performance problems for two reasons:

- Excessive swapping causes unnecessary disk I/O operations.
- The CPU time required to swap pages in and out of memory could be spent performing more useful work.

A value greater than 2 swaps per second in the SWAPS counter of the CPU entity indicates a possible problem. To determine the impact of swapping on your system, use this formula to determine the maximum cost of swapping in terms of disk I/Os:

```
No. swaps in all CPUs  *  2  = No. disk I/Os for swapping
```

For example, on systems running D-series RVUs, a disk can reasonably handle 20 to 35 I/Os per second. If five CPUs each generate two faults per second, the result is 20 I/Os per second. Thus, the system is devoting much of a disk to swapping, causing an unacceptably high fault rate.

On systems running G-series RVUs, a disk can handle approximately 40 to 70 I/Os per second and can tolerate a somewhat higher swap rate. However, you should still try to reduce unnecessary swapping.

Note. Code page faults require only one disk I/O. Also, data pages do not need to be rewritten unless they have been altered. For these reasons, the swapping formula might give a high estimate of swapping cost in terms of disk I/Os.

Swapping usually indicates physical memory problems. To control swapping, you must balance the physical memory requirements of all processes across all CPUs. If none of these suggestions solve your swapping problems, you need more memory.

1. Check that each CPU has enough memory.

   Use the PEEK product to determine the maximum number of process control blocks (PCBs) used in each CPU and the number of physical memory pages in each CPU. (For more information, see the *PEEK Reference Manual*.)

2. Balance system memory requirements across all CPUs.

   A high swap rate in a single CPU indicates that the CPU does not have enough physical memory to handle its current workload. Balancing the memory requirements balances swapping.

3. Adjust cache size.

   Swapping usually indicates a lack of memory. To free memory, reduce the maximum amount of memory that can be used for disk cache. See Using Cache on page 7-15.

4. Examine the source code to see that applications are not locking pages into memory.

   Frequent allocation and deallocation of extended data segments causes swapping.

5. Check the duration of processes on CPUs to identify short-term processes.

   When a process starts, none of its required data pages are in memory. A series of page faults brings the code and data into memory. Because the SWAPS value you examine is swaps per second, this paging activity appears distributed across the duration of the process.

   For a normal process, the duration of the process is long enough to see a small SWAPS value unless the CPU needs more memory. However, for a short-term process, the initial page faults can cause a high SWAPS value even if the CPU has enough memory. To determine whether the high SWAPS value is caused by a real memory problem or short-term processes, check the duration of the processes in that CPU. (In Measure reports, the FOR field of the report header shows the duration of a process. In Enform reports, use the DELTA-TIME field.)

To balance the memory requirements across CPUs, you first need to examine the memory requirements in each CPU. To do this, list the CPUs in order of their SWAPS counter values:

```
10+ LIST CPU *, BY SWAPS
```

Do not accept a high SWAPS value as indicating memory problems without checking for short-term processes (see item 5 on page 7-11).

When you know which CPUs have a memory problem and which have enough memory, balance the memory workload by moving processes from the problem CPUs to the other CPUs. To determine the memory requirements of each process in a CPU, list the processes by their PRES-PAGES-QTIME counter values (specify *n* as the number of the CPU):

```
20+ LIST PROCESS n,*, BY PRES-PAGES-QTIME
```

Move processes so that process memory requirements are balanced across CPUs. Move one process at a time. After moving a process, check the effect of the change. If you have shifted the memory problem from one CPU to the other, move the process back and try a process with a smaller memory requirement.

Balancing Disk Activity

These components of the disk subsystem can affect performance:

- Controllers. On systems running D-series RVUs, multiple disks on each controller can cause performance problems as the disks compete for use of the controller.

  On systems running G-series RVUs, the ServerNet architecture provides faster response time and higher throughput for I/O requests. Thus, balancing controller loads for disk activity is generally not a significant performance issue.

- File system, disk process, and disk. Because these three operate serially, you can treat them as a single unit for performance analysis. These entities are used to examine this portion of the disk subsystem:

FILE	Examines logical I/O operations on a file
DISCOPEN	Examines physical I/O operations on a file in detail
DISKFILE	Provides an overview of physical I/O operations on a file
PROCESS	Examines the disk process
DISC	Examines physical disk access

Balancing disk activity includes:

1. Balance disk processes across CPUs.

   By spreading the activity of the primary disk processes evenly across the CPUs, you prevent the processes from contending with each other for CPU time. For more information, see Balancing Disk Processes on page 7-13.

2. Minimize disk I/O operations.

Too many index levels in a key-sequenced file, poorly written applications, and the wrong cache size for disk processes all generate unnecessary disk I/Os. Use FUP to check and correct the index levels in key-sequenced files. For more information, see Checking I/O Activity.

Note. On newer disk controllers, the SEEK-BUSY-TIME will be zero. Seek times are not provided. To calculate estimated time per I/O request:

```
DISC-BUSY / (READ-RATE + WRITE-RATE)
```

---

3. Distribute I/O activity evenly across all the disks. Because of application limitations and the volume of I/O activity, balancing disk activity can be a complex operation. These steps describe one approach to the problem:

   a. Balance swap activity across the disks. To examine swap activity, list the disks by their SWAPS counter values:

      ```
      16+ LIST DISC *, BY SWAPS
      ```

      To balance swapping, move object files off a heavily used disk or specify SWAP FILE in the RUN command or the call to NEWPROCESS. Having multiple copies of object files can aggravate swapping because code pages are not shared. This is especially true if you create multiple versions of heavily used object files such as library and system files. A more efficient practice is to divide copies of object files onto multiple volumes. For example, place objects A through M on one volume and objects N through Z on a second volume.

   b. Balance disk queues, as described in Balancing Disk Queues on page 7-18.

Balancing Disk Processes

Examine your system diagram. Distribute primary I/O control processes evenly across all CPUs. The I/O control processes are all high-priority processes. If you concentrate them in one CPU, they contend with each other for CPU time.

If most of the I/O processes are on a few CPUs, you probably will have to reconfigure the system and run SYSGEN on the system again. However, if the I/O processes are distributed evenly, you might be able to balance the load by moving the primary I/O process to a different CPU.

If your system diagram shows a number of heavily used disks on a single controller, the disks are likely contending for use of the controller, which can cause a performance problem.

The workload should be about the same on all PINs except the last. The last PIN should be operating at about 5 to 10 percent of capacity.

Checking I/O Activity

A single logical I/O operation on a key-sequenced disk file typically requires from one to three physical I/O operations. To determine the ratio of physical to logical I/O operations for a disk, add the READS and WRITES counters for the disk (physical

I/Os) and compare the result against the REQUESTS counter (logical I/Os). Averaging more than two physical I/Os per logical I/O indicates a possible performance problem.

The main causes of unnecessary physical I/O operations can be divided into two classes: file structure (or access method) and cache configuration.

Structuring a File

Each index level in a key-sequenced file can cause an additional physical I/O operation. Keeping a minimal number of index levels (no more than two) reduces the number of physical I/O operations required for each logical I/O operation on the file. Enter the FUP INFO command to determine the current number of index levels in a key-sequenced file. To reduce the number of index levels, reload the file with larger IBLOCK and BLOCK sizes or partition the file. The IBLOCK and BLOCK sizes must be the same.

Adding and deleting records eventually causes files to fragment. Enter the FUP INFO command to determine whether the file has a reasonable amount of slack. Too much slack can cause unnecessary disk seek time and waste disk space. Too little slack can cause block splits.

If a file is heavily used, partitioning it across multiple volumes spreads out the workload. Partitions alleviate disk hot spots and spread data over multiple CPUs and controllers. Partitioning also promotes parallel processing.

Other file attributes can cause unnecessary CPU overhead:

- Using an alternate key requires two key-sequenced searches rather than one. You can use the DISCOPEN entity to determine how frequently the application accesses the alternate-key file:

```
6+ LIST DISCOPEN alternate key file name
```

A poorly designed application can use alternate keys as frequently as primary keys. To correct the problem, you must redesign the application with these considerations:

- Updating alternate-key files during peak hours can hurt performance. If possible, turn off automatic alternate-key updates during periods of heavy usage and update the alternate keys during periods of low use. (See the ALTKEY *key* NOUPDATE option of the FUP SET command in the *File Utility Program (FUP) Reference Manual*.)
- Using data or index compression on a key-sequenced file saves disk space at the expense of CPU time. With compression off, the file system uses a binary search algorithm when accessing records. With compression on, the file system must use a sequential search. Thus saving disk space is rarely worth the CPU cost.

  Enter the FUP INFO command to determine whether a file is compressed. If so, you might want to reload the file with compression off. (For more information, see the *File Utility Program (FUP) Reference Manual*.)

Using Cache

A read or write operation to memory is much faster than a read or write operation to disk. The objective of disk cache is to keep frequently accessed information in memory, saving the time otherwise spent performing physical disk I/O.

Read and buffered write operations can use cache. Unbuffered write operations always cause a disk I/O. Because buffered writes can be collected in cache and written out as necessary, they save disk I/Os and, therefore, CPU time. For TMF-audited files, buffered write operations are the default. For nonaudited files, buffered write operations are not usually recommended because a failed disk or multiple failed CPUs can cause data loss. However, for nonaudited files, buffered writes might be efficient in cases where a job is rerun to re-create data. In most cases, Measure and Spooler data files are good candidates for buffered writes.

The recommended cache size for a disk depends on a number of factors. The size of physical memory on the CPUs containing the primary and backup disk processes and the file activity on the disk are the most important.

- Physical memory is shared by the memory manager (to allocate process code and data pages) and the disk cache for each disk process (primary or backup) on the CPU. The ideal division of memory gives the memory manager all the memory it needs (no extra) and gives the rest to the disk cache.

  If disk cache is too small, disk processes might perform unnecessary physical disk I/Os. If disk cache is too large, it can induce unnecessary swap operations and cache faults. (Swaps are also disk I/Os and therefore time consuming.)

- File activity determines cache use:

  - Random access on a key-sequenced file. When performing random access on a key-sequenced file, you must access one or more index blocks plus the data block for each I/O operation. By configuring enough cache to hold as many of the index blocks for the file as possible, you avoid the physical disk I/Os associated with the index levels and so improve performance. By adding extra cache for the data blocks, you can improve the chance of a cache hit on a data block and avoid the possibility of forcing a data block out of cache before a possible update operation.

  - Random access on an entry-sequenced or relative file. By providing enough cache to hold a substantial percentage of the file, you can increase the chances of a cache hit and thus improve performance. However, in the event of a cache miss, an entry-sequenced or relative file requires one I/O where the key-sequenced file might require more than one.

  - Sequential access on any file. Because you are accessing the information only once, the cache requirements for the file are minimal. Even for a key-sequenced file, a minimal cache configuration should keep the required index blocks in cache until they are replaced by the index blocks required for the next set of data blocks.

    - Small files. By providing enough cache to hold the entire file, you can avoid many disk I/Os and so improve performance. Providing cache does not guarantee that the file remains in cache. Unless the small file is busy, its blocks are likely to be swapped out in favor of busier files.

  Use the STATISTICS and DETAIL clauses of the FUP INFO command to examine the block sizes and index levels of the major files on each disk.

This text provides guidelines for sizing cache for DP2 disks. Adjusting disk cache can have a major effect on system performance. Do not change cache sizes unless you have a problem.

- Although you configure cache only for the primary disk process, the system automatically configures the same amount of cache for the backup disk process. Both CPUs are affected.
- When examining the file activity on a disk, consider only heavily used files. Briefly used files do not influence performance and should not influence cache sizing.

Disk cache consists of four separate caches, each containing blocks of different sizes (512 bytes, 1024 bytes, 2048 bytes, and 4096 bytes). RVUs prior to D40.00 provide a number of blocks for each cache that equals the number of PINs multiplied by 7 and rounded up to a full-page boundary. D40.00 and later RVUs provide a maximum of 56 blocks for each cache. Thus, when configuring disk cache, consider that the impact on CPU physical memory includes all four caches for each disk process, primary and backup, in the CPU.

When configuring the cache sizes for each disk, consider the file activity (as discussed at the beginning of this section) in each of the four caches for that disk. The cache a file uses is based on the file block size. Examine the block size of the major files on the disk by using the ENDING-FREE-BLOCKS counter of the DISK entity. (Alternatively, you can use the STATISTICS or DETAIL clause of the FUP INFO command.)

Examine the cache statistics collected in the C*n* counters of the DISC entity. These counters can help you size the cache:

- FAULTS. A cache fault occurs when the disk process expects to find a block in cache but discovers that the memory manager has removed it. Cache faults are a result of contention between the CPU memory manager and the CPU disk processes. A high FAULTS value (greater than 5) might be caused by several things:
    - The disk cache on the CPU might be too large.
    - Too many processes in the CPU might be locking down too many memory pages for the PFS for each process.
    - Too many system processes might be locking down memory for resident code and data pages.
    - Too little physical memory might be available to the CPU.

- ○ Backup disk processes might be consuming amounts of cache larger than their primary memory (TMF and control points).

Depending on the cause of the cache faults, you might choose to reduce the amount of cache configured for the disk, reduce the amount of cache configured in the CPU, move processes out of the CPU, or reconfigure system processes.

- BLKS (blocks allocated). If the disk has far more blocks allocated than it has in use, the amount of cache configured for this disk is too large. Determine which of the caches to reduce by considering the file activity in each cache.
- Counters reflecting hit/miss ratios. When you examine the hit/miss ratio for a particular cache, note the file activity for that cache. For example, a cache with a substantial amount of sequential I/O activity should have a good hit/miss ratio. A cache with a substantial amount of random I/O activity has a chance of producing a good hit/miss ratio. A cache with a substantial amount of unbuffered write activity cannot produce any cache write hits.

In addition, note the file activity of the other caches. For example, if all file activity for the disk is in one cache, give that cache block size the most memory. If the file activity for the disk is split between two caches, consider which files can benefit most from a larger cache.

The DISC counters that reflect cache hits and cache misses are:

DISC Counter	**The percentage of...**
CACHE READS	Cache I/O operations that were reads
CACHE READ HITS	Cache read operations that found the requested block in cache
CACHE READ MISSES	Cache read operations that did not find the requested block in cache
CACHE WRITES	Cache I/O operations that were writes
CACHE WRITE DIRTYS	Cache write operations that found the requested block in cache and dirty (saving a disk I/O)
CACHE WRITE CLEANS	Cache write operations that found the requested block in cache and clean
CACHE WRITE MISSES	Cache write operations that did not find the requested block in cache

In general, for each CPU, configure DP2 disk cache so the READ HITS and WRITE DIRTYS for each of the four caches for each disk process (primary and backup) in the CPU are as high as possible while the CACHE FAULTS for each disk process and the Measure SWAPS counter for the CPU remain low.

To get more information for cache sizing on systems running D-series RVUs, use the PUP LISTCACHE command with the STAT option, initializing the counters with the INIT option. Initialize the counters any time you begin a measurement—after modifying the cache configuration, before a peak period, and so on.

Balancing Disk Queues

Disk queues indicate the amount of time an I/O request spent waiting within the disk subsystem. Because a CPU frequently waits for an I/O operation, disk queues can cause major performance problems. By balancing the disk queues across all disks, the average I/O request wait time becomes as short as possible. Because an I/O request travels through the disk subsystem serially, calculate the total disk queue time by adding these counter values:

- The RECV-QTIME counter for the disk process (PROCESS entity), which represents how long the I/O request waited for the disk process (the external queue for the disk process). Often, a high RECV-QTIME value indicates that the disk process resides on an overly busy CPU.
- The REQUEST-QTIME counter for the disk (DISC entity), which represents how long the I/O process waited for the disk (the internal queue for the disk process). Often, a high REQUEST-QTIME counter indicates an overly busy disk.

To balance disk queues, first examine the queue length of each disk. (In Enform, you can define a new field for disk queue lengths and use that field for balancing the queues. In MEASCOM, you must do the calculations by hand.) When you know which disks have long queues and which have shorter queues, balance the queue lengths by moving files from one disk to another.

Balancing queue lengths is difficult because you must balance both disk processes and disk queues simultaneously. In addition, you might have to work around application restrictions and shortcomings; for example, a server program that uses hard-coded file names instead of ASSIGN messages or DEFINEs to determine target files to open.

In deciding which file to move and its new location, consider individual queues rather than total queue length:

- If the summation of the RECV-QTIME counters for all disk processes in the CPU you are relieving is high, the CPU is overly busy. Move frequently accessed files off the disk. Choose a file with a large number of logical I/O operations by examining the READS and WRITES counters of the FILE reports. Move the chosen file to a CPU with a relatively low RECV-QTIME counter; that is, a CPU with a relatively light CPU load.
- If the summation of the REQUEST-QTIME counters for disk processes in the CPU you are relieving is high, the disks are overly busy. Move a file that causes substantial disk I/O to another disk. Choose a file with a large number of physical I/O operations by examining the DRIVER-INPUT-CALLS and DRIVER-OUTPUT-CALLS counters of the DISCOPEN reports. Move the chosen file to a CPU with a relatively low REQUEST-QTIME counter; that is, a CPU with a relatively light physical I/O load.

Also consider the ease with which you can move the file. For example, alternate key and swap files are relatively easy to move; TMF-audited files are not.

Continue until the disk queues are balanced, but move only one file at a time. After moving a file, check the effect of the change. If you have shifted the problem from one disk to another, either move the file back and try a different file or try moving the file to a different disk.

Balancing CPU Activity

You can consider CPU activity balanced when all CPUs have approximately the same CPU-QTIME in the LIST CPU command display. Balanced CPU activity also denotes an even distribution of work requests among CPUs. CPU queue lengths should be examined and balanced as well.

Processes receive CPU time based on their priority. (Process priority assignments are discussed later. For a complete discussion of priorities and scheduling, see the *Guardian Programmer's Guide*.) Processes with the same priority compete for processing time. Processes at different priorities do not compete because higher priority processes preempt lower priority processes. For this reason, you should balance processing requirements for each priority level across all CPUs. That is, processing requirements for processes at priority 220 should be balanced, processes at priority 200 should be balanced, and so on for each priority level. In a mixed environment, batch processes should run at a much lower priority than anything else on the system.

To balance CPU activity, first examine the activity in each CPU. To do so, list the CPUs in order of their CPU-BUSY-TIME counter values. (In Enform, you can define a new CPU busy time field and use that field for CPU balancing.)

```
7+ LIST CPU *, BY CPU-BUSY-TIME
```

When you know which CPUs are overutilized and which have processing time to spare, balance the CPU activity by moving processes from the overutilized CPUs to the other CPUs.

To determine the processing requirements of all processes in a CPU, list the processes by their CPU-BUSY-TIME counter values, and balance the processing requirements for processes at a given priority level. (In MEASCOM, you must note the priority level in the report header. In Enform, you can list processes over the PRIORITY field of the process records.)

To list the processes in CPU 4 according to their CPU-BUSY-TIME counter values:

```
8+ LIST PROCESS 4,*, BY CPU-BUSY-TIME
```

In choosing a process to move, consider these factors as well as the CPU-BUSY-TIME counter value:

- The duration of the process. A short-term process does not require much processing time, regardless of its CPU-BUSY-TIME value.
- The ease with which the process can be moved. Moving a system I/O process (IOP) can be done by a PRIMARY command. Moving an application process can be simple or impossible depending on the application code.

Continue the move until the CPU activity at each priority level is balanced across all CPUs (or is as balanced as it can be given your workload). Move one process at a time. After moving a process, check the effect of the change. If you shifted the problem from one CPU to the other, move the process back and try a different process.

The priorities assigned to various types of processes can also affect performance. For example, it is common to prevent requester processes from interrupting server processes by keeping the requester processes at a lower priority.

This is one common scheme for setting priorities for processes. The priorities are kept 10 points apart for clarity and to avoid problems. The system bumps a process priority each time another process is merged in front of another process on the ready list to ensure that the process is eventually serviced by the disk process.

1. Disk processes (220)
2. Communication processes (200)
3. One command interpreter for problems (190)
4. PATHMON (180)
5. Server processes (170)
6. Requester processes (160)
7. Command interpreters (150)
8. Batch jobs (130)

   If possible, separate batch and transaction processing.

Note. There are exceptions to the priority levels of requester and server processes. Determine these case by case by experimentation.

Limiting the Number of Processes Measured

The operating system has an architectural limit of 65,534 concurrent processes per CPU. The actual number of concurrent processes possible in a CPU depends on the system's resources, such as memory.

To improve Measure performance and keep the amount of data manageable, set up the Measure configuration to collect measurements for specific processes rather than all processes.

Evaluating Response Time

Response time is the user's chief indication of performance. However, the RESPONSE-TIME counter for the TERMINAL entity does not measure the response time as seen by the user. The user sees response time as the time between pressing the RETURN or function key and the time the system displays a response on the screen. For Measure, response time is the time between the terminal process receiving the RETURN or function key and the terminal process posting a write to the terminal.

Response time should improve as the system is balanced. Do not treat response time subjectively. A system can seem faster or slower depending on how much needs to be accomplished.

Resolving Remaining Problems

If your system still has a performance problem:

- For more tuning information on systems running Pathway applications, see the *NonStop TS/MP and Pathway System Management Guide* and the *NonStop TS/MP and Pathway Management Reference Manual*.
- Check for application design problems. Poorly designed applications can cause performance problems that simple load balancing cannot solve. These symptoms indicate design problems:
    - Looping batch processes that overload disks
    - Poorly assigned application process priorities causing unnecessary queues and underutilized CPUs
    - Frequent, unnecessary application process creations and file opens and closes
    - Unnecessary or overly long record locks, as indicated by a high REQUESTS-BLOCKED counter in the DISC report
    - Long internal queues

  PROCESS reports list MQC-ALLOCATIONS and MQC-ALLOC-FAILURES. However, because an MQC request takes 10 seconds to fail, a CPU can waste time waiting for MQCs without the problem appearing in MQC-ALLOC-FAILURES.
- Reduce spooler overhead by specifying printer locations, combining small jobs into a single large job, and suppressing banner pages whenever possible.

A

Creating an Enform Report From Measure Data

You can use Enform queries to create customized performance reports from Measure data. The Enform product can access Measure data files after they are converted to structured files and a data dictionary is created. For this procedure, see Producing Structured Files of Measurement Data on page 4-31.

In Enform reports, you can display data for more than one entity on the same report to show relationships, create new report fields, select entities in a report by using any combination of field values, or display the fields in a report by ascending or descending field values.

For complete Enform documentation, see the *ENFORM Reference Manual*.

Calculating Values

You can use Enform queries to perform addition, subtraction, multiplication, and division operations on the measurement data. Because structured files contain uninterpreted data, Enform queries are commonly used to calculate the same types of averages or percentages that interpreted data would provide.

To calculate averages and percentages from the field values in structured files, be familiar with the Enform OPEN, LIST, and WHERE statements. For detailed information, see the *ENFORM Reference Manual*.

Busy Values

The uninterpreted value provided by the structured report for a busy counter is the number of microseconds busy. To calculate percent busy, use this formula:

```
(counter * 100.000) / delta-time
```

Specifying three decimal places (100.000) in this formula indicates that the resulting values are calculated to three decimal places.

This Enform session displays the PROCESS CPU-BUSY-TIME counter in microseconds in the column CPU-BUSY-TIME and as a percent busy value in the column PERCENT BUSY:

```
>OPEN PROCESS;
>LIST PROGRAM-FILE-NAME, CPU-BUSY-TIME,
>((CPU-BUSY-TIME*100.000)/DELTA-TIME) HEADING "PERCENT BUSY",
>WHERE PROGRAM-FILE-NAME CONTAINS "MCOM";

   PROGRAM-FILE-NAME      CPU-BUSY-TIME     PERCENT BUSY
------------------------  --------------    ------------

$DATD   QUOTAS  MCOM             7883841           1.676
$DATD   QUOTAS  MCOM              382553            .452
```

The PROGRAM-FILE-NAME field of the PROCESS record contains the name of the executing program file. The DELTA-TIME field contains the duration of the report window in microseconds. The WHERE clause of the LIST command selects the records to be included in the report.

Queue Lengths

The uninterpreted value provided by the structured report for a queue counter is the number of microseconds spent in the queue. To calculate average queue length, use this formula:

```
(counter * 1.000) / delta-time
```

This Enform session displays the PROCESS PRES-PAGES-QTIME counter in microseconds in the column PRES-PAGES-QTIME and as an average queue length in the column AVG LENGTH:

```
>LIST PROGRAM-FILE-NAME,PRES-PAGES-QTIME,
>((PRES-PAGES-QTIME*1.000)/DELTA-TIME) HEADING "AVG LENGTH",
>WHERE PROGRAM-FILE-NAME CONTAINS "MCOM";

   PROGRAM-FILE-NAME      PRES-PAGES-QTIME    AVG LENGTH
------------------------  ----------------    ----------

$DATD   QUOTAS  MCOM           52231001254       111.082
$DATD   QUOTAS  MCOM            5318962216        62.978
```

The PROGRAM-FILE-NAME field of the PROCESS record contains the name of the executing program file. The DELTA-TIME field contains the duration of the report window in microseconds. The WHERE clause of the LIST command selects the records to be included in the report.

Rates

The uninterpreted values provided by the structured report for incrementing and accumulating counters are the number of operations performed. To calculate operations per second, use this formula:

```
(counter * 1.000) / (delta-time/1000000)
```

You must divide by one million to get seconds because the length of the report window, DELTA-TIME, is in microseconds.

This Enform session displays the PROCESS DISPATCHES counter as the number of operations performed in the column DISPATCHES and as the number of operations performed per second in the column under DISP RATE:

```
>LIST PROGRAM-FILE-NAME,DISPATCHES,
>((DISPATCHES*1.000)/(DELTA-TIME/1000000))HEADING"DISP RATE",
>WHERE PROGRAM-FILE-NAME CONTAINS "MCOM";

   PROGRAM-FILE-NAME      DISPATCHES   DISP RATE
------------------------  ----------  ----------

$DATD   QUOTAS  MCOM            2485       5.284
$DATD   QUOTAS  MCOM             169       2.001
```

The PROGRAM-FILE-NAME field of the PROCESS record contains the name of the executing program file. The WHERE clause of the LIST command selects the records to be included in the report.

Creating User-Defined Variables

Use DECLARE statements to create user-defined variables in Enform reports. User-defined variables are most often used in LIST statements to calculate new report items from existing record fields.

To use variables wherever you would use field names, you must create a new record definition and assign the variable values to the fields of the new record. (See Creating User-Defined Records on page A-4.)

This DECLARE statement defines the variable PROC-TIME. By default, a variable's initial value is 0. You can specify a different initial value by using the Enform SET command.

```
>DECLARE proc-time INTERNAL F19.3 AS F9.3
>HEADING "Process/Busy Time/(seconds)";
```

The INTERNAL clause defines the PROC-TIME internal data structure as F19.3, where 19 is the total number of character positions, including the decimal point, and 3 is the number of digits to the right of the decimal point. (An 18-digit decimal number with three digits to the right of the decimal place is used for internal Enform calculations.)

The AS clause defines the PROC-TIME display format as F9.3. That is, PROC-TIME values are displayed in the format *"nnnnn.nnn"* (9 character places including the decimal, with 3 digits to the right of the decimal). The display format is important for two reasons:

- If the value calculated for a field is larger than the declared display format, the calculated value is treated as an overflow, and the report field is filled with asterisks (*).
- The size of each field display format contributes to the overall length of each report line. If the sum of the display format sizes and the spaces between the items is larger than the number of characters available on a report line, each report line is wrapped onto two lines of the listing or terminal display, which results in confusing reports.

The HEADING clause defines the text used to label columns containing the PROC-TIME variable. The slash specifies a line break, so HEADING "Process/Busy Time/(seconds)" produces the heading:

```
  Process
 Busy Time
 (seconds)
 ---------
     .
     .
     .
```

This example uses the PROC-TIME variable in a LIST statement to calculate a new report item from existing record fields:

```
>OPEN process;
>LIST BY cpu-num HEADING "CPU/Number",
>proc-time :=
>((SUM(cpu-busy-time OVER cpu-num))/1000000);
```

You can also use variables in the WHERE clause of a LIST command. However, only the initial value of the variable is used when evaluating the WHERE clause.

You cannot use a variable in an aggregate. That is, a variable cannot be a BY item, an OVER item, or the parameter of an Enform function.

Creating User-Defined Records

Like user-defined variables, user-defined records let you calculate new report fields from existing fields. After declaring a field in a user-defined record, you can use that field in an Enform aggregate, such as a LIST BY command.

Although a number of restrictions are placed on user-defined variables, no such restrictions are placed on user-defined records. You can create new record fields by performing calculations on existing record fields.

To define and assign field values to a new record:

1. Decide on the field names and definitions. For example:

   ```
   proc-time :=
       ((SUM(cpu-busy-time OVER cpu-num))/1000000)
   sys-time  :=
       ((SUM(cpu-busy-time OVER cpu-num WHERE
       program-file-name CONTAINS "OSIMAGE"))/1000000)
   disc-time :=
       ((SUM(cpu-busy-time OVER cpu-num WHERE
       priority >= 220))/1000000)
   user-time := (proc-time - sys-time)
   ```

   CPU-BUSY-TIME and CPU-NUM are fields in the existing PROCESS record.

2. Create a file containing the DDL record definition for the fields of the new record. This example defines a record, BUSYTIME, containing the fields listed under item 1:

   ```
   RECORD busytime.  FILE IS busytime.

   02 cpu-num   TYPE binary 16 unsigned.
   02 proc-time TYPE binary 64.
   02 sys-time  TYPE binary 64.
   02 disc-time TYPE binary 64.
   02 user-time TYPE binary 64.
   ```

   The RECORD statement defines the record structure. The FILE IS clause identifies the file to contain the records. The 02 is the level number of the field within the record. The level number indicates the field's relationship to other fields within the record. The field name follows the level number. The TYPE clause defines the data type and size of the field.

   For the full syntax and a complete explanation of the RECORD statement, see the *Data Definition Language (DDL) Reference Manual*.

3. Add the new DDL record definition to the DDL dictionary that contains the Measure DDL records. (For an explanation of how to create the DDL dictionary, see Producing Structured Files of Measurement Data on page 4-31.)

   Assuming that the DDL record definition appears in a source file named BUSYDDLS, this command adds the record definition to the DDL dictionary on the current subvolume:

   ```
   DDL /IN BUSYDDLS/ DICT
   ```

4. Use an Enform FIND statement to assign values to the fields of the new record:

   ```
   >OPEN process, busytime;
   >FIND busytime
   >(
   >by process.cpu-num,
   >proc-time := ((SUM(cpu-busy-time OVER
   >              process.cpu-num))/1000000),

   >sys-time  := ((SUM(cpu-busy-time OVER
   >              process.cpu-num WHERE program-file-name
   ```

```
>              CONTAINS "OSIMAGE"))/1000000),
>disc-time := ((SUM(cpu-busy-time
>              OVER process.cpu-num WHERE
>              priority >= 220))/1000000),
>user-time := ( ((SUM(cpu-busy-time OVER
>                process.cpu-num))/1000000)
>             -((SUM(cpu-busy-time OVER
>                process.cpu-num WHERE program-file-name
>                CONTAINS "OSIMAGE"))/1000000) )
>);
```

The FIND statement begins by naming the new record to receive the values listed in the statement. The remainder of the FIND statement is much like the LIST statement. However, instead of displaying the specified values, the FIND statement writes the specified values to the new record.

Each field value specified in the FIND statement must include the name of the field in the new record that is to receive the field value. In the example, the first field value specified is PROCESS.CPU-NUM. The new record has a CPU-NUM field, so the value of PROCESS.CPU-NUM is assigned to BUSYTIME.CPU-NUM. Use of the BY clause causes one BUSYTIME record to be created for each CPU.

The remaining field values in the FIND statement do not have matching field names in the new record, so the FIND statement explicitly names the field in the new record that is to receive the value specified in the FIND statement. For example, the PROC-TIME field receives the sum of all PROCESS.CPU-BUSY-TIME fields in the CPU.

Because the values being assigned to the new record are grouped according to the CPU-NUM field (the OVER item), the CPU-NUM field must be included in the new record.

5. You can now use the new record as you would any other record. This example displays two of the new fields. Although the user-defined variable PROC-TIME cannot be used as a BY item, the field PROC-TIME can.

```
>LIST BY PROC-TIME,SYS-TIME;

   PROC-TIME          SYS-TIME
----------------  ----------------
         218.423           134.772
           4.192             4.036
```

For more examples of DDL RECORD statements and FIND queries for CPU and PROCESS data files, see Appendix B, Examples of RECORD Statements and FIND Queries.

Creating an Enform Report

To create an Enform report from Measure data:

1. Use DDL to add the required RECORD statements to the DDL dictionary.
2. If you use key-sequenced or alternate-key files, create a FUP schema file. Alternate-key files improve Enform performance when you sort the measurement data by field.
   a. Use the ?FUP command of DDL to create the schema file.
   b. Use FUP to create the key-sequenced and alternate-key files described by the schema file.
3. Use the Measure REPORT attributes to generate structured files. See Producing Structured Files of Measurement Data on page 4-31.
4. Use the Enform product to create the unstructured files containing the data to be used in the report.
5. Use FUP to load the key-sequenced and alternate-key files with data from the unstructured files.
6. Use the Enform product to create the report.

If you generally use the same reports each time you analyze new measurement data, you can create command files that perform the preceding steps. Examples of command files and their related files follow.

- The command file NEWSUBVL performs Steps 1 and 2, setting up a subvolume so the second command file, STARTENF, can create the report. Whenever you copy the Enform queries and associated files to a new subvolume, run NEWSUBVL.
- The command file STARTENF performs Steps 3 through 6, creating the report and writing it to the spooler under the name $S.#ENFOUT.

NEWSUBVL—Establishing the Subvolume

To execute NEWSUBVL (create all files in a single subvolume):

1. Type in NEWSUBVL:

```
NEWSUBVL:

 DDL /IN $SYSTEM.SYSnn.MEASDDLS/ DICT !
 DDL /IN DDLXCHNG/ DICT
 FUP /IN FUPPURGE/
 EDIT/OUT $s.#jnk/fupxchng;CB/SET TYPE/SET EXT
    (20,20),TYPE/A;E
 FUP /IN FUPSCHNG/
 ENFORM /IN COMPILE/
```

The first line of NEWSUBVL creates a new DDL dictionary using the RECORD statements in $SYSTEM.SYS*nn*.MEASDDLS. Examine NEWSUBVL to ensure that the specified SYS*nn* subvolume is correct for your system. If not, edit NEWSUBVL and correct it.

Many of the commands in NEWSUBVL assume that the current subvolume contains certain required files. The remaining steps in this list create these files.

2. Type in DDLXCHNG, as shown in Example A-1 on page A-9. DDLXCHNG contains the RECORD statements that define the unstructured (UNBASE and UNPROC) and key-sequenced (BASEREP and PROCREP) files that will contain the information to be used in the report.

   The ?FUP command at the beginning of DDLXCHNG directs DDL to create a FUP schema file, FUPXCHNG. The schema file created by DDL contains the FUP commands that define a file to match each RECORD statement. Later, you use FUPXCHNG as input to FUP to create the key-sequenced and alternate key files.

   You can use FUPXCHNG as written by DDL. However, in this example, NEWSUBVL edits FUPXCHNG to create all the files with primary and secondary extents of 20 pages.

   Example A-2 on page A-12 shows the FUPXCHNG file created by DDL after it has been edited by NEWSUBVL.

3. Type in FUPPURGE, as shown here. FUPPURGE purges the files defined by the schema file. You cannot use FUP and the schema file to create the unstructured and key-sequenced files if files with the specified names already exist.

   ```
   FUPPURGE:

    ALLOW 100 ERRORS
    ALLOW 100 WARNINGS
    PURGE (UNPROC, UNBASE) !
    PURGE (PROCREP, BASEREP) !
    PURGE (PROCREP0) !
   ```

4. Type in COMPILE. COMPILE contains the Enform commands that compile the queries used to create the report. By compiling the queries in NEWSUBVL, you save time when you run them to generate the report.

   ```
   COMPILE:

      ?COMPILE findbcpu TO rfbase
      ?COMPILE findproc TO rfproc
      ?COMPILE qdproc TO rqdproc
   ```

5. Enter the queries FINDBCPU, FINDPROC, and QDPROC. (See Example A-3 on page A-12 through Example A-5 on page A-14.) FINDBCPU contains the FIND statement that defines the fields in the UNCPU record. FINDPROC contains the FIND statement that defines the fields in the UNPROC record. QDPROC contains the Enform statements that generate the final report.

6. Execute NEWSUBVL:

```
44> OBEY NEWSUBVL
```

Example A-1. DDLXCHNG File for Enform Reporting (page 1 of 3)

```
?FUP FUPXCHNG!
?SECTION unproc

RECORD unproc.       FILE IS unproc.

02 loadid             TYPE character 8.
02 from-timestamp     TYPE binary 64.
02 to-timestamp       TYPE binary 64.
02 delta-time         TYPE binary 64,3.
02 system-name        TYPE character 8.
02 cpu-num            TYPE binary 16 unsigned.
02 pin                TYPE binary 16 unsigned.
02 process-name       TYPE character 8.
02 program-file-name  TYPE character 24.
02 priority           TYPE binary 16 unsigned.
02 cpu-busy           TYPE binary 64,3.
02 atime-busy         TYPE binary 64,3.
02 atime-ready        TYPE binary 64,3.
02 ready-busy         TYPE binary 64,3.
02 atime-memq         TYPE binary 64,3.
02 disp-rate          TYPE binary 64,3.
02 fault-rate         TYPE binary 64,3.
02 pres-pages         TYPE binary 64,3.
02 pres-pagem         TYPE binary 16 unsigned.
02 ext-segs           TYPE binary 64,3.
02 ext-segsm          TYPE binary 16 unsigned.
02 vsem-rate          TYPE binary 64,3.
02 msg-rate-nr        TYPE binary 64,3.
02 msg-rate           TYPE binary 64,3.
02 sbyte-rate         TYPE binary 64,3.
02 rbyte-rate         TYPE binary 64,3.
02 recv-qlen          TYPE binary 64,3.
02 recv-qlenm         TYPE binary 16 unsigned.
02 recv-rate-nr       TYPE binary 64,3.
02 recv-rate          TYPE binary 64,3.
02 vbyte-rate         TYPE binary 64,3.
02 ybyte-rate         TYPE binary 64,3.
02 link-busy          TYPE binary 64,3.
02 link-rate          TYPE binary 64,3.
02 link-fail          TYPE binary 64,3.
02 links-inuse        TYPE binary 64,3.
02 links-inusem       TYPE binary 16 unsigned.
02 chkpt-rate         TYPE binary 64,3.
end
```

Example A-1. DDLXCHNG File for Enform Reporting (page 2 of 3)

```
?SECTION procrep
RECORD procrep.      FILE IS procrep ENTRY-SEQUENCED.

02 loadid             PIC a(8)                Heading "LOAD/ID".
02 from-timestamp     TYPE binary 64.
02 to-timestamp       TYPE binary 64.
02 delta-time         TYPE binary 64,3.
02 system-name        TYPE character 8        Heading "SYS/NAME".

02 pid.
   04 cpu-num         TYPE binary 16 unsigned Display "m<Z9>"
                                              Heading "CPU".
   04 pin             TYPE binary 16 unsigned Display "m<ZZ9>"
                                              Heading "PIN".
02 process-name       TYPE character 8        Heading "PROCESS/NAME".
02 program-file-name                          Heading "PRGRM/FILE/NAME".
   04 volume          TYPE character 8.
   04 sub-vol         TYPE character 8.
   04 name            TYPE character 8.
02 priority           TYPE binary 16 unsigned Display "m<ZZ9>"
                                              Heading "PRI".
02 cpu-busy           TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "CPU/BUSY".
02 atime-busy         TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "ATIME/BUSY/in ms".
02 atime-ready        TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "ATIME/READY/in ms".
02 ready-busy         TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "READY/VS./BUSY".
02 atime-memq         TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "ATIME/MEMQ/in ms".
02 disp-rate          TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "DISP/RATE".
02 fault-rate         TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "FAULT/RATE".
02 pres-pages         TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "PRES/PAGES".
02 pres-pagem         TYPE binary 16 unsigned Display "m<ZZZ9>"
                                              Heading "PRES/PAGES/MAX".
02 ext-segs           TYPE binary 64,3        Display "m<ZZ9.99>"
                                              Heading "EXT/SEGS".
02 ext-segsm          TYPE binary 16 unsigned Display "m<ZZ9>"
                                              Heading "EXT/SEGS/MAX".
02 vsem-rate          TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "VSEM/RATE".
02 msg-rate-nr        TYPE binary 64,3        Display "m<ZZZZZZ9>"
                                              Heading "MESSAGE/CNTR".
02 msg-rate           TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "MESSAGE/RATE".
02 sbyte-rate         TYPE binary 64,3        Display "m<ZZZZZ9.9>"
                                              Heading "SEND/BYTE/RATE".
02 rbyte-rate         TYPE binary 64,3        Display "m<ZZZZZ9.9>"
                                              Heading "REPLY/TO
                                                 SEND/BYTE/RATE".
02 recv-qlen          TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "$RCV/RECV/QLEN".
02 recv-qlenm         TYPE binary 16 unsigned Display "m<ZZ9>"
                                              Heading "$RCV/RECV/QLEN/MAX".
02 recv-rate-nr       TYPE binary 64,3        Display "m<ZZZZZZ9>"
                                              Heading "$RCV/RECV/CNTR".
```

Example A-1. DDLXCHNG File for Enform Reporting (page 3 of 3)

```
02 recv-rate          TYPE binary 64,3        Display "m<ZZ9.999>"
                                              Heading "$RCV/RECV/RATE".
02 vbyte-rate         TYPE binary 64,3        Display "m<ZZZZZ9.9>"
                                              Heading "$RCV/RECV/BYTE/RATE".
02 ybyte-rate         TYPE binary 64,3        Display "m<ZZZZZ9.9>"
                                              Heading "$RCV/REPLY/BYTE/RATE".
02 link-busy          TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "LINK/BUSY".
02 link-rate          TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "LINK/RATE".
02 link-fail          TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "LINK/FAIL".
02 links-inuse        TYPE binary 64,3        Display "m<ZZ9.99>"
                                              Heading "AVG/LINKS/IN USE".
02 links-inusem       TYPE binary 16 unsigned Display "m<ZZ9>"
                                              Heading "LINKS/IN USE/MAX".
02 chkpt-rate         TYPE binary 64,3        Display "m<Z9.999>"
                                              Heading "CHKPT/RATE".

key "pc" is cpu-num.
key "pn" is process-name.
key "pf" is program-file-name.name.
                      SEQUENCE IS PID.
end

?SECTION baserep
RECORD baserep.      FILE IS baserep ENTRY-SEQUENCED.

02 b-loadid           TYPE character 8.
02 b-system-name      TYPE character 8.
02 b-os-version
   03 b-letter        TYPE character 1.
   03 b-number        TYPE binary 8.
02 b-from-time        TYPE binary 64.
02 b-to-time          TYPE binary 64.
02 b-delta-time       TYPE binary 64,3.
02 b-cpu-type         TYPE binary 16 unsigned.

                       SEQUENCE IS b-LOADID.

end
RECORD unbase.       FILE IS unbase KEY-SEQUENCED.

02 loadid             TYPE character 8.
02 system-name        TYPE character 8.
02 os-version
   03 letter          TYPE character 1.
   03 number          TYPE binary 8.
02 from-timestamp     TYPE binary 64.
02 to-timestamp       TYPE binary 64.
02 delta-time         TYPE binary 64,3.
02 cpu-type           TYPE binary 16 unsigned.

key is loadid.

end
```

Example A-2. FUPXCHNG File for Enform Reporting

```
< SCHEMA PRODUCED DATE - TIME : 12/10/90  09:47:46
< SECTION UNPROC
< Record UNPROC created on 12/10/90 at 09:47
RESET
   set ext (20,20), type U
   SET REC 278
CREATE UNPROC
< SECTION PROCREP
< Record PROCREP created on 12/10/90 at 09:47
RESET
   SET ALTKEY ( "pc", KEYOFF 40, KEYLEN 2, FILE 0 )
   SET ALTKEY ( "pn", KEYOFF 44, KEYLEN 8, FILE 0 )
   SET ALTKEY ( "pf", KEYOFF 68, KEYLEN 8, FILE 0 )
   SET NO ALTCREATE
   SET ALTFILE ( 0, PROCREP0 )
   set ext (20,20), type E
   SET REC 278
   SET BLOCK 512
CREATE PROCREP
   RESET
   set ext (20,20), type K
   SET KEYLEN 14
   SET REC 14
   SET BLOCK 512
   SET IBLOCK 512
CREATE PROCREP0

< SECTION BASEREP
< Record BASEREP created on 12/10/90 at 09:48
RESET
   set ext (20,20), type E
   SET REC 44
   SET BLOCK 512
CREATE BASEREP
< SECTION UNBASE
< Record UNBASE created on 12/10/90 at 09:48
RESET
   set ext (20,20), type K
   SET KEYOFF 0
   SET KEYLEN 8
   SET REC 44
   SET BLOCK 512
   SET IBLOCK 512
CREATE UNBASE
```

Example A-3. FINDBCPU Query for Enform Reporting

```
?DICTIONARY

OPEN cpu, unbase;

FIND UNIQUE unbase
(
BY cpu.loadid
  system-name    := cpu.system-name
  os-version     := cpu.os-version
  from-timestamp := MIN(cpu.from-timestamp OVER cpu.loadid)
  to-timestamp   := MAX(cpu.to-timestamp OVER cpu.loadid)
  delta-time     := MAX(cpu.delta-time OVER cpu.loadid)
  cpu-type       := cpu.cpu-type
 );
```

Example A-4. FINDPROC Query for Enform Reporting

```
?DICTIONARY

OPEN process;
OPEN cpu;
OPEN unproc;

LINK process TO cpu VIA cpu-num;

FIND UNIQUE unproc (
process.loadid,
process.from-timestamp,
process.to-timestamp,
process.delta-time,
process.system-name,
process.cpu-num,
process.pin,
process.process-name,
process.program-file-name,
process.priority,
cpu-busy     := ((process.cpu-busy-time * 100) / process.delta-time),
atime-busy   := (if process.dispatches > 0 then
                ((process.cpu-busy-time / 1000) / process.dispatches)
                else 0),
atime-ready  := (if process.dispatches > 0 then
                ((ready-time / 1000) / process.dispatches) else 0),
atime-memq   := (if process.dispatches > 0 then
                ((process.mem-qtime / 1000) / process.dispatches) else 0),
disp-rate    := (process.dispatches / (process.delta-time / 1000000)),
fault-rate   := (page-faults / (process.delta-time / 1000000)),
pres-pages   := (pres-pages-qtime / process.delta-time),
pres-pagem   := pres-pages-max,
ext-segs     := (ext-segs-qtime / process.delta-time),
ext-segsm    := ext-segs-max,
vsem-rate    := (vsems / (process.delta-time / 1000000)),
msg-rate-nr  := messages-sent,
msg-rate     := (messages-sent / (process.delta-time / 1000000)),
sbyte-rate   := (sent-bytes / (process.delta-time / 1000000)),
rbyte-rate   := (returned-bytes / (process.delta-time / 1000000)),
recv-qlen    := (recv-qtime / process.delta-time),
recv-qlenm   := recv-qlen-max,
recv-rate-nr := messages-received,
recv-rate    := (messages-received / (process.delta-time / 1000000)),
vbyte-rate   := (received-bytes / (process.delta-time / 1000000)),
ybyte-rate   := (reply-bytes / (process.delta-time / 1000000)),
link-rate    := (mqc-allocations / (process.delta-time / 1000000)),
link-fail    := (mqc-alloc-failures / (process.delta-time / 1000000)),

links-inuse  := (mqcs-inuse-qtime / process.delta-time),
links-inusem := max-mqcs-inuse,
chkpt-rate   := (checkpoints / (process.delta-time / 1000000)),
)
WHERE ((process.cpu-busy-time > 0 OR process.priority = 220)
                          AND process.delta-time > 0
                          AND cpu.mqcs > 0)
!**     (pri = 220 is to get backup disk processes with no activity)
```

Example A-5. QDPROC Query for Enform Reporting (page 1 of 3)

```
!=============== Measure analysis by process-group
?DICTIONARY
SET @DATE-FORMAT TO "M2-D2-Y2"
SET @WIDTH TO 132;
SET @LINES TO 54;

DECLARE TEST-DATE      AS DATE *
        SERVICE-TIME  INTERNAL F19.3
        CPU-TIME      INTERNAL F19.3
        AR-TIME       INTERNAL F19.3
        AB-TIME       INTERNAL F19.3
        DLTA-FACT     INTERNAL F19.3

OPEN PROCREP;
OPEN BASEREP;

LINK BASEREP.B-LOADID TO PROCREP.LOADID;

LIST

     BY LOADID                      NOPRINT

     BY CPU-NUM                     HEADING "CPU/NUM"

     BY PROGRAM-FILE-NAME.NAME      HEADING "PROGRAM/NAME"

     BY PROCESS-NAME

     DLTA-FACT := (PROCREP.DELTA-TIME / BASEREP.B-DELTA-TIME)
                         NOPRINT

     ((CPU-BUSY * 10.) * DLTA-FACT)
                                    AS I4
                                    HEADING "CPU/MSEC/SEC"
                                    SUBTOTAL OVER CPU-NUM
                                    TOTAL
     ((ATIME-MEMQ * DISP-RATE) * DLTA-FACT)
                                    AS I4
                                    HEADING "FALT/MSEC/SEC"
                                    SUBTOTAL OVER CPU-NUM
                                    TOTAL

     (PRES-PAGES + EXT-SEGS)
                                    AS I4
                                    HEADING "MEM/PAGES"

     RECV-QLEN
                                    AS F4.1
                                    HEADING "RECV/QLEN"

     (ATIME-READY / ATIME-BUSY)
                                    AS F4.1
                                    HEADING "READY/BUSY/RATIO"
```

Example A-5. QDPROC Query for Enform Reporting (page 2 of 3)

```
PRIORITY
                             AS I3
                             HEADING "PRI"

SERVICE-TIME :=
(IF RECV-RATE <= 0 THEN 100000.0
                   ELSE
   ((ATIME-READY * DISP-RATE) / RECV-RATE))
               NOPRINT

(IF SERVICE-TIME > 9999.8 THEN "         "
                   ELSE
             ((CPU-BUSY / RECV-RATE) * 10.))
                           AS F6.1
                           HEADING "CPU/COST/REQ"

(IF SERVICE-TIME > 9999.8 THEN "         "
                   ELSE
                 SERVICE-TIME)
                           AS F6.1
                           HEADING "SERV/TIME/REQ"

(IF SERVICE-TIME > 9999.9 THEN "         "
                   ELSE
                (SERVICE-TIME * (RECV-QLEN + 1)))
                           AS F6.1
                           HEADING "RESP/TIME/REQ"

(MSG-RATE)
                           AS F4.1
                           HEADING "MSG/RATE"
(MSG-RATE * DLTA-FACT)
                           AS F5.1
                           HEADING "MSG/RATE/WINDW"
                           SUBTOTAL OVER CPU-NUM
                           TOTAL

(RECV-RATE)
                           AS F4.1
                           HEADING "RECV/RATE"

(RECV-RATE * DLTA-FACT)
                           AS F5.1
                           HEADING "RECV/RATE/WINDW"
                           SUBTOTAL OVER CPU-NUM
                           TOTAL

(DELTA-TIME / 1000000)
                           AS I5
                           HEADING "MEAS/SEC"

COUNT (PROCREP.PID OVER PROGRAM-FILE-NAME.NAME)
                           AS I4
                           SUBTOTAL OVER CPU-NUM
                           HEADING "PROC/CNT"
```

Example A-5. QDPROC Query for Enform Reporting (page 3 of 3)

```
     AVG ((PROCREP.DELTA-TIME / 1000000) OVER PROGRAM-FILE-NAME.NAME)
                                AS I4
                                HEADING "AVG/PROC/LIFE"

            WHERE PROCREP.CPU-BUSY > 1
            OR NOT PROGRAM-FILE-NAME.NAME = "OSIMAGE"
            OR NOT PROGRAM-FILE-NAME.SUB-VOL BEGINS WITH "SYS"
                      AND
                ATIME-BUSY > 0
                      OR
                PROCREP.PROCESS-NAME = "$VIRTUAL"

AT START
    PRINT  SKIP 1 "TEST NO: " B-LOADID
           SKIP 1 "MEAS DATE - "
                     TIMESTAMP-DATE AS DATE "M2-D2-Y2"
           SKIP 1 "FROM TIME - "
                     TIMESTAMP-TIME AS TIME "H2:M2:S2"
           SKIP 1 "TO TIME - "
                     TIMESTAMP-TIME AS TIME "H2:M2:S2"
           SKIP 1 "SYSTEM NAME " B-SYSTEM-NAME
           SKIP 1 "OS VERSION  " B-OS-VERSION.B-LETTER
                                 B-OS-VERSION.B-NUMBER
           SKIP 2
TITLE
    "\NEW" SKIP 1 "MEASUREMENT ANALYSIS : PROCESS DETAIL"
    SKIP 1
    "DATE RUN : " @DATE AS DATE *,
    SPACE 5
    "TEST NO: "    LOADID SPACE 5
```

STARTENF—Creating the Report

The Enform report created by the examples in this appendix uses information from the Measure CPU and PROCESS reports. Therefore, before you can create the report, you must take a measurement and write CPU and PROCESS data to structured report files. This example assumes the measurement data file NOV04 contains CPU and PROCESS data to use for the Enform report:

```
45> MEASCOM
MEASURE Performance Monitor - T9086D30 - (31OCT94) - \BUYER
Copyright Tandem Computers Incorporated 1986-1994
1+ ADD NOV04
2+ SET REPORT FORMAT STRUCTURED
3+ LIST CPU *
4+ LIST PROCESS *
5+ EXIT
```

After you copy the structured CPU and PROCESS files to the subvolume containing the Enform information, execute STARTENF. Create all files in the subvolume containing NEWSUBVL and its associated files.

1. Type in (or copy) STARTENF. Many commands in STARTENF assume that the current subvolume contains certain required files. The remaining steps in this list create these files.

   ```
   STARTENF:

    FUP /IN FUPPURG/
    ENFORM /in ENFFBASE/
    FUP /IN LOADBASE/
    TACL /IN PRNTPROC, OUT $S.#ENFOUT/
   ```

2. Type in (or copy) FUPPURG. FUPPURG deletes the unstructured files (UNPROC and UNBASE, which you re-create using the Enform product) and the contents of the key-sequenced and alternate-key files (PROCREP and BASEREP, which you reload using FUP).

   ```
   FUPPURG:

    ALLOW 100 ERRORS
    ALLOW 100 WARNINGS
    PURGEDATA (PROCREP, BASEREP)
    PURGE (UNPROC, UNBASE)!
   ```

3. Type in (or copy) ENFFBASE. ENFFBASE executes the FINDBASE and FINDPROC queries that create the unstructured files, UNBASE and UNPROC. (The SET @STATS ON command generates Enform performance statistics.)

   ```
   ENFFBASE:

    SET @STATS ON
    ?EXECUTE RFBASE
    ?EXECUTE RFPROC
   ```

4. Type in (or copy) LOADBASE. LOADBASE loads the key-sequenced and alternate-key files, BASEREP, PROCREP, and PROCREP0, using the data from the unstructured files, UNBASE and UNPROC.

   ```
   LOADBASE:

    ALLOW 100 ERRORS
    ALLOW 100 WARNINGS
    COPY UNBASE, BASEREP, RECIN 44, COUNT 1
    LOAD UNPROC, PROCREP, RECIN 278
    LOADALTFILE 0, PROCREP
   ```

   You copy only one record into BASEREP because the information describes the measurement rather than the measurement data. The record lengths for the RECIN option of the FUP COPY and LOAD commands were copied from the FUPXCHNG file.

5. Type in (or copy) PRNTPROC and ENFQPROC. PRNTPROC invokes the Enform product, specifying ENFQPROC as the input file. ENFQPROC contains the Enform

command to execute the QDPROC query, which creates the report. (The SET @STATS ON command generates Enform performance statistics.)

```
PRNTPROC:

 CLEAR ALL PARAM
 ENFORM/IN ENFQPROC/

ENFQPROC:

 SET @STATS ON
 ?EXECUTE QDPROC
```

6. Execute STARTENF:

```
46> OBEY STARTENF
```

B Examples of RECORD Statements and FIND Queries

This appendix contains examples of Data Definition Language (DDL) RECORD statements and Enform FIND queries for the CPU and PROCESS entities. The FIND queries and RECORD statements can be used to create new fields for uninterpreted counter values. This appendix also contains examples of DDL RECORD statements that can be used to create alternate-key files for these new fields.

Example B-1. DDL RECORD Statement for Unstructured CPU File (page 1 of 2)

```
RECORD uncpu.           FILE IS uncpu.

02 loadid            TYPE character 8.
02  load-id          Redefines loadid.
    03  prefix-id    TYPE character 5.
    03  interval-id  TYPE character 3.
02 system-name       TYPE character 8.
02 os-version.
   03 letter         TYPE character 1.
   03 number         TYPE binary 8.
02 cpu-num           TYPE binary 16 unsigned.
02 from-timestamp    TYPE binary 64.
02 to-timestamp      TYPE binary 64.
02 delta-time        TYPE binary 64.
02 cpu-type          TYPE binary 16 unsigned.
02 cpu-busy          TYPE binary 64,3.
02 cpu-qlen          TYPE binary 64,3.
02 cpu-qlenm         TYPE binary 16 unsigned.
02 mem-qlen          TYPE binary 64,3.
02 mem-qlenm         TYPE binary 16 unsigned.
02 disp-rate         TYPE binary 64,3.
02 swap-rate         TYPE binary 64,3.
02 send-busy         TYPE binary 64,3.
02 disc-rate         TYPE binary 64,3.
02 chit-rate         TYPE binary 64,3.
02 tran-rate         TYPE binary 64,3.
02 resp-time         TYPE binary 64,3.
02 accel-busy        TYPE binary 64,3.
02 tns-busy          TYPE binary 64,3.
02 comptrap-rate     TYPE binary 64,3.
02 tnsr-busy         TYPE binary 64,3.
02 page-sizeb        TYPE binary 16 unsigned.
02 meminit-lock      TYPE binary 32 unsigned.
02 pagereq-rate      TYPE binary 64,3.
02 pagescan-rate     TYPE binary 64,3.
02 Start-freemem     TYPE binary 32 unsigned.
02 End-freemem       TYPE binary 32 unsigned.
02 Start-UCME        TYPE binary 32 unsigned.
02 End-UCME          TYPE binary 32 unsigned.
02 Start-UDS         TYPE binary 32 unsigned.
02 End-UDS           TYPE binary 32 unsigned.
02 Start-SDS         TYPE binary 32 unsigned.
```

Example B-1. DDL RECORD Statement for Unstructured CPU File (page 2 of 2)

```
02 End-SDS          TYPE binary 32 unsigned.
02 Start UCL        TYPE binary 32 unsigned.
02 End-UCL          TYPE binary 32 unsigned.
02 Start-SCL        TYPE binary 32 unsigned.
02 End-SCL          TYPE binary 32 unsigned.
end
```

Example B-2. Enform FIND Query for Unstructured CPU File

```
?DICTIONARY

OPEN cpu, uncpu;

FIND UNIQUE uncpu
(
cpu.loadid,
ascd cpu.cpu-num,
cpu.from-timestamp,
cpu.to-timestamp,
cpu.delta-time,
cpu-busy      := ((cpu-busy-time * 100) / cpu.delta-time),
cpu-qlen      := (cpu-qtime / cpu.delta-time),
cpu-qlenm     := cpu.cpu-qlen-max,
mem-qlen      := (mem-qtime / cpu.delta-time),
mem-qlenm     := cpu.mem-qlen-max
disp-rate     := (dispatches / (cpu.delta-time / 1000000)),
swap-rate     := (swaps / (cpu.delta-time / 1000000)),
send-busy     := ((send-busy-time * 100) / cpu.delta-time),
disc-rate     := (disc-ios / (cpu.delta-time / 1000000)),
chit-rate     := (cache-hits / (cpu.delta-time / 1000000)),
tran-rate     := (transactions / (cpu.delta-time / 1000000)),
resp-time     := (IF transactions > 0
                  THEN ((response-time / 1000000) / transactions) ELSE 0),
accel-busy    := ((cpu.accel-busy-time * 100) / cpu.delta-time),
tns-busy      := (( cpu.tns-busy-time * 100) / cpu.delta-time),
comptrap-rate := (cpu.comp-traps / (cpu.delta-time / 1000000)),
tnsr-busy     := ((tnsr-busy-time * 100) / cpu.delta-time),
page-sizeb    := page-size-bytes,
meminit-lock  := mem-initial-lock,
pagereq-rate  := (page-requests / (cpu.delta-time / 1000000)),
pagescan-rate := (page-scans / (cpu.delta-time / 1000000)),
Start-freemem := Starting-free-mem,
End-freemem   := Ending-free-mem,
Start-UCME    := Starting-UCME,
End-UCME      := Ending-UCME,
Start-UDS     := Starting-UDS,
End-UDS       := Ending-UDS,
Start-SDS     := Starting-SDS,
End-SDS       := Ending-SDS,
Start-UCL     := Starting-UCL,
End-UCL       := Ending-UCL,
Start-SCL     := Starting-SCL,
End-SCL       := Ending-SCL,
)
    WHERE cpu.delta-time > 0;
```

Example B-3. DDL RECORD Statement for Entry-Sequenced CPU File (page 1 of 2)

```
?FUP CPUFUP

RECORD cpurep.        FILE IS cpurep ENTRY-SEQUENCED.
02 loadid           TYPE character 8         HEADING "LOAD/ID".
02  load-id          Redefines loadid.
    03  prefix-id   TYPE character 5.
    03  interval-id TYPE character 3.
02 system-name      TYPE character 8         HEADING "SYSTEM/NAME".
02 os-version.
   03 letter        TYPE character 1.
   03 number        TYPE binary 8            DISPLAY "m<Z99>".
02 cpu-num          TYPE binary 16 unsigned  DISPLAY "m<Z9>"
                                             HEADING "CPU/NUM".
02 from-timestamp   TYPE binary 64.
02 to-timestamp     TYPE binary 64.
02 delta-time       TYPE binary 64           HEADING "DELTA/TIME".
02 cpu-type         TYPE binary 16 unsigned  DISPLAY "m<ZZ9.999>"
                                             HEADING "CPU/TYPE".
02 cpu-busy         TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                             HEADING "CPU/BUSY".
02 cpu-qlen         TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "CPU/QLEN".
02 cpu-qlenm        TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                             HEADING "CPU/QLENM".
02 mem-qlen         TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "MEM/QLEN".
02 mem-qlenm        TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                             HEADING "MEM/QLENM".
02 disp-rate        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "DISP/RATE".
02 swap-rate        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "SWAP/RATE".
02 send-busy        TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "SEND/BUSY".
02 disc-rate        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "DISC/RATE".
02 chit-rate        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "CHIT/RATE".
02 tran-rate        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "TRAN/RATE".
02 resp-time        TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "RESP/TIME".
02 accel-busy       TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "ACCEL/BUSY".
02 tns-busy         TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "TNS/BUSY".
02 comptrap-rate    TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "COMPTRAP/RATE".
02 tnsr-busy        TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                             HEADING "TNSR/BUSY".
02 page-sizeb       TYPE binary 16 unsigned  DISPLAY "m<ZZZ9>"
                                             HEADING "PAGE/SIZE/BYTES".
02 meminit-lock     TYPE binary 32 unsigned  DISPLAY "m<ZZZ9>"
                                             HEADING "MEM/INITIAL/LOCK".
02 pagereq-rate     TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "PAGE/REQ/RATE".
02 pagescan-rate    TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                             HEADING "PAGE/SCAN/RATE".
```

Example B-3. DDL RECORD Statement for Entry-Sequenced CPU File (page 2 of 2)

```
02 Start-freemem  TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/FREE/MEM".
02 End-freemem    TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/FREE/MEM".
02 Start-UCME     TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/UCME".
02 End-UCME       TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/UCME".
02 Start-UDS      TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/UDS".
02 End-UDS        TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/UDS".
02 Start SDS      TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/SDS".
02 End-SDS        TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/UDS".
02 Start UCL      TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/UCL".
02 End-UCL        TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/UCL".
02 Start-SCL      TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "STARTING/SCL".
02 End-SCL        TYPE binary 32 unsigned   DISPLAY "m<ZZZ9>"
                                            HEADING "ENDING/SCL".

key "cc" is cpu-num.
               SEQUENCE IS CPU-NUM.

end
```

These examples are based on the PROCESS DDL record for systems running D-series RVUs. The PROCESS record for systems running G-series RVUs has slightly different fields.

Example B-4. DDL RECORD Statement for Unstructured PROCESS File (page 1 of 2)

```
RECORD unproc.       FILE IS unproc.

02 loadid               TYPE character 8.
02  load-id          Redefines loadid.
    03  prefix-id    TYPE character 5.
    03  interval-id  TYPE character 3.
02 from-timestamp       TYPE binary 64.
02 to-timestamp         TYPE binary 64.
02 delta-time           TYPE binary 64.
02 system-name          TYPE character 8.
02 cpu-num              TYPE binary 16 unsigned.
02 pin                  TYPE binary 16 unsigned.
02 process-name         TYPE character 8.
02 program-file-name    TYPE character 24.
02 priority             TYPE binary 16 unsigned.
02 cpu-busy             TYPE binary 64,3.
02 atime-busy           TYPE binary 64,3.
02 atime-ready          TYPE binary 64,3.
02 atime-memq           TYPE binary 64,3.
02 disp-rate            TYPE binary 64,3.
```

Example B-4. DDL RECORD Statement for Unstructured PROCESS File (page 2 of 2)

```
02 fault-rate          TYPE binary 64,3.
02 pres-pages          TYPE binary 64,3.
02 pres-pagem          TYPE binary 16 unsigned.
02 ext-segs            TYPE binary 64,3.
02 ext-segsm           TYPE binary 16 unsigned.
02 vsem-rate           TYPE binary 64,3.
02 msg-rate            TYPE binary 64,3.
02 sbyte-rate          TYPE binary 64,3.
02 rbyte-rate          TYPE binary 64,3.
02 recv-qlen           TYPE binary 64,3.
02 recv-qlenm          TYPE binary 16 unsigned.
02 recv-rate           TYPE binary 64,3.
02 vbyte-rate          TYPE binary 64,3.
02 ybyte-rate          TYPE binary 64,3.
02 link-rate           TYPE binary 64,3.
02 link-fail           TYPE binary 64,3.
02 links-inuse         TYPE binary 64,3.
02 links-inusem        TYPE binary 16 unsigned.
02 chkpt-rate          TYPE binary 64,3.
02 accel-busy          TYPE binary 64,3.
02 tns-busy            TYPE binary 64,3.
02 comptrap-rate       TYPE binary 64,3.
02 tnsr-busy           TYPE binary 64,3.
02 page-sizeb          TYPE binary 16 unsigned.
02 allocseq-rate       TYPE binary 64,3.
02 UCLqlen             TYPE binary 64,3.
02 UCLmax              TYPE binary 16 unsigned.
02 fileopen-rate       TYPE binary 64,3.
02 infocall-rate       TYPE binary 64,3.
end
```

Example B-5. Enform FIND Query for Unstructured PROCESS File (page 1 of 2)

```
?DICTIONARY

OPEN process;
OPEN cpu;
OPEN unproc;

LINK process TO cpu VIA cpu-num;

FIND UNIQUE unproc
(
process.loadid,
process.from-timestamp,
process.to-timestamp,
process.delta-time,
process.system-name,
process.cpu-num,
process.pin,
process.process-name,
process.program-file-name,
process.priority,
cpu-busy        := ((process.cpu-busy-time * 100) / process.delta-time),
atime-busy      := ((process.cpu-busy-time / 1000) / process.dispatches),
atime-ready     := ((ready-time / 1000) / process.dispatches),
atime-memq      := ((process.mem-qtime / 1000) / process.dispatches),
```

Example B-5. Enform FIND Query for Unstructured PROCESS File (page 2 of 2)

```
disp-rate      := (process.dispatches / (process.delta-time / 1000000)),
fault-rate     := (page-faults / (process.delta-time / 1000000)),
pres-pages     := (pres-pages-qtime / process.delta-time),
pres-pagem     := pres-pages-max,
ext-segs       := (ext-segs-qtime / process.delta-time),
ext-segsm      := ext-segs-max,
vsem-rate      := (vsems / (process.delta-time / 1000000)),
msg-rate       := (messages-sent / (process.delta-time / 1000000)),
sbyte-rate     := (sent-bytes / (process.delta-time / 1000000)),
rbyte-rate     := (returned-bytes / (process.delta-time / 1000000)),
recv-qlen      := (recv-qtime / process.delta-time),
recv-qlenm     := recv-qlen-max,
recv-rate      := (messages-received / (process.delta-time / 1000000)),
vbyte-rate     := (received-bytes / (process.delta-time / 1000000)),
ybyte-rate     := (reply-bytes / (process.delta-time / 1000000)),
link-rate      := (mqc-allocations / (process.delta-time / 1000000)),
link-fail      := (mqc-alloc-failures / (process.delta-time / 1000000)),
links-inuse    := (mqcs-inuse-qtime / process.delta-time),
links-inusem   := max-mqcs-inuse,
chkpt-rate     := (checkpoints / (process.delta-time / 1000000)),
accel-busy     := ((process.accel-busy-time * 100) / process.delta-time),
tns-busy       := ((process.tns-busy-time * 100) / process.delta-time),
comptrap-rate  := (process.comp-traps / (process.delta-time / 1000000)),
tnsr-busy      := ((process.tnsr-busy-time * 100) / process.delta-time),
page-sizeb     := process-size-bytes,
allocseg-rate  := (alloc-seg-calls / (process.delta-time / 1000000)),
UCLqlen        := (UCL-qtime / process.delta-time),
UCLmax         := UCL-max,
fileopen-rate  := (file-open-calls / (process.delta-time / 1000000)),
infocall-rate  := (info-calls / (process.delta-time / 1000000)),
)
    WHERE process.cpu-busy-time > 0
      AND process.dispatches    > 0;
```

Example B-6. DDL RECORD Statement for Entry-Sequenced PROCESS File (page 1 of 2)

```
?FUP PROCFUP

RECORD procrep.       FILE IS procrep ENTRY-SEQUENCED.

02 loadid              PIC a(8)               HEADING "LOAD/ID".
02  load-id         Redefines loadid.
    03  prefix-id   TYPE character 5.
    03  interval-id TYPE character 3.
02 from-timestamp      TYPE binary 64.
02 to-timestamp        TYPE binary 64.
02 delta-time          TYPE binary 64.
02 system-name         TYPE character 8         HEADING "SYS/NAME".
02 pid.
   03 cpu-num          TYPE binary 16 unsigned  DISPLAY "m<Z9>"
                                                HEADING "CPU".
   03 pin              TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                                HEADING "PIN".
02 process-name        TYPE character 8         HEADING "PROCESS/NAME".
02 program-file-name                            HEADING "PRGRM/FILE/NAME".
   03 volume           TYPE character 8.
   03 sub-vol          TYPE character 8.
   03 name             TYPE character 8.
02 priority            TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                                HEADING "PRI".
02 cpu-busy            TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "CPU/BUSY".
02 atime-busy          TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "ATIME/BUSY/in ms".
02 atime-ready         TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "ATIME/READY/in ms".
02 atime-memq          TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "ATIME/MEMQ/in ms".
02 disp-rate           TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "DISP/RATE".
02 fault-rate          TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "FAULT/RATE".
02 pres-pages          TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "PRES/PAGES".
02 pres-pagem          TYPE binary 16 unsigned  DISPLAY "m<ZZZ9>"
                                                HEADING "PRES/PAGES/MAX".
02 ext-segs            TYPE binary 64,3         DISPLAY "m<ZZ9.99>"
                                                HEADING "EXT/SEGS".
02 ext-segsm           TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                                HEADING "EXT/SEGS/MAX".
02 vsem-rate           TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "VSEM/RATE".
02 msg-rate            TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "SEND/MSG/RATE".
02 sbyte-rate          TYPE binary 64,3         DISPLAY "m<ZZZZZ9.9>"
                                                HEADING "SEND/BYTE/RATE".
02 rbyte-rate          TYPE binary 64,3         DISPLAY "m<ZZZZZ9.9>"
                                                HEADING
                                                  "REPLY/TO SEND/BYTE/RATE".
```

Example B-6. DDL RECORD Statement for Entry-Sequenced PROCESS File (page 2 of 2)

```
02 recv-qlen           TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "$RCV/RECV/QLEN".
02 recv-qlenm          TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                                HEADING "$RCV/RECV/QLEN/MAX".
02 recv-rate           TYPE binary 64,3         DISPLAY "m<ZZ9.999>"
                                                HEADING "$RCV/RECV/RATE".
02 vbyte-rate          TYPE binary 64,3         DISPLAY "m<ZZZZZ9.9>"
                                                HEADING
                                                  "$RCV/RECV/BYTE/RATE".
02 ybyte-rate          TYPE binary 64,3         DISPLAY "m<ZZZZZ9.9>"
                                                HEADING
                                                  "$RCV/REPLY/BYTE/RATE".
02 link-rate           TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "LINK/RATE".
02 link-fail           TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "LINK/FAIL".
02 links-inuse         TYPE binary 64,3         DISPLAY "m<ZZ9.99>"
                                                HEADING "AVG/LINKS/IN USE".
02 links-inusem        TYPE binary 16 unsigned  DISPLAY "m<ZZ9>"
                                                HEADING "LINKS/IN USE/MAX".
02 chkpt-rate          TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "CHKPT/RATE".
02 accel-busy          TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "ACCEL/BUSY".
02 tns-busy            TYPE binary 64,3         DISPLAY "m<Z9.999>"
                                                HEADING "TNS/BUSY".
02 comptrap-rate       TYPE binary 64,3         DISPLAY "m<ZZZ9.999>"
                                                HEADING "COMPTRAP/RATE".
02 tnsr-busy           TYPE binary 64,3.        DISPLAY "m<Z9.999>"
                                                HEADING :TNSR/BUSY".
02 page-sizeb          TYPE binary 16 unsigned. DISPLAY "m<ZZZ9>"
                                                HEADING $PAGE/SIZE/BYTES".
02 allocseq-rate       TYPE binary 64,3.        DISPLAY "m<ZZZ9.999>"
                                                HEADING "ALLOC/SEG/RATE".
02 UCLqlen             TYPE binary 64,3.        DISPLAY "m<Z9.999>"
                                                HEADING "UCL/QLEN".
02 UCLmax              TYPE binary 16 unsigned. DISPLAY "m<ZZ9>"
                                                HEADING "UCL/MAX".
02 fileopen-rate       TYPE binary 64,3.        DISPLAY "m<ZZZ9.999>"
                                                HEADING "FILE/OPEN/RATE".
02 infocall-rate       TYPE binary 64,3.        DISPLAY "m<ZZZ9.999>"
                                                HEADING "INFO/CALLS/RATE".
key "pc" is cpu-num.
key "pn" is process-name.
key "pf" is program-file-name.name.
                       SEQUENCE IS PID.
end
```

C

Loading Measure Data Into an SQL Table

To create a structured data file and then load the data from the file into an SQL table:

1. Create a data dictionary using the DDL utility and MEASDDLS file:

   ```
   > DDL

   !?dict subvol !
   !?source measddls
   !exit
   ```

2. Use MEASCOM to create a structured output data file. This example creates a file that contains data for all PROCESS entities in the measurement MEASFILE:

   ```
   > MEASCOM

   1+ add measfile
   2+ set report format structured
   3+ list process *, rate off
   ```

3. Use the SQL CONVERT command to create a command file (CNVSRC) that converts the structured file to an SQL table:

   ```
   > SQLCI

   >>convert record process to table $vol2.perf.prodata
   +>catalog $vol2.perf, dictionary $system.sys01, file is
   process
   +>load, source cnvsrc clear;
   >>exit
   ```

4. Edit the CNVSRC file to remove the reserved word GROUP:

   ```
   > edit cnvsrc;&
   > ca/GROUP /GROUP_/all;&
   > exit
   ```

5. Execute CNVSRC to create an SQL table and load data from the structured file to an SQL table:

   ```
   SQLCI

   >>obey cnvsrc
   >>exit
   ```

D Example of Measurement Application in C

When programming in C or C++, refer to the C declarations in files MEASCHMA and MEASDECS to see whether a field is within a union. If the field is within a union, include the union name when writing code that uses the field.

Existing C or C++ programs that use Measure data might not compile using the C structure declarations in new PVUs of Measure's MEASCHMA or MEASDECS. References to fields that now are within a union must be changed to include the union name in the reference.

Example D-1 shows a C program that uses the Measure callable procedures to configure and run a measurement.

Example D-1. Measurement Application in C (page 1 of 4)

```
#include <cextdecs>  nolist
#include <stdio.h>  nolist
#include <string.h> nolist
#include <stdlib.h> nolist
#include "measc" nolist     /* Contains all MEASURE
                               declarations, derived
                               from MEASCHMA */
#include "cmeasddl" nolist  /* contains record structure
                               declarations,
                               derived from MEASDDLS */
int main(int argc, char *argv[])
{
    #define TRUE     -1
    #define FALSE    0
    #define MAX_NUM_MEASUREMENTS   64

 /* all the measure contab definitions */
    typedef struct
    {
        contab_hdr_def Header;
        cpu_desc_def   EDesc;
        contab_trailer_def Trailer;
    } ContabForCPU;

typedef struct
    {
        short FName[12];
    } MeasNamesDef;

    ContabForCPU CPURec;
    meascb_def   MeasCB;
    MeasNamesDef MeasNames[MAX_NUM_MEASUREMENTS];
    cpu_def      CpuData;
```

Example D-1. Measurement Application in C (page 2 of 4)

```
char DataFName[] = "MDATA";
char ExtDataFName[40];
short IntDataFName[12],DataFNum,Error,MeasNum,ActMeasNum;
short cpus,BytesRead;
long long  StartTime,StopTime,Interval,FCall;
short i, Len;
long Entities[MAX_T], CounterSpace[MAX_T];


printf("Initializing contab header record...\n");
CPURec.Header.type = CONTAB_T;
CPURec.Header.len  = sizeof(CPURec);
for (i = 0; i <= MAX_T; ++i)
    CPURec.Header.u_sections.sections[i] = 0;
CPURec.Header.u_sections.sections[CPU_T] =
        sizeof(CPURec.Header);

printf("Initializing CPU record...\n");
CPURec.EDesc.type = CPU_T;
CPURec.EDesc.len  = sizeof(CPURec.EDesc);
CPURec.EDesc.cpu_number = ALL_ELEMENTS;

printf("Initialize the Trailer record\n");
CPURec.Trailer.type = CONTAB_TRAILER_T;
CPURec.Trailer.len  = sizeof(CPURec.Trailer);

    /* Initialize the MEASCB */
memset(&MeasCB,-1,sizeof(MeasCB));

printf("Starting Measure subsystem...\n");
Error = MEASMONCONTROL((short *)&MeasCB,TRUE);
if (Error != 0 && Error != 3217)
{
  printf("MEASMONCONTROL ERROR %d\n",Error);
}

memset(ExtDataFName, 0, sizeof(ExtDataFName));
printf("Creating data file using MEASOPEN...\n");

Error = FILENAME_RESOLVE_(DataFName, 5, ExtDataFName,
                                   sizeof(ExtDataFName),
                                   &Len);

if (Error != 0)
{
  printf("FILENAME_RESOLVE_ Error for %s\n",DataFName);
}
Error = FILENAME_TO_OLDFILENAME_(ExtDataFName, Len,
                                 IntDataFName);
if (Error != 0)
{
  printf("FILENAME_RESOLVE_ Error for %s\n",DataFName);
}
Error = MEASOPEN(IntDataFName,&DataFNum,TRUE,TRUE);
```

Example D-1. Measurement Application in C (page 3 of 4)

```
    if (Error != 0)
    {
    printf("MEASOPEN ERROR %d\n",Error);
    }
    printf("Configuring CPU measurement...\n");
    Error = MEASCONFIGURE((short *)&MeasCB,DataFNum,&MeasNum,
                          (short *)&CPURec);
    if (Error != 0)
    {
      printf("MEASCONFIGURE ERROR %d\n",Error);
    }

    printf("Start Measurement %s",DataFName);
    Error = MEASCONTROL((short *)&MeasCB,MeasNum,
                        CONVERTTIMESTAMP(JULIANTIMESTAMP(),0));
    if (Error != 0)
    {
      printf("MEASCONTROL ERROR %d\n",Error);
    }

    printf("Checking for active measurement...\n");
    Error = MEASMONSTATUS((short *)&MeasCB,&ActMeasNum,
                          (short *)MeasNames);
    if (Error != 0)
    {
      printf("MEASMONSTATUS ERROR %d\n",Error);
    }

    printf("Checking CPU entity is being measured...\n");
    Error = MEASSTATUS((short
*)&MeasCB,MeasNum,&cpus,&StartTime,

&StopTime,&Interval,Entities,CounterSpace);
    if (Error != 0)
    {
      printf("MEASSTATUS ERROR %d\n",Error);
    }

    printf("Stopping measurement...\n");
    Error = MEASCONTROL((short *)&MeasCB,MeasNum,-1,
                        CONVERTTIMESTAMP(JULIANTIMESTAMP(),0));
    if (Error != 0)
    {
      printf("MEASCONTROL ERROR %d\n",Error);
    }

    printf("Closing Datafile...\n");
    Error = MEASCLOSE(DataFNum);
    if (Error != 0)
    {
      printf("MEASCLOSE ERROR %d\n",Error);
    }
```

Example D-1. Measurement Application in C (page 4 of 4)

```
    printf("Opening Data file for analysis...\n");
    Error = MEASOPEN(IntDataFName,&DataFNum,FALSE,TRUE);
    if (Error != 0)
    {
      printf("MEASOPEN ERROR %d\n",Error);
    }

    printf("Doing Analysis...\n");
    FCall = 0;
    do
    {
      Error =MEASREAD(DataFNum,(short*)&CPURec.EDesc,(short*)
             &CpuData,(short)sizeof(CpuData),&BytesRead,
&FCall);
        printf("cpu_num = %d\n",CpuData.cpu_num);
    }while(FCall != 0);

    printf("Stopping Measure subsystem...\n");
    Error = MEASMONCONTROL((short *)&MeasCB,FALSE);
    if (Error != 0 && Error != 3217)
    {
      printf("MEASMONCONTROL ERROR %d\n",Error);
    }
}
```

E

Converting Existing Applications or Enform Reports to ZMS Style Record Formats

This appendix describes how to convert existing Enform reports or existing applications to ZMS-style record formats.

Because legacy-style records are still supported, you are not required to convert existing applications or Enform reports to ZMS-style record formats. However, because ZMS-style record management lets you store data from multiple release levels in a common file structure, you might want to convert.

The process is the same for applications and Enform reports except for two additional considerations when you convert applications.

The conversion procedure depends on whether you want to use new counter fields in the ZMS-style records that do not appear in the legacy-style records.

Using New Counter Fields

To convert to ZMS-style record formats using new records that do not appear in the legacy-style records, you must significantly modify the Enform source:

1. Recompile the Enform programs against a dictionary produced from MEASDDLS.
2. In the ZMS-style naming format, qualify each reference to a record identifier or counter field.

   For example, rename CPU.dispatches to ZMSCPU.ctr.dispatches.

Using Existing Counter Fields

To convert to ZMS-style record formats using only records that appear in the legacy-style records:

1. Recompile the Enform source files using a dictionary built from the MEASDDLZ file.
2. In the Enform source, remove references to these fields (which the recompile left unresolved):

Field	**Reason for Removal**
All max-qlen fields	Max queue values were of questionable use in the legacy interface and are removed in the ZMS style. Measure architecture is moving to a model in which counter records always exist, so such values would represent a maximum since processor reload and could never be reset.
`entity`.ctrl *`entity`*.unit	These attributes of D-series I/O addressing have been invalid on all G-series RVUs. You can list D-series data in ZMS-style records, but D-series addressing attributes are mapped to G-series addressing fields and must be referred to by the G-series names.
CPU.mem-qtime	This counter applies only to data prior to the G05.00 RVU. If analysis of this attribute is required, continue to use the legacy-style interface.
CPU.processor-status	This attribute is in all G-series RVUs but always reports as 16 CPUs, so it provides little value.
CPU.send-busy-time	This counter applies only to D-series data. If analysis of this attribute is required, continue to use the legacy-style interface.
CPU.starting-ucl-lock CPU.ending-ucl-lock CPU.starting-uds-lock CPU.ending-uds-lock CPU.starting-scl-lock CPU.ending-scl-lock CPU.starting-sds-lock CPU.ending-sds-lock PROCESS.lock-pages-qtime PROCESS.ucl-lock-qtime	These legacy-style record counters were reserved but never instrumented.
DISC.seeks DISC.seek-busy-time DISC.ablks-inuse-qtime DISC.cblks-inuse-qtime OPDISK.seeks OPDISK.seek-busy-time	These counters have been obsolete since before D-series RVUs.

Application Conversion Considerations

To obtain ZMS-style records from the callable interface, you must modify the procedure calls to pass the *`templateversion`* parameter as described in the *Measure Reference Manual*.

If the application is written in TAL or pTAL, you might have to convert constants and variable fields used in calculations to FIXED types if the underlying record fields have changed.

Index