A Broadcom Company

DB2 Thread Control Series

Guide and Reference

Thread / STOPPER

and

Thread / SENTRY

Version 7.1

May 2014

Publication TTS-001-28

Relational Architects International

This Guide: RAI Publication TTS-001-28

This document applies to Thread/SERIES Version 7 Release 1 (May 2014), and all subsequent releases, unless otherwise indicated in new editions or technical newsletters. Specifications contained herein are subject to change and will be reported in subsequent revisions or editions.

Purchase Orders for publications should be addressed to:

Documentation Coordinator
Relational Architects International
Riverview Historic Plaza
33 Newark Street
Hoboken NJ 07030

Tel: 201 420-0400
Fax: 201 420-4080
Email sales@relarc.com
www.relarc.com

Reader comments regarding the product, its documentation, and suggested improvements are welcome. A reader comment form for this purpose is provided at the back of this publication.

Ed 14E1

Preface

This manual is intended for the person(s) who will install and/or use the Thread/SERIES program products and tailor them for the target environment. The reader is assumed to be familiar with JCL, DB2, z/OS and REXX along with the programming and operations standards in use at the installation site.

The Thread/SERIES Guide and Reference is organized as follows:

Chapter 1 provides an overview of the product's components and capabilities.

Chapter 2 describes and illustrates how to use the Thread/STOPPER Dialog to monitor and control DB2 application and utility threads. Chapter 2 illustrates the flow of panels presented during a Thread/STOPPER Dialog session and describes the information appearing on these displays. Subsequent sections describe and illustrate the tasks of canceling various kinds of application and utility threads.

Chapter 3 describes the ***commands*** you can issue through both the Thread/ STOPPER Batch and Console facilities. It also describes the ***qualifiers*** used to specify the thread(s) upon which a command will act.

Chapter 4 describes the Thread/STOPPER Batch Facility with which you can specify Thread/STOPPER command(s) within the context of a batch jobstep. Chapter 4 describes and illustrates how to make requests of the Batch Facility using the commands and qualifiers common to both the Thread/STOPPER Batch and Console Facilities. Chapter 4 also documents the initialization parameters with which the Batch and Console Facility programs are invoked. These initialization parameters govern operation of both the Thread/STOPPER Batch and Console Facilities during a particular run.

Chapter 5 focuses on the operator and authorized user interactions with the Thread/STOPPER Console Facility. The Console Facility executes Thread/STOPPER commands entered from an MVS console. Chapter 5 describes how to invoke the Console Facility and issue requests using standard Thread/STOPPER commands and qualifiers.

Chapter 6	describes the Thread/SERIES Cancel Inactive Threads Facility, a utility with which to cancel inactive database access threads within the context of a batch jobstep. This functionality has been relocated to the Thread/STOPPER Batch and Console Facilities (described in Chapters 4 and 5 respectively). As such, Chapter 6 is now obsolete, removed, and reserved for future use.
Chapter 7	describes and illustrates the Thread/SERIES Audit View Facility used to browse the audit trail recorded by Thread/SERIES components for actions taken vis-a-vis DB2 threads. Chapter 7 illustrates the panels displayed by the Audit View Facility and describes the information appearing on these displays.
Chapters 8-9	are reserved for future use.
Chapter 10	introduces Thread/SENTRY -- an automated monitoring and enforcement facility for DB2 threads. Chapter 10 describes how to define your organization's policies as rules that Thread/SENTRY should enforce. Thread/SENTRY makes it unnecessary to manually identify DB2 problem threads or to explicitly cancel them. Instead, you can define what constitutes a 'problem thread' in terms of threshold measurements such as elapsed time, CPU time, number of SQL statements executed, number of locks held, etc. Thread/SENTRY detects such problem threads *automatically* based on your pre-defined criteria.
Chapter 11	discusses Thread/SENTRY operation. Chapter 11 first describes and illustrates the JCL with which to run Thread/SENTRY as a submitted job or started task. Subsequent sections describe the commands with which to communicate with Thread/SENTRY from the console in order to control its runtime operation.
Chapters 12-19	are reserved for future use.
Chapter 20	discusses Thread/SERIES installation in detail. It also briefly describes the files supplied on the product distribution tape.
Appendix A	describes the audit trail maintained for the actions taken by the various Thread/SERIES components. Audit trail information is recorded in a table within the same DB2 subsystem in which an effected thread was executing. This audit trail includes many details about the thread itself as well as information describing why a particular action was taken, by whom, and when it was done.
Appendix B	documents several topics that apply to all Thread/SERIES components. Exceptions and information unique to a specific Thread/SERIES component are noted as appropriate. Appendix B first discusses eligibility criteria for thread cancellation. The next topic describes the different mechanisms used by Thread/SERIES to cancel DB2 threads, followed by a description of thread behavior in response to different cancellation methods. The last topic describes and illustrates the notification messages sent by Thread/SERIES to users whose threads are canceled as well as the messages received by those designated as Thread/SERIES administrators.

Appendix C describes how to create site written messages that will be issued when Thread/SENTRY policies are violated. Appendix C also describes how to customize the text of the *default* notification messages Thread/SERIES components send to users and administrators.

Appendix E describes how to develop Thread/SERIES exit routines to handle non-standard processing requirements.

Appendix F describes how to define thread(s) against which Thread/SERIES should take no action.

Appendix G describes how to target multiple DB2 authorization IDs within a single LIMIT or EXCLUDE Thread/SENTRY policy.

Appendix M describes the messages issued by various Thread/SERIES components.

Appendix Z describes problem determination procedures for the Thread/SERIES components.

Notational conventions

The following notational conventions are used in this Guide:

> Uppercase commands and their operands should be entered as shown but need not be entered in uppercase.

> Operands shown in lower case are variables; a value should be substituted for them.

> Operands shown in brackets [ ] are optional, with a choice indicated by vertical bars |. One or none may be chosen; the defaults are underscored.

> Operands shown in braces { } are alternatives; one must be chosen.

> An ellipsis (. . .) indicates that the parameter shown may be repeated to specify additional items of the same category.

Table of Contents

Section A

What's New in

Thread/SERIES

Summary of Changes to Thread/SERIES Version 7.1

- Thread/SERIES fully supports DB2 Version 11 -- Program Product 5615-DB2.
- The new ADD_ZIIP_DATA(NO|YES) operand of the LIMIT statement is introduced to govern whether or not Thread/SENTRY should include the CPU time consumed on an IBM specialty engine while calculating CLASS1and CLASS2 CPU times.
- The following new LIMIT policy operands are introduced to monitor Multi-Row Fetch SQL activity: MAX_QUERIED, MAX_FETCHED, MAX_CHANGED, UOW_QUERIED, UOW_FETCHED, UOW_CHANGED, IMIN_QUERIED, IMIN_FETCHED, IMIN_CHANGED, IMAX_QUERIED, IMAX_FETCHED and IMAX_CHANGED.
- The Thread/SENTRY notify email subject now contains the target DB2 subsystem ID to make it easier for Thread/SENTRY administrators to distinguish among notifications pertaining to different DB2 subsystems.

Summary of Changes to Thread/SERIES Version 6.1.9 (B1)

- Thread/SENTRY can now optionally externalize the DB2 IFI and z/OS WLM data associated with violations of specially-named policies. The new TTS$TSD keyword parameter DEBUG_POLICY identifies such specially-named policies by the first two characters of the policy name. The default DEBUG_POLICY value "__" directs Thread/SENTRY to externalize the DB2 IFI and z/OS WLM data for any violated policy whose name starts with "__" (P111167B).

- Thread/SENTRY can now optionally write a formatted dump of its internal control blocks to the destination identified by the DUMPDD parameter in the TTS$TSD module. The default DUMPDD value is TTSDUMP. For example, add "//TTSDUMP DD SYSOUT=*" to your Thread/SENTRY execution JCL to direct Thread/SENTRY to produce such a dump (P111167A).

- The new Thread/SENTRY Table of Group ID definitions will optionally enable you to target multiple DB2 authorization IDs with a single LIMIT or EXCLUDE policy. The source module within member TTS$TGI of the TTSCNTL library illustrates how to define the Table of Group ID definitions to Thread/SENTRY. Member TTSJTGI of the TTSCNTL library contains a jobstream with which to assemble and link edit the Thread/SENTRY Table of Group ID definitions (P111012A).

Summary of Changes to Thread/SERIES Version 6.1.9 (A3)

- Thread/SERIES fully supports DB2 Version 10 - Program Product 5605-DB2 (P110284A).

- This release incorporates corrective maintenance for the following Thread/SENTRY (TSN) PMRs:

  P110180A TTSND NOTIFY messages externalize DB2 thread token

  P110069A RFARMX +IRX0043I Error running TMC$EVM, line 18: Routine not found

  P110063B TTSND
  NOTIFY_INACTIVE_{ADMIN|USER}(OFF|WARN|CANCEL|BOTH) - added

  P110027A TTSQPGM Notify target TSO user upon cancel by TSN exit

Summary of Changes to Thread/SERIES Version 6.1.8 (B1)

- This release incorporates corrective maintenance for the following Thread/SENTRY (TSN) PMRs:

  P109221A TTSLEAQ TSN overrides ACTION(WARN) with ACTION(CANCEL)

  P109173C TTSMFV Mask invalid creation date and time with N/C

Summary of Changes to Thread/SERIES Version 6.1.8

- This release incorporates corrective maintenance for the following Thread/SENTRY (TSN) PMRs:

  P109153B RFARMX TSN suffers intermittent IRX0240I errors.

  P109153A RFARMD TSN suffers intermittent S201 ABENDs.

  P109148A TTSQMD New TSN message TTS836I reports on the DROP TCP/IP actions instead of the generic TTS809W message.

  P109147A TTSLTS New TSN message TTS835I diagnoses the '-START TRACE' TTSLTE and '-STOP TRACE' activities.

  P109146A TTSLEACQ TSN implements the action escalation for inactive TTSLEAP (IDLE) threads, i.e. the TSN policies can specify TTSLEAQ ACTION(WARNING,CANCEL) for IDLE threads. TTSLIT

  P109134B TTSLIT TSN suffers intermittent S138 ABENDs.

  P109134A TMCB2OPV TSN Compiler ignores the Thread/SERIES MSGDISP VERBOSE and DEBUG settings in TTS$TSD module.

  P109130A TTSQCCC Drop remote threads using TCP/IP and PORT combo.

  P109012A TTSMAS TSN suffers S0C4 while processing parallel threads. S0C4 ABEND with "ILC 4 INTC 10" in TTSL.TTSMAS.+48C.

  P109131A TTSQDIST TSN makes an attempt to diagnose IFI errors after P108237B DROP TCP/IP request. This attempt fails with S0C4-04 ABEND in TTSL.RSQDIFI.+16A.

Summary of Changes to Thread/SERIES Version 6.1.7

- The ACTION operand of the LIMIT statement now accepts multiple actions, separated by comas. Thread/SENTRY performs one action per wakeup interval while the problem DB2 thread persists. When specified, such action escalation proceeds from WARNING, to CANCEL, to FORCE and lastly to KILL.

- The new AUTO_FORCE operand of the TTS#TSD macro specifies whether or not Thread/SENTRY should use ACTION(FORCE) instead of ACTION(CANCEL) to remove a local thread that is currently executing within the application rather than DB2 (and as such, no SQL statement is being executed).

Summary of Changes to Thread/SERIES Version 6.1.6

- Now Thread/SENTRY accepts sub-second specification for any TIME based limit, e.g. MAX_CPU(00:00:00.04) or MAX_CPU(00:00:00.123456)

- Now Thread/STOPPER Batch and Console facilities cancel all threads holding DB/TS/TB locks.

Summary of Changes to Thread/SERIES Version 6.1.5

- The `NOTIFY_ENABLED`, `NOTIFY_FREQUENCY` and `NOTIFY_MAXIMUM` operands can be specified within the `NOTIFY_LIST` statement. These operands are used to override the corresponding defaults of the `MONITOR` statement or the Thread/SERIES table of system defaults (defined within module TTS$TSD).

- The new `NOTIFY_ON_WARNING(NO|YES)` operand specified on the `MONITOR` statement or defined within the Table of System Defaults (module TTS$TSD) directs Thread/SENTRY to issue (YES) or suppress (NO) notifies for any pending warning requests.

- Thread/SENTRY can optionally reset a pending warning request should a target thread no longer violate the policy that triggered the original warning. Both the `RESET_WARNING` and `NOTIFY_ON_WARNING` defaults must be set to YES in order for Thread/SENTRY to examine pending violations and possibly reset them.

- This release incorporates corrective maintenance for the following PMRs:

PMR	Object(s)	Description
P107250A	TTSQLCL	Cancel immediately any thread not running in DB2
P107285A	TTSMCT	Double storage for parallel thread management
P107318B	TTSFTDS	The message TTS071 is issued erroneously for any thread whose DB2 SSID matches the datasharing group name.

Summary of Changes to Thread/SERIES Version 6.1.4

- E-mail notification to administrators at workstations is now supported via the EMAIL_IDS( ) operand of the NOTIFY_LIST statement

- Full support for DB2 9 for z/OS

- The means to remove the communication connections (TCP/IP and/or SNA/VTAM) that underlie a distributed thread. This mechanism is particularly suitable for removing Database Access Threads that are inactive (also know as idle threads).

- Full support for IFCID data returned in %U (Unicode) format. Such data is processed properly (and displayed meaningfully) whether returned by DB2 in Unicode or standard format.

- The new "D" option (entered from the scrollable list of DB2 subsystems) displays more detailed information about a particular DB2 subsystem or Data Sharing Group.

- New panel TTSTDSG displays DB2 subsystems from a Data Sharing Group perspective

- New panel TTSTDSD presents a view of DB2 subsystems that displays the date and time when the subsystems were started

- A new TTSD dialog invocation parm governs which view is presented in the DB2 subsystem display. The choices include the classic view (panel TTSTDS). the new Data Sharing Group view (displayed via panel TTSTDSG) or the new Date/TIme view (displayed via panel TTSTDSD).

- New panel TTSDSSS displays detailed data associated with a particular DB2 standalone subsystem (i.e. a subsystem that is not a member of a Data Sharing Group).

- New panel TTSDDSG displays detailed data associated with a particular subsystem member of a DB2 Data Sharing Group/

- Support for NOTIFY_LIST names that begin with the special string '_WTO'. Ordinarily, when a thread violates a Thread/SENTRY policy, the owner of the violating thread is notified of the action taken by Thread/SENTRY. In addition, any administrative users specified on the NOTIFY_LIST statement to which the violated policy makes reference are also notified. However, when the referenced NOTIFY_LIST name starts with the special string "_WTO" (as in _WTOABC), Thread/SENTRY writes the user notification message to the operator (via a WTO), rather than send a notification message to the owner of the thread. Notifications sent to administrative users specified in the NOTIFY_LIST are processed without change (not effected by the _WTO special string).

- The new WTO_HRDCPY operand is introduced to direct whether or not WTO notify messages should be queued for hard copy only. See the Thread/SERIES Table of System Defaults (residing in the TTS$TSD member of the TTSCNTL library) to alter the default setting WTO_HRDCPY=NO.

Summary of Changes to Thread/SERIES Version 6.1.0

- Support for data returned by DB2 in Unicode format from the catalog, control blocks and/or IFI data
- Support for long variable names such as those for DB2 authorization ID, location, package, collection, program, and procedure as well as for long schema / qualifier names.
- CPU TIME CLASS 1 calculations are revised to include the CPU TIME associated with stored procedures, UDFs, triggers and WLM enclaves
- The modules comprising the Thread/STOPPER Batch and Console facilities are refreshed.

Summary of Changes to Thread/SERIES Version 5.3.7 Level B1

Version 5.3.7 Level B1 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Support for DB2 Version 8.
- Thread/SENTRY now restarts an accidentally stopped TRACE monitor class 1 and resumes normal processing.

Summary of Changes to Thread/SERIES Version 5.3.7

Version 5.3.7 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Thread/SENTRY now can RESET a previously QUIESCED address space even when it no longer maintains a DB2 thread.
- All previously QUIESCED address spaces are RESET when Thread/SENTRY shuts down.

See the description of the Thread/SENTRY command MCS_CONSOLE in section 10.4 of the Thread/SERIES User Guide and Reference for more information about the QUIESCE and RESET processing.

Summary of Changes to Thread/SERIES Version 5.3.6

Version 5.3.6 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Thread/SENTRY now can be started before any DB2 subsystem is active, e.g. at IPL time like any other started task.

- Thread/SENTRY listener processes suspend their execution when the DB2 subsystem they monitor terminates. A listener process resumes monitoring once its target DB2 subsystem becomes active again.

Summary of Changes to Thread/SERIES Version 5.3.5

Version 5.3.5 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Thread/SENTRY LIMIT policies let you limit the number of GETPAGE requests any thread can perform against a particular DB2 buffer pool as in the following example:

```
LIMIT

PID(POLICY1)
ACTION(CANCEL)  MAX_BUFFER_POOL_ID(2)  MAX_GETPAGES(10000)
```

  POLICY1 above directs Thread/SENTRY to CANCEL any thread which issues more than 10,000 GETPAGE requests against buffer pool BP02.

Summary of Changes to Thread/SERIES Version 5.3.4

Version 5.3.4 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- The operational scope of Thread/STOPPER Batch and Console facilities now extends to DB2 threads within a DB2 data sharing group (DSG).

- Thread/SERIES can now cancel DB2 threads whose token value exceeds 99,999.

Summary of Changes to Thread/SERIES Version 5.3.3

Version 5.3.3 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Thread/SENTRY LIMIT and EXCLUDE policies can now specify a DB2 stored procedure name as in the following examples:

```
EXCLUDE
        PID(POLICY1) STORED_PROCEDURE(SP1)

LIMIT
        PID(POLICY2) STORED_PROCEDURE(SP2%)
```

  POLICY1 above directs Thread/SENTRY to EXCLUDE any thread which is CURRENTLY executing in the stored procedure named SP1.

  POLICY2 above directs Thread/SENTRY to evaluate any thread which is CURRENTLY executing in a stored procedure with a name like SP2%. The LIMITs defined for this policy are applicable ***ONLY*** when the thread is CURRENTLY active within a stored procedure whose name matches the pattern SP2%

  > ***NOTE:*** *stored procedure may run in either a WLM or DB2 established Stored Procedure Address Space.*

- The ***Table of No Actions*** now lets you exclude threads based on a stored procedure name as in the following example:

```
TTS#TNA STORED_PROCEDURE=RAI%,     Never FORCE a stored procedure +
ACTION_SUPPRESSED=FORCE   when its name is like 'RAI%'
```

- The Thread/STOPPER dialog displays the name of DB2 stored procedure name (if any).

Summary of Changes to Thread/SERIES Version 5.3.2

Version 5.3.2 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- You can issue a CANCEL command against a DB2 thread running under a DB2 system services address space.

Summary of Changes to Thread/SERIES Version 5.3.1

Version 5.3.1 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Thread/SENTRY LIMIT policies let you exclude one or more threads based on ***NOT*** criteria

```
LIMIT
        PID(WHATNOT)
        PLAN(\MYPLAN)
        MAX_SQL(10)
```

  This policy directs Thread/SENTRY to cancel all threads using any DB2 plan ***except*** `MYPLAN`.

  The "\" sign (denoting NOT) may be specified ahead of any operand of the LIMIT statement that identify the thread(s) to which the policy applies. See the Thread/SERIES User Guide and Reference paragraphs 10.7.1 and 10.7.1.1 for the list of such operands.

Summary of Changes to Thread/SERIES Version 5.3

Version 5.3 of Thread/SERIES supports all functions available in prior releases of the product, plus the following enhancements:

- Support for DB2 Version 7

- Thread/SERIES now lets you to predefine criteria for which Thread/SERIES actions should be implicitly suppressed. An action is suppressed when a thread selected by a Thread/SERIES component matches an entry in a site defined ***Table of No Actions*** - TTS$TNA. Thread/SERIES provides three macros for the purpose of defining threads for which one or more actions are inappropriate.

- Enhanced support for CICS / DB2 Threads

  Thread/STOPPER support for CICS / DB2 threads has been enhanced. The standard Thread/SERIES CANCEL and FORCE commands have been enhanced to issue a series of DB2, MVS, CICS and communications commands to remove CICS / DB2 threads.

- Additional Thread Control Commands

  Users of the Thread/STOPPER dialog can now enter a "/" or "?" in the Thread Command field to display a prompting popup window that enumerates and describes the various Thread/STOPPER commands.

  In addition, the new P and K commands let users of the Thread/STOPPER dialog issue CICS CEMT SET TASK(taskid) PURGE and FORCEPURGE respectively.

- An enhanced **R**AI **T**hread/SERIES **S**erver (whose default name is RTS0) replaces the generalized RAI server (whose default name RAI0). Customers upgrading from a previous Thread/SERIES release MUST install this customized server (whose default name is RTS0).

- The resource profile RAI.TTS.TCAN becomes a required profile definition to cancel DB2 threads using Thread/SERIES.

Chapter 1

Thread/SERIES Overview

Thread/SERIES provides a set of dedicated facilities to *gracefully cancel* DB2 threads, and do so with minimal impact. Thread/SERIES lets you deal with the inevitable operational mistakes, network failures, application errors and runaway queries that create 'problem' DB2 threads -- *without* disrupting large numbers of users.

Thread/SERIES cancels a DB2 thread without terminating the thread's allied address space. Thread cancellation via Thread/SERIES is more granular and far less drastic than recycling an entire CICS or IMS transaction processing region's connection to DB2 via STOP and START commands. Thread/SERIES is also more granular in operation than the MVS CANCEL and FORCE commands. While these commands terminate an entire job, TSO user or started task, Thread/STOPPER cancels only the task associated with a DB2 thread.

Moreover, Thread/SERIES works in cases where the DB2 commands CANCEL THREAD and CANCEL DDF THREAD are ineffective. This can occur when the thread is hung up in the application or within the communications network (TCP/IP and/or VTAM).

> ***DISCUSSION:*** *The DB2 CANCEL THREAD command schedules a thread to terminate. Cancellation is immediate for local threads. In contrast, a distributed thread must be processing within DB2 in order for the DB2 CANCEL THREAD command to terminate the distributed thread. Otherwise the* ***thread does not terminate****. Threads suspended in communications software like TCP/IP or VTAM must return control to DB2 before these threads can be terminated. In these cases, Thread/SERIES components automatically detect such 'suspended in communication software' states and cancels the thread's communications sessions.*

Thread/SERIES supports various methods of thread cancellation using DB2, MVS, TCP/IP and/or SNA/VTAM commands -- both individually and in combination. It automatically determines the correct method to use, based on the current state of the DB2 thread.

1.1 Types of DB2 threads that can be terminated

Thread/SERIES is an enterprise solution that supports all *local* and *distributed* DB2 applications connected through the various DB2 attachment facilities. These include client server applications, CICS and IMS transactions, TSO and QMF sessions, and DB2 batch jobssteps connecting through either RRSAF or CAF. Moreover, Thread/SERIES supports cancellation of DB2 utility jobsteps via an IFI interface to the DB2 TERM UTILITY command. In addition, it

- supports termination of threads that access the local DB2 subsystem
- supports cancellation of *distributed threads* that utilize both system and application directed database access
- supports cancellation of *idle Database Access Threads (DBATs)*

1.2 Thread/SERIES Benefits

- Improves availability of critical Transaction Processing systems
- Provides the means to cancel local (non-distributed) DB2 threads, and do so with minimal impact
- Enables TSO users with canceled DB2 threads to remain in session
- Improves operational control of the DB2 environment
- Provides a *single point of control* for all DB2 application and utility threads. You can monitor and control multiple DB2 subsystems from a single console.
- Does not adversely impact other transactions within an transaction processing region. Thread/SERIES is more granular than the DSNC STOP and STRT commands that recycle an entire CICS attachment or the /STOP and /START SUBSYS commands that recycle the IMS connection to DB2
- Prevents disruptions from problem DB2 threads
- Avoids forcible cancellation of TSO sessions that hold DB2 threads
- Provides the means to cancel idle DB2 threads that hold DB2 locks and thereby release their resources
- Lets you obtain diagnostic dumps as an option

1.3 Benefits versus native DB2 Facilities

Thread/SERIES affords greater ease of use than the DB2 CANCEL THREAD and CANCEL DDF THREAD commands. It

- presents more information, in a much more meaningful format, than does DB2
- operates as an ISPF dialog rather than as a line mode command
- translates codes into meaningful information
- translates ASID values into jobnames
- affords granular security options -- over and above restricting the ability to cancel threads to those with DB2 SYSOPR, SYSCTRL or SYSADM authority

1.4 Thread/SERIES Components

Thread/STOPPER Dialog

The Thread/STOPPER Dialog provides a series of ISPF panels with which to monitor and, if necessary, ***gracefully cancel*** DB2 threads

Batch Facility

The Thread/STOPPER Batch Facility allows you to specify a single Thread/STOPPER command or set of commands within the context of a batch jobstep.

Console Facility

The Thread/STOPPER Console Facility allows MVS Console operators and other authorized users to issue Thread/STOPPER commands from an MVS console. The Console Facility runs as a started task or can be submitted as a conventional MVS job.

Thread/SENTRY

Thread/SENTRY provides *automated* monitoring and policy enforcement for DB2 threads. Thread/SENTRY makes it unnecessary to manually identify DB2 problem threads or to explicitly cancel them. Instead, you can define what constitutes a 'problem thread' in terms of threshold measurements such as elapsed time, CPU time, number of SQL statements executed, number of locks held, etc. Thread/SENTRY detects such problem threads *automatically* based on criteria you pre-define.

Audit View Facility

The Thread/SERIES Audit View Facility provides a means to browse the audit trail that Thread/SERIES records for each action it takes vis-a-vis DB2 threads. The audit trail is maintained in a table named THREAD_AUDIT that resides within the same DB2 subsystem in which either the audited thread was executing or from which a Thread/SERIES command was issued. Some columns of the Thread_Audit table provide statistics and details about the thread while other columns document who took action against a thread, when, and for what reason.

The Audit View Facility first presents a Query by Example panel with which you can identify the Thread Audit records you wish to view. This QBE style dialog allows you to select rows within the Thread_Audit table without the need to construct SQL select statements. You can also specify an optional sort order and direction for each column.

Chapter 2

Using the Thread/STOPPER Dialog

This chapter describes how to use the Thread/STOPPER Dialog to monitor and, if necessary, *gracefully cancel* DB2 threads. Section 2.1 describes how to invoke the Thread/STOPPER Dialog once it has been installed. Sections 2.2 though 2.5 describe and illustrate the flow of panels presented during a Thread/STOPPER Dialog session as well as the information appearing on the subsystem summary, subsystem detail, thread summary and thread detail displays respectively.

The last three sections of this chapter discuss how to cancel different types of DB2 threads. Sections 2.6, 2.7 and 2.8 each contain an annotated example that illustrates how to terminate local threads, distributed threads and IBM DB2 utility threads, respectively. In each case, you simply select the thread you want to cancel, then respond affirmatively to the thread cancellation prompt.

2.1 Invoking the Thread/STOPPER Dialog

The Thread/STOPPER Dialog can be invoked with a variety of options, modes and parameters (as described in Chapter 20.15 of this publication). For example, the dialog can operate in either of two modes: The first mode connects to a *single* DB2 subsystem while the second mode supports *simultaneous* connections to *multiple* DB2 subsystems within the local MVS system. Section 20.15 describes and illustrates how to configure the TTSRUN exec with which the Thread/STOPPER Dialog is invoked.

Alternatively, the Thread/STOPPER Dialog may have been setup as an ISPF selection menu option (as described in Section 20.17.) In this case, see your Thread/SERIES product administrator for details concerning invocation of the Thread/STOPPER Dialog.

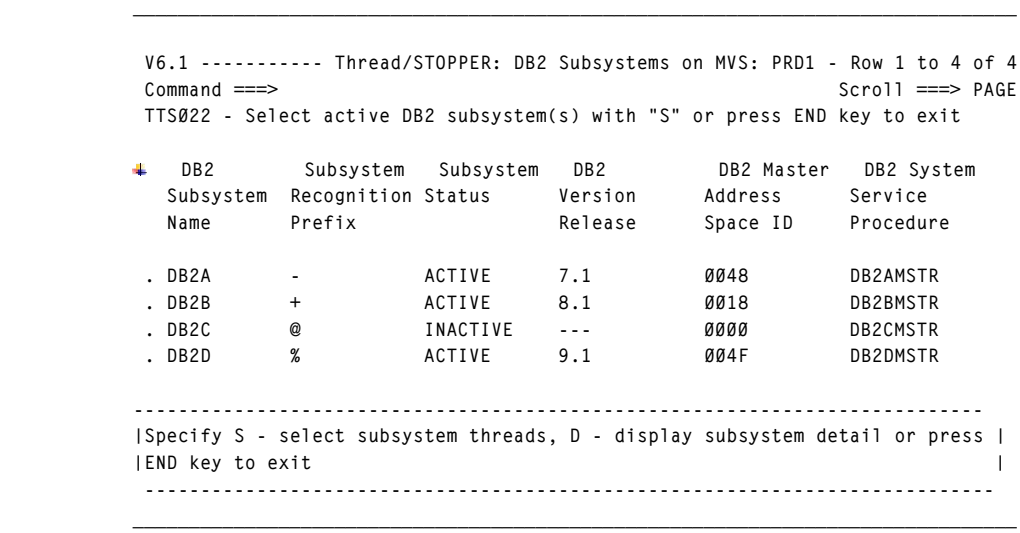
2.2 DB2 Subsystem Displays

The DB2 Subsystem Summary panels are described and illustrated in this Section.

Figure 2.1 illustrates a scrollable summary of all the DB2 subsystems defined within the local MVS system. Figure 2.1 presents the original or *Classic* view of the subsystem summary. Section 2.2.1 describes and illustrates two alternative views which present subsystem summary information from first a *"data sharing group"* perspective and then from the perspective of subsystem *"starting date and time"*.

All the subsystem summary displays include both active and inactive subsystems. In each case, select all subsystems you wish to monitor and control by keying an 'S' into the subsystem's row selection field. Alternatively, key a 'D' into the subsystem's row selection field to display more detailed information about a particular DB2 subsystem (as described and illustrated in Section 2.2.3).

> ***NOTE:*** *These displays pertain only to Thread/STOPPER's multiple subsystem mode of operation. In single subsystem mode, these panels are bypassed (not displayed). Instead, the DB2 subsystem is specified either on the Thread Qualification panel discussed in Section 2.2.2 or as a parameter when the Thread/STOPPER Dialog program is invoked (Section 20.15).*



```
V6.1 ----------- Thread/STOPPER: DB2 Subsystems on MVS: PRD1 - Row 1 to 4 of 4
Command ===>                                                  Scroll ===> PAGE
TTS022 - Select active DB2 subsystem(s) with "S" or press END key to exit

   DB2         Subsystem   Subsystem   DB2          DB2 Master   DB2 System
  Subsystem  Recognition Status      Version      Address      Service
  Name       Prefix                  Release      Space ID     Procedure

. DB2A       -           ACTIVE      7.1          0048         DB2AMSTR
. DB2B       +           ACTIVE      8.1          0018         DB2BMSTR
. DB2C       @           INACTIVE    ---          0000         DB2CMSTR
. DB2D       %           ACTIVE      9.1          004F         DB2DMSTR

-------------------------------------------------------------------------
|Specify S - select subsystem threads, D - display subsystem detail or press |
|END key to exit                                                            |
 -------------------------------------------------------------------------
```

Figure 2.1 DB2 Subsystem Summary- "Classic View"

The column headings in Figure 2.1 denote the following:

DB2 subsystem name	name by which the subsystem is known to MVS
Subsystem recognition character	used to pass commands to the subsystem
Subsystem status	indicates whether the DB2 subsystem is currently active or inactive
DB2 Version / Release	the version and release level of the DB2 subsystem. This value appears as dashes ("---") if the subsystem is not active

DB2 Master Address space ID	Four hexadecimal digits which appear as 0000 if the subsystem is inactive
Procedure name	Started procedure name associated with the DB2 system services (master) address space

2.2.1 Alternate Views of the DB2 Subsystem Summary

Figure 2.2 illustrates another scrollable summary of DB2 subsystems know to the local MVS system, this time from a *"data sharing group"* perspective.

```
V6.1 -------------- Thread/STOPPER: DB2 seen from z/OS Z7L1  Row 1 to 11 of 11
Command ===>                                                  Scroll ===> HALF

                                                          - Data Sharing Group
  DB2  Command  DB2 SSID     IRLM IRLM     ZPARM   IFCID  Group    Attach Mbr
  SSID Prefix   Ver Status   SSID Procname module  Encode Name     Name   LPAR

. D8G               DSGroup                              D8G      D8G
. D8GB -D8GB    --- Inactive                                      D8G
. D8GC -D8GC    --- Inactive                                      D8G
. D8GD -D8GD    --- Inactive                                      D8G
. D8G  -D8G     --- Inactive                                      D8G
. DSN9 -DSN9    9.1 Active   IRL9 DSN9IRLM DSNZPARM  U
. DB9A -DB9A    9.1 Active   IR9A DB9AIRLM DSNZPARM  U
. DSN8 -DSN8    8.1 Active   IRL8 DSN8IRLM DSNZPARM  U
. DB6B -DB6B    --- Inactive
. DSN7 -DSN7    7.1 Active   IRL7 IRL7PROC DSNZPARM  S
. D8GA -D8GA    8.1 Active   D8JA D8GAIRLM DSNZD8GA  S    D8G      D8G    Z7L1

-------------------------------------------------------------------------------
|Specify S - select subsystem threads, D - display subsystem detail or press |
|END key to exit                                                              |
```

Figure 2.2 DB2 Subsystem Summary- Data Sharing View

The column headings described in the *"Classic View"* also apply to 'Data Sharing Group' view illustrated in Figure 2.2. The additional column headings associated with this view denote the following:

IRLM SSID	specifies the name by which the IRLM subsystem associated with this DB2 subsystem is known to MVS.
IRLM ProcName	identifies the started procedure name associated with the IRLM instance associated with this DB2 subsystem
ZPARM module	identifies the name of the load module from which a DB2 subsystem obtains its runtime parameters. Note that in a data sharing group, each member has its own subsystem parameters module.
IFCID Encoding	specifies how DB2 returns IFCID data from fields that may contain Unicode values. IBM denotes these IFCID fields in the DB2 machine readable materials with the %U designation. A "U" appearing in the row for a particular DB2 subsystem indicates that %U data is returned in Unicode format (which Thread/SERIES translates to a viewable format)). Alternatively, when an "S" appears in the row for a particular DB2 subsystem, this indicates the DB2 subsystems return %U data in standard format (which Thread/SERIES need not translate).

The following fields apply only to DB2 Data Sharing Groups and their members. For standalone DB2 subystems, these values are blank.

Group Name	identifies the name that encompasses the entire DB2 data sharing group.
Group Attach Name	identifies a "generic" attachment name to be used by the TSO/batch attachment, the call attachment facility (CAF), DL/I batch, utilities, and the Resource Recovery Services attachment facility (RRSAF).
Mbr LPAR	identifies the name of the z/OS LPAR (MVS system) on which this member was last active.

Lastly, Figure 2.3 illustrates a scrollable summary of DB2 subsystems know to the local MVS system that includes the *date and time"* each active subsystem was started.

```
V6.1 ------------ Thread/STOPPER: DB2 as seen from z/OS: Z7L Row 1 to 11 of 11
Command ===>                                                 Scroll ===> HALF

  DB2  Command Subsystem DB2 IRLM IRLM     Starting                    IFCID
  SSID Prefix  Status    Ver SSID ProcName Date/Time                   Encoding

. D8G          DSGroup
. D8GB -D8GB   Inactive  ---
. D8GC -D8GC   Inactive  ---
. D8GD -D8GD   Inactive  ---
. D8G  -D8G    Inactive  ---
. DSN9 -DSN9   Active    9.1 IRL9 DSN9IRLM Ø5/31/xxxx-17:Ø3:14              U
. DB9A -DB9A   Active    9.1 IR9A DB9AIRLM Ø6/11/xxxx-14:5Ø:2Ø              U
. DSN8 -DSN8   Active    8.1 IRL8 DSN8IRLM Ø5/31/xxxx-16:42:41              U
. DB6B -DB6B   Inactive  ---
. DSN7 -DSN7   Active    7.1 IRL7 IRL7PROC Ø5/31/xxxx-16:42:41              S
. D8GA -D8GA   Active    8.1 D8JA D8GAIRLM Ø6/Ø5/xxxx-12:Ø5:14              S

-------------------------------------------------------------------------
|Specify S - select subsystem threads, D - display subsystem detail or press |
|END key to exit                                                            |
 -------------------------------------------------------------------------
```

Figure 2.3 DB2 Subsystem Summary – Date / Time View

The column headings described in both the *"Classic"* and *'Data Sharing Group'* views also apply to the *'Date/Time'* view illustrated in Figure 2.3. The additional column headings associated with this view denote the following:

Starting Date/Time indicates when this active DB2 subsystem was started. The Date appears first in mm/dd/yyyy format followed by the subsystem's starting time in hh:mm:ss format.

2.2.2 DB2 Subsystem Commands

All the subsystem summary displays include both active and inactive subsystems. In each case, select all subsystems you wish to monitor and control by keying an 'S' into the subsystem's row selection field. Alternatively, key a 'D' into the subsystem's row selection field to display more detailed information about a particular DB2 subsystem.

In Figure 2.4, the DB2 subsystems named DB2A and DB2B are selected. Thus, Thread/STOPPER will display a Thread Qualification panel (as described in Section 2.4) from which to select threads associated with the two selected subsystems.

```
---------- Thread/STOPPER: DB2 Subsystems on MVS: PRD1 - Row 1 to 4 of 4
 Command ===>                                            Scroll ===> PAGE
 TTS022 - Select active DB2 subsystem(s) with "S" or press END key to exit

   DB2        Subsystem   Subsystem   DB2          DB2 Master   DB2 System
   Subsystem  Command     Status      Version      Address      Service
   Name       Prefix                  Release      Space ID     Procedure

 S DB2A       -           ACTIVE      7.1          0048         DB2AMSTR
 S DB2B       +           ACTIVE      8.1          0018         DB2BMSTR
 . DB2C       @           INACTIVE    ---          0000         DB2CMSTR
   DB2D       %           ACTIVE      9.1          004F         DB2DMSTR
```

Figure 2.4 DB2 Subsystem Display - after Selections

You can continue to scroll and select DB2 subsystems as long as necessary. Once you have selected all the DB2 subsystems you wish to monitor, press the Enter key to proceed to the Thread Qualification panel. Alternatively, enter the END command without selecting any subsystems to exit the Thread/STOPPER dialog entirely.

2.2.3 DB2 Subsystem Detail Displays

This section illustrates the displays associated with individual DB2 subsystems and Data Sharing Groups. The first two panels pertain to *standalone* DB2 subsystems (i.e. those not associated with or members of a DB2 data sharing group) while Figures 2.7, 2.8 and 2.9 illustrate the three DB2 Data Sharing Group displays.

```
V6.1 ----------- Thread/STOPPER: Standalone DB2 Subsystem Detail -------------
Command ===>


DB2 Subsystem Data
   DB2 Subsystem       ===> DSN9
   DB2 Version:Release ===> 9.1
   Subsystem status    ===> Active
   ZPARM module name   ===> DSNZPARM
   Prefix scope:set    ===> S        S-Sysplex:startup,M-Sysplex:IPL,X-Sys:IPL
   DB2 command prefix  ===> -DSN9
   DB2 starting date   ===> Ø5/31/xxxx
   DB2 starting time   ===> 17:Ø3:14
   IFCID Data encoding ===> U        (U)nicode or (S)tandard for %U data
   Local site name     ===> RADSN9
   Local LU name       ===> DB2DSN9

DB2 / IRLM Address Spaces
   DB2MSTR proc name   ===> DSN9MSTR
   DB2MSTR ASID        ===> ØØ4B
   STOKEN for DB2MSTR  ===> ØØØØØ12C
   ALET for DB2MSTR    ===> 16842834
   DB2DBM1 ASID        ===> ØØ4D
   STOKEN for DB2DBM1  ===> ØØØØØ134
   ALET for DB2DBM1    ===> 16842835
   IRLM Subsystem Name ===> IRL9
   IRLM Procedure Name ===> DSN9IRLM
```

Figure 2.5 Detail of an active, standalone DB2 Subsystem

```
V6.1 ------------ Thread/STOPPER: Inactive DB2 Subsystem Detail ---------------
Command ===>

Inactive DB2 Subsystem
   DB2 Subsystem       ===> DSN9
   DB2 Version:Release ===> ---
   Subsystem status    ===> Inactive
   Prefix scope:set    ===> S        S-Sysplex:startup,M-Sysplex:IPL,X-Sys:IPL
   DB2 command prefix  ===> -DSN9
   DB2MSTR proc name   ===> DSN9MSTR
```

Figure 2.6 Detail of an inactive, standalone DB2 Subsystem

The next three figures illustrate the DB2 Data Sharing Group displays. Figure 2.7 below displays information about the DB2 Data Sharing Group itself, separate and apart from the DB2 member subsystems that comprise it. Figures 2.8 and 2.9 respectively illustrate an active, and inactive subsystem member of a DB2 data sharing group.

```
V6.1 ---------------- Thread/STOPPER: DB2 Data Sharing Group -----------------
Command ===>

DB2 Data Sharing
   Group name          ===> D8G
   Group attach name   ===> D8G
   Number of Members   ===> 00000002 in the Data Sharing Group D8G
   Group Level ID      ===> 8        Data Sharing Group Level ID
   Coupling Facility   ===> 14       Level (shared by the Group)
   SCA Size            ===> 2560     shared by the group
```

Figure 2.7 Detail of the Data Sharing Group (as a whole)

```
V6.1 ----------- Thread/STOPPER: Data Sharing Group Member Detail ------------
 Command ===>


 DB2 Subsystem Data
    DB2 Subsystem       ===> D8GA
    DB2 Version:Release ===> 8.1
    Subsystem status    ===> Active
    ZPARM module name   ===> DSNZD8GA
    Prefix scope:set    ===> S        S-Sysplex:startup,M-Sysplex:IPL,X-Sys:IPL
    DB2 command prefix  ===> -D8GA
    DB2 starting date   ===> Ø6/Ø5/xxxx
    DB2 starting time   ===> 12:Ø5:14
    IFCID Data encoding ===> S        (U)nicode or (S)tandard for %U data
    Local site name     ===> D8GLID
    Local LU name       ===> DB2D8GA

 DB2 / IRLM Address Spaces
    DB2MSTR proc name   ===> D8GAMSTR
    DB2MSTR ASID        ===> ØØ56
    STOKEN for DB2MSTR  ===> ØØØØØ158
    ALET for DB2MSTR    ===> 16842852
    DB2DBM1 ASID        ===> ØØ59
    STOKEN for DB2DBM1  ===> ØØØØØ164
    ALET for DB2DBM1    ===> 16842853
    IRLM Subsystem Name ===> D8JA
    IRLM Procedure Name ===> D8GAIRLM

 DB2 Data Sharing
    Group name          ===> D8G
    Group attach name   ===> D8G
    Number of Members   ===> ØØØØØØØ2 in the Data Sharing Group D8G
    XCF member name     ===> D8GA     DB2 member name in the XCF group
    Group Level ID      ===> 8        Data Sharing Group Level ID
    Member active LPAR  ===> Z7L1     z/OS LPAR on which this member is active
    Member state        ===> Active
    Name of the Primary ===> D8GAMSTR ASID which issued the XCF JOIN macro
```

Figure 2.8 Detail of an active member of a DB2 Data Sharing Group

```
V6.1 -------- Thread/STOPPER: Inactive Member of a Datasharing Group ---------
Command ===>

DB2 Subsystem Data
   DB2 Subsystem       ===> D8GB
   DB2 Version:Release ===> ---
   Group attach name   ===> D8G
   Subsystem status    ===> Inactive
   Prefix scope:set    ===> S        S-Sysplex:startup,M-Sysplex:IPL,X-Sys:IPL
   DB2 command prefix  ===> -D8GB
   DB2MSTR proc name   ===> D8GBMSTR
```

Figure 2.9 Detail of an inactive member of a DB2 Data Sharing Group

2.3 Thread Qualification panel

The Thread/STOPPER Dialog lets you select ***which*** DB2 threads to monitor and display on the basis of various criteria. The qualification panel (illustrated in Figure 2.10) lets you specify constraints on the threads to be monitored (and possibly canceled). If you specify no qualification criteria, information is obtained for ***all*** allied and database access threads associated with the specified subsystem(s).

The DB2 subsystem prompt (appearing in underlined type in Figure 2.10) is displayed only when Thread/STOPPER is operating in single subsystem mode. In multiple subsystem mode, the DB2 subsystems whose threads you wish to monitor and control are selected via the DB2 Subsystem Summary panels described in previous sections.

```
------------------- Thread/STOPPER: DB2 Thread Qualification ------------------
 Command ===>

 Select DB2 threads using full or partial qualifications (except as noted)
    DB2 subsystem   ===>
    DB2 plan name   ===>              (e.g. XYZ = all plans starting with 'XYZ')
    Auth ID         ===>              (Current primary authorization ID)
    Operator ID     ===>              (Original operator ID)
    Connection name ===>              (e.g. TSO, BATCH, DB2CALL, UTILITY)
    Correlation ID  ===>
    Addr space ID   ===>              (4 hex digits - no partial qualifiers )
    MVS job name    ===>              (Thread job name - no partial qualifiers )
    Location name   ===>

 Identify threads by Logical Unit of Work ID (composite of following fields)
    Network ID      ===>
    LU name         ===>
    Thread number   ===>              (Uniqueness value - 12 hex digits)

.-------------------------------------------------------------------------------.
| TTS030 - Qualify which DB2 threads are of interest and press ENTER.  To see  |
| all threads, press ENTER without supplying any qualifications.  Press the    |
| END key to return to the DB2 subsystem selection display.                    |
.-------------------------------------------------------------------------------.
```

Figure 2.10 Thread Qualification Panel

NOTE: *You can partially qualify all of these fields (except for Address Space ID and MVS job name) with just significant characters. Thread/ STOPPER pads the remainder of each qualification field with binary zeros. For example, you can specify a generic plan name like ABCD to monitor all plans whose names start with the characters ABCD.*

The set of thread qualifiers denote the following:

DB2 Subsystem	denotes a 4 byte character field which identifies where the thread(s) were established. This field is presented ***only*** in single subsystem mode. It is ***not*** presented in multiple subsystem mode.
DB2 Plan name	denotes an 8 byte character field whose significant characters identify the DB2 plan name(s) desired.
Auth ID	identifies the 8 character primary authorization ID of the desired thread(s).
Operator ID	denotes the 8 character original authorization ID of the thread(s).
Connection name	identifies the 8 character DB2 connection type.
Correlation ID	denotes a 12 byte field whose significant characters identify the Correlation ID of the desired thread(s)
Address Space ID	4 Hex digits specify the address space identifier (ASID) of the thread's allied address space. ***The ASID qualifier does not support generic search arguments.***
MVS job name	denotes the name of the job associated with the allied address space from which one or more thread(s) originate. ***The job name qualifier does not support generic search arguments.***
Location Name	this 16 byte character field identifies the name of the location from which distributed agents originate.
LUW ID	this 24 byte character field identifies the desired thread(s) by their logical-unit-of-work ID.

2.4 Thread Summary Display

The next panel the user sees appears in Figure 2.11 which illustrates the scrollable summary of threads which meet the qualification criteria. From this display, you can select a thread in order to display detailed information about it and, if necessary, cancel it. Alternatively, you can press ENTER to refresh the display.

Key an 'S' into a thread's row selection field (as in the line whose type is underlined) in order to display the Thread Detail panel discussed in Section 2.5.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 5 of 5
 Command ===>                                                 Scroll ===> PAGE
 TTS053 - (S)elect a thread or press ENTER to refresh summary display

   Date: xx/10/25   Time: 16:08       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2B RAI6        RLXPLANC RAI6     DB2CALL  00:54:04 0:00:002      118 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:26:36 0:00:000        7 TR:PGM
   DB2B RAI009D     DDFSMP1  RAI009   DB2CALL  N/A                     24 RA:IDL
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:28:05 0:00:001       14 T :PGM
 S DB2A RAIC        EXAMPLE  RAIC     DB2CALL  00:00:23 0:00:016       22 T :DB2
 . DB2B RAIC        EXAMPLE  RAIC     DB2CALL  00:00:23 0:00:002       28 T :DB2
```

Figure 2.11 Thread Summary Display

The column headings of the Thread Summary display denote the following:

DB2 the name of the DB2 subsystem in which the thread exists

Correlation a 1 to 12 character recovery 'correlation-id' associated with the thread

DB2 plan the 1 to 8 character name of the DB2 plan associated with the thread

Auth ID The DB2 authorization ID associated with the thread

Connect Name identifies the 1 to 8 character DB2 connection type

Elapsed time represents the elapsed 'wall clock' time since the thread was created in hours, minutes and seconds

TCB Time represents the amount of TCB CPU time expended so far by the thread, both within the application and within DB2 . For database access agents this value represents TCB time for the agent. A value of 'N/A' means no timing is available

# of SQL Requests represents an 'accurate' count of the number of SQL data manipulation language (DML) statements issued by the

application thread. The value displayed for '`# of SQL Requests`' is in contrast to the 'wrap around' half word counter appearing in the output of the -DISPLAY THREAD command (whose maximum value is 32767). Anytime the -DISPLAY THREAD counter exceeds this value, it 'wraps around' or resets to 0. In contrast, the '`# of SQL Requests`' value is accurate beyond 2 billion

Status presents a 1 or 2 character code which describes the status of the connection. These values correspond to the status codes reported by the -DISPLAY THREAD command and convey the same meanings. The particular status codes (in alphabetic order) are as follows:

D	the thread is in the process of termination
DA	The database access thread slot is currently not associated with a remote connection and is available to be assigned to a type 2 inactive thread.
DI	The thread is disconnected from an execution unit. There is no TCB associated with the DB2 thread. This state is only valid when 'connection name'=RRSAF.
N	the thread is in either IDENTIFY or SIGNON status.
ND	The thread is in either IDENTIFY or SIGNON status, and the thread is currently not associated with any TCB.
PT	A parallel task thread was established (plan allocated).
QD	the thread is queued for termination
QT	the CREATE THREAD request was queued
R2	A distributed thread is performing a remote access on behalf of a request from another location. The thread is currently an inactive connection (type 2 inactive thread) and is waiting for an agent to become available to process.
RA	denotes remote access for a distributed thread
RK	A distributed thread is performing remote access on behalf of a request from another location. The thread is performing an operation that invoked Kerberos services. This status is displayed until Kerberos services returns control to DB2.
RN	denotes a remote distributed thread that is suspended
RQ	a distributed thread is remotely queued

RX	The distributed thread is executing an XA transaction on behalf of a request from another location.
	SA an active stored procedure
SP	A thread is executing within a stored procedure. This status is displayed until the stored procedure terminates and returns control to DB2.
	SW denotes a stored procedure waiting to be scheduled
T	an allied, non-distributed thread is established and a plan allocated
TD	An allied thread was established (plan allocated), and the thread is currently not associated with any TCB.
	TN an allied, distributed thread is suspended
TR	an allied thread was distributed to access data at another location

Where	denotes where the thread is currently executing.
PGM	indicates the thread is currently executing within application code
DB2	means the thread is currently active within DB2
COM	indicates the thread is executing or suspended within the communications network (TCP/IP and/or SNA/VTAM).
IDL	denotes an inactive database access thread.

Thread Summary - Selection Commands

The following row selection commands are supported:

S selects the thread for detailed display
U unselects the row

2.5 Thread Detail Display

The thread detail display shown in Figure 2.12 describes the DB2 thread. The Thread command field accepts a single character which corresponds to one of the thread commands described below: A "/" or "?" may also be entered in the command field to prompt for the following commands:

Blank updates the statistics displayed for the current thread. "S" is the default command. As a precaution, the 'S' command is ***reinstated*** each time the display is refreshed.

C is used to CANCEL the current thread.

D abends the MVS task associated with the current thread and requests that a dump be written to a SYSUDUMP dataset allocated to the jobstep of the abended task.

F can be used to FORCE the termination of threads that are not eligible for conventional cancellation. These include database access threads within a DDF address space and threads which are being deleted by DB2.

The FORCE command can also be used to remove threads which do not respond to standard cancellation commands. FORCE should be used judiciously since it can corrupt data integrity.

With respect to CICS transactions, the FORCE command abends the DB2 thread subtask, issues a CEMT SET TASK(taskid) PURGE against the CICS transaction and removes the VTAM or TCP/IP connection associated with the CICS transaction.

I is used to cancel the ISPF logical screen associated with the thread. If the target thread is not executing within a TSO/ISPF environment, then the "I" command is converted internally to a Cancel command.

K is used to forcibly purge a CICS transaction via a CEMT SET TASK(taskid) FORCEPURGE command

P is used to purge a CICS transaction via a CEMT SET TASK(tasked) PURGE command

Q is used to QUIESCE the MVS address space associated with the thread. The thread address space effectively becomes non-dispatchable with the lowest possible performance characteristics.

R is used to RESUME the MVS address space associated with the thread. The thread address space resumes execution with its original (pre-quiesced) performance characteristics.

S updates the statistics displayed for the current thread. "S" is the default command. As a precaution, the 'S' command is ***reinstated*** each time the display is refreshed.

V is used to terminate the VTAM or TCP/IP session(s) that underlie a distributed thread. The “V” command does not apply to local or utility threads.

Dialog users can enter a "/" or “?” in the Thread Command field to display the following prompting popup window.

```
 ____________________________________________________
|                                                    |
| Thread command ... S                               |
|                                                    |
| blank - Update thread statistics                   |
|     C - Cancel thread                              |
|     D - Cancel with Dump                           |
|     F - Force (abend) thread                       |
|     I - Remove ISPF logical screen                 |
|     K - Kill (Force Purge) CICS transaction        |
|     P - Purge CICS transaction                     |
|     Q - Quiesce thread's address space             |
|     R - Resume  thread's address space             |
|     S - Update thread statistics                   |
|     V - Remove thread's VTAM or TCP/IP connection  |
|____________________________________________________|
```

```
 ----------------------- Thread/STOPPER: Thread Detail -----------------------
 Command ===>
 TTS050 - Press ENTER to update stats, enter CMD letter or ? or hit PF3 to exit
    Thread command  ===> S               (Stats,Cancel,Force, / for prompts)
                                                              More:     +
DB2 Thread Identification
    DB2 subsystem   ===> DB2E            Connection Type ===> CALL ATTACH
    Correlation ID  ===> RAI007          Job name        ===> RAI007
    DB2 plan name   ===> RLX614E         ASID            ===> 004D
    Auth ID         ===> RAI007          Connection name ===> DB2CALL
    Status/Literal  ===> T               Local (non-distributed) thread
    ACE address     ===> 079BCA18        Thread token    ===> 949
    Current SQL ID  ===> RAI007          Original OperID ===> RAI007

 Logical Unit of Work ID
    Network ID      ===> USAMSI0A        LU name         ===> DB2APP3
    Thread number   ===> B194B98845FE    Commit Count    ===> 40
    Create mmddyyyy ===> 10/30/xxxx      Create hh:mm:ss ===> 15:06:54

 Thread Activity (press ENTER to update statistics)
    Where executing ===> PGM             Package         ===> RLXSQL
    SQL statements  ===> 1728            Getpage count   ===> 16005
    Total elapsed   ===> 24:45:23.109669 Elapsed in DB2  ===> 00:00:13.787186
    TCB (PGM & DB2) ===> 00:00:04.307939 TCB (DB2 only)  ===> 00:00:04.067669
    I/O wait time   ===> 00:00:00.436961 Lock wait time  ===> 00:00:00.012831

 Product under which the SQL application is executing
    Product name    ===> DSN             Product Version ===> 05
    Product release ===> 01              Prod mod level  ===> 0

 SQL applications running at a DB2 subsystem
    Location name   ===> RADB2E           SNA NetID       ===> USAMSI0A
```

```
    LU name         ===> DB2APP3         Connection name ===> DB2CALL
    Connection type ===> BATCH           Correlation ID  ===> RAI007
    Auth ID         ===> RAI007          Plan name       ===> RLX614E

Distributed thread with 0002 hops (The last connection hop is described)
    Connection type ===> SNA/VTAM
    Remote location ===> RADSN8           SNA/VTAM location
    Connection ID   ===> D3F7F45288512713 VTAM session ID

Client/Server Thread Information
   Client platform ===> Linux
   Client Appl     ===> httpd
   Client AuthID   ===> rai016

 Thread's Accounting String
    Char 001 - 050  ===> MVS Accounting String
    Char 051 - 100  ===>
    Char 101 - 150  ===>
    Char 151 - 200  ===>
```

Figure 2.12 DB2 Thread Detail

The Thread Detail display shown in Figure 2.12 describes the DB2 thread. The fields under the Thread Activity heading are updated each time you press ENTER.

Some of the values appearing under the heading ***Thread Identification*** are:

Job Name denotes the name of the job associated with the allied address space from which the thread originated.

Status/Literal presents the same 1 or 2 character code from the Thread Summary which describes the status of the connection. In addition, the literal which follows it provides a more meaningful description of the thread's status.

Thread token is the one to five digit decimal number that DB2 assigns to each thread. Threads connected to DB2 subsystems at a release level ***prior*** to DB2 Version 4.1 have no thread token assigned. Their thread token value displays as 0.

The set of prompts under the heading ***Logical Unit of Work*** serve to uniquely identify a thread within the network.

The set of values appearing under the heading ***Thread Activity*** are recalculated each time you press ENTER. When the following literals appear within a field, they denote the following:

N/P value is not present
N/C value cannot be calculated
N/A value is not applicable

Where Executing denotes *where* the thread is currently executing.

PGM indicates the thread is currently executing within application program code

DB2 means the thread is currently active within DB2

COM means the thread is currently active within the communications network (TCP/IP and/or SNA/VTAM).

IDLE denotes an inactive DDF thread.

Package Name identifies the name of the DB2 package that is currently executing within the plan. If a DBRM rather than a package is currently active, then the prompt appears as follows:

```
DBRM Name       ===> RLXSQL
```

Note that a CLASS 7 trace must be active in order for a Package or DBRM Name to be available for display. (See the discussion of Accounting Class Traces in Pre-installation Planning Section 20.2.1). Otherwise the following prompt appears to indicate the current package or DBRM is either not present or cannot be determined.

```
Current Program ===> N/P
```

SQL statements represents an 'accurate' count of the number of SQL data manipulation language (DML) statements issued by the application thread.

Getpage count denotes the number of `GET PAGE` requests issued on behalf of the thread. This includes conditional and unconditional requests as well as successful and unsuccessful requests. This value is an excellent measure of thread activity in that the getpage count is reliably updated by DB2 during thread execution.

Total Elapsed represents the elapsed 'wall clock' time since the thread was created.

Elapsed in DB2 represents the total elapsed time spent by the thread within DB2.

TCB (PGM & DB2) represents the amount of TCB CPU time expended so far by the thread, within both the application and DB2. For database access agents this value represents TCB time for the agent. A value of 'N/P' means no timing is available.

TCB (DB2 only) represents accumulated home CPU time while in DB2. A value of 'N/P' means no timing is available.

I/O wait time represents the amount of time the thread spent waiting to perform I/O.

Lock wait time represents the amount of time the thread spent waiting to acquire locks.

The set of prompts under the heading ***Product under which the SQL application is executing*** serve to uniquely identify the product that generated the thread's accounting string. Values are displayed for all threads for which MVS and DDF accounting information is present.

Product identifies the product that generated the accounting string. The product identifier may assume one the following values:

- DSN denotes DB2 for OS/390 or DB2 for MVS/ESA
- ARI denotes SQL/DS or DB2 for VM
- SQL denotes DB2 client/server
- QSQ denotes DB2/400

Product Version identifies the version of the product that generated the accounting string.

Product Release identifies the release level of the product that generated the accounting string.

Prod mod level identifies the modification level of the product that generated the accounting string.

The set of prompts under the heading ***SQL applications running at a DB2 subsystem*** refer to MVS and DDF accounting information for those threads whose Product Name value is 'DSN'. That is, threads whose accounting strings are generated by either DB2 for OS/390 or DB2 for MVS/ESA. When the Product Name value is other than 'DSN', these fields are blanks.

Location name identifies the DB2 location name for the DB2 system that created the accounting string.

SNA NetID identifies the SNA NETID of the DB2 system that created the accounting string.

LU name identifies the SNA LU name of the DB2 system that created the accounting string.

Connection Name identifies the DB2 Connection Name at the DB2 system where the SQL application is running.

Connection Type identifies the DB2 Connection Type at the DB2 system where the SQL application is running.

Correlation ID identifies the DB2 Correlation ID at the DB2 system where the SQL application is running.

Auth ID identifies the DB2 authorization ID that the SQL application used, prior to name translation and prior to driving the connection exit at the DB2 site where the SQL application is running.

Plan name identifies the DB2 PLAN that the SQL application used at the DB2 site running the SQL application.

The set of prompts under the ***Distributed thread*** headings refer to distributed allied and data base access threads. The last location or conversation associated with the thread is displayed. In the case of a non-distributed thread, these value display as N/A or Not Applicable.

Connection Type identifies the type of network connection used for this communication 'hop'. Either TCP/IP or SNA/VTAM.

Remote location identifies the TCP/IP address or SNA/VTAM location associated with the remote site.

Connection ID uniquely identifies the connection. For SNA/VTAM connections, this is the session ID. For TCP/IP connections, a 16 bit local port and 16 bit remote port are displayed.

The set of prompts under the heading ***Client/Server Thread Information*** refers to DDF accounting information for those threads whose Product Name value is 'SQL'. That is, threads whose accounting strings are created by DB2 client server products such as DB2 for Windows NT, DB2 for OS/2 and DB2 for various Unix implementations. When the Product Name value is other than 'SQL', these fields are blanks.

Client Platform identifies the Client Platform where the SQL application is running. This is a blank padded value.

Client Appl identifies the name of the Client Application. This is a blank padded value.

Client AuthID identifies the authorization ID of the client application process.

The set of prompts under the heading ***Thread's Accounting String*** apply to *all* threads for which accounting strings are available. The accounting string value may be up to 200 bytes in length. The accounting string is displayed in its entirety on up to four lines of 50 characters each.

2.5.1 Scrolling the Thread Detail Display

The Thread Detail Panel can be ***scrolled*** when more lines of information are present than can be displayed at one time. The Thread Detail Panel contains a scrollable area when the **More:** symbol appears below the Thread Command line (as illustrated below in underlined type).

```
---------------------- Thread/STOPPER: Thread Detail ----------------------
Command ===>
TTS050 - Press ENTER to update stats, enter CMD letter or ? or hit PF3 to exit
   Thread command  ===> S
                                                                  More:    -+
```

The **More:** scroll indicators denote the following:

- More: + You can scroll forward.
- More: - You can scroll backward.
- More: - + You can scroll both forward and backward.

2.6 Canceling Local Application Threads

This section describes and illustrates how to use the Thread/STOPPER Dialog to cancel a local DB2 application thread. First, key an 'S' into the thread's row selection field (as highlighted and underlined in Figure 2.11) to display the Thread Detail panel shown in Figure 2.12.

The Thread Command prompt is initialized to 'S' and appears in underlined type in Fig. 2.11. Just press ENTER to update activity statistics for the Thread. Alternatively, key a 'C' into this field and press ENTER in order to cancel the thread, enter 'D' to cancel the thread with a dump or enter 'F' to force cancellation of the thread. You can also explicitly request cancellation of the TCP/IP or SNA/VTAM session(s) that underlie the conversation(s) between distributed systems through the 'V' command or request cancellation of the ISPF Logical Screen associated with a thread through the 'I' command. You can also QUIESCE the MVS address space associated with the thread through the Q command or RESUME the address space through the R command. Lastly, you can press PF3 to bypass thread cancellation and return immediately to the Thread Summary display.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 5 of 5
 Command ===>                                                 Scroll ===> PAGE
 TTS053 - (S)elect a thread or press ENTER to refresh summary display

   Date: xx/10/25   Time: 16:08       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2B RAI6        RLXPLANC RAI6     DB2CALL  00:54:04 0:00:002      118 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:26:36 0:00:000        7 TR:PGM
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:28:05 0:00:001       14 T :PGM
 S DB2A RAIC        EXAMPLE  RAIC     DB2CALL  00:00:01 0:00:016       22 T :DB2
 . DB2B RAIC        EXAMPLE  RAIC     DB2CALL  00:00:00 0:00:000       28 T :DB2
```

Figure 2.13 DB2 Thread Summary

```
-------------------------- Thread/STOPPER: Thread Detail --------------------------
 Command ===>
 TTS050 - Press ENTER to update statistics, enter CMD letter or hit PF3 to exit
    Thread command  ===> S               (Stats,Cancel,Dump,Force,VTAM,ISPF)
                                         (Quiesce,Resume)
 DB2 Thread Identification
    DB2 subsystem   ===> DB2B            Connection Type ===> CALL ATTACH
    Correlation ID  ===> RAI1            Job name        ===> RAI1
    DB2 plan name   ===> EXAMPLE         ASID ID         ===> 006A
    Auth ID         ===> RAI1            Connection name ===> DB2CALL
    Status/Literal  ===> T               Local (non-distributed) thread
    ACE address     ===> 047F55C8        Thread token    ===> 130

Logical Unit of Work ID
    Network ID      ===> ALIC            LU name         ===> APPLDB2B
    Thread number   ===> AAF9DD7D40F5    Commit Count    ===> 1
    Create mm/dd/yy ===> 04/19/xxxx      Create hh:mm:ss ===> 19:43:55

 Thread Activity (press ENTER to update statistics)
    Where executing ===> PGM             Package Name    ===> RLXSQL
    SQL statements  ===> 403             Getpage count   ===> 6827
    Total elapsed   ===> 02:57:44.082255 Elapsed in DB2  ===> 00:00:12.805160
    TCB (PGM & DB2) ===> N/P             TCB (DB2 only)  ===> 00:00:06.048142
    I/O wait time   ===> 00:00:00.000000 Lock wait time  ===> 00:00:00.003168
```

Figure 2.14 Local DB2 Thread Detail

When you request that the thread be canceled (with or without a dump), Thread/STOPPER displays a pop-up window while thread cancellation in is progress. Thread/STOPPER typically waits an installation defined interval (the default is 5 seconds). If the thread still exists at the end of this interval, then the Thread Detail panel is redisplayed with the following message:

```
TTS173 - Thread still exists.  Press ENTER to check status or try another CMD
```

You can recheck the status of the thread every few seconds by pressing ENTER. Once Thread/STOPPER determines the thread is successfully canceled, it redisplays the Thread Summary panel as illustrated in Figure 2.15. The row associated with the canceled thread is flagged with a 'T' in its row selection field to denote a terminated thread. Further information regarding the canceled thread appears in the message window at the bottom of the display. This information (in addition to other thread and cancellation details) is recorded in the THREAD_AUDIT table described in Appendix A.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 5 of 5
 Command ===>                                                 Scroll ===> PAGE
 TTS053 - (S)elect a thread or press ENTER to refresh summary display

   Date: xx/10/25   Time: 16:08       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2B RAI6        RLXPLANC RAI6     DB2CALL  00:54:04 0:00:002      118 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:26:36 0:00:000        7 TR:PGM
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:28:05 0:00:001       14 T :PGM
 T DB2A RAIC        EXAMPLE  RAIC     DB2CALL  00:00:01 0:00:016       22 T :DB2
 . DB2B RAIC        EXAMPLE  RAIC     DB2CALL  00:00:00 0:00:000       28 T :DB2



.-------------------------------------------------------------------------.
| TTS171 - Cancel successful for the following DB2 thread.  Plan: EXAMPLE, DB2 |
| Auth ID: RAIC, Job: RAIC, Address space: 0035 and Unique Value          |
| AE38F536E89A.  The thread task was abended with system code 222 and abend  |
| reason code 00DB2000.                                                   |
'-------------------------------------------------------------------------'
```

Figure 2.15 Thread Summary reporting a canceled local DB2 thread

2.7 Canceling Distributed Application Threads

Canceling *distributed* DB2 application threads with Thread/STOPPER works much the same way as for local threads. Once again, key an 'S' into the thread's row selection field (as highlighted and underlined in Figure 2.16) to display the Thread Detail panel shown in Figure 2.17.

The same dialog flow and thread cancellation commands used in the previous section to cancel local threads apply to distributed threads.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 8 of 8
 Command ===>                                                 Scroll ===> PAGE
 TTS053 - (S)elect a thread or press ENTER to refresh summary display

   Date: xx/10/25   Time: 19:38       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where
 . DB2A RAI2        RLXEDIT  RAI2     DB2CALL  03:00:08 0:00:001      124 RA:DB2
 S DB2B RAI2        RLXEDIT  RAI2     DB2CALL  03:00:20 0:00:002      135 TR:PGM
 . DB2B RAIA        RLXEDIT2 RAIA     DB2CALL  02:22:38 0:00:002       56 T :PGM
 . DB2A RAI1        RLXPLANC RAI1     DB2CALL  00:01:09 0:00:000        8 RA:DB2
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:34:19 0:00:001       90 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:01:22 0:00:000        7 TR:PGM
 . DB2A RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
 . DB2B RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
```

Figure 2.16 DB2 Thread Summary

```
------------------------ Thread/STOPPER: Thread Detail -----------------------
 Command ===>
 TTS050 - Press ENTER to update statistics, enter CMD letter or hit PF3 to exit
    Thread command  ===> S              (Stats,Cancel,Dump,Force,VTAM,ISPF)
                                        (Quiesce,Resume)
 DB2 Thread Identification
    DB2 subsystem   ===> DB2B           Connection Type ===> CALL ATTACH
    Correlation ID  ===> RAI1           Job name        ===> RAI1
    DB2 plan name   ===> RLXPLANC       ASID ID         ===> 006A
    Auth ID         ===> RAI1           Connection name ===> DB2CALL
    Status/Literal  ===> TR             Distributed allied thread
    ACE address     ===> 047F55C8       Thread token    ===> 130

 Logical Unit of Work ID
    Network ID      ===> ALIC           LU name         ===> APPLDB2B
    Thread number   ===> AAF9DC0BFF3C    Commit Count    ===> 1
    Create mm/dd/yy ===> 10/25/xxxx     Create hh:mm:ss ===> 19:37:26

 Thread Activity (press ENTER to update statistics)
    Where executing ===> PGM            Package Name    ===> RLXSQL
    SQL statements  ===> 403            Getpage count   ===> 6827
    Total elapsed   ===> 02:57:44.082255 Elapsed in DB2  ===> 00:00:12.805160
    TCB (PGM & DB2) ===> N/P            TCB (DB2 only)  ===> 00:00:06.048142
    I/O wait time   ===> 00:00:00.000000 Lock wait time  ===> 00:00:00.003168
```

Figure 2.17 DB2 Thread Detail

When you request that a distributed thread be canceled (with or without a dump), Thread/STOPPER displays a pop-up window while thread cancellation in is progress. Thread/ STOPPER typically waits an installation defined interval (the default is 5 seconds). If the thread still exists at the end of the interval, then the Thread Detail panel is redisplayed with the following message.

```
TTS173 - Thread still exists.  Press ENTER to check status or try another CMD
```

You can recheck the status of the distributed thread every few seconds by pressing ENTER. If the thread remains active, you can reissue the ***C***ancel command or terminate the TCP/IP and/or VTAM sessions that underlie the distributed thread via the ***V*** command.

When Thread/STOPPER determines the thread is canceled, it redisplays the Thread Summary panel as illustrated in Figure 2.18. The row associated with the canceled thread is flagged with a 'T' in its row selection field to denote a terminated thread. Further information regarding the canceled thread appears in the message window at the bottom of the display. This information (in addition to other thread and cancellation details) is recorded in the THREAD_AUDIT table described in Appendix A.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 8 of 8
 Command ===>                                                 Scroll ===> PAGE

   Date: xx/10/25   Time: 19:40       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2A RAI2        RLXEDIT  RAI2     DB2CALL  03:00:08 0:00:001      124 RA:DB2
 . DB2B RAI2        RLXEDIT  RAI2     DB2CALL  03:00:20 0:00:002      135 TR:PGM
 . DB2B RAIA        RLXEDIT2 RAIA     DB2CALL  02:22:38 0:00:002       56 T :PGM
 . DB2A_RAI1        RLXPLANC RAI1     DB2CALL  00:01:09 0:00:000        8 RA:DB2
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:34:19 0:00:001       90 T :PGM
 T DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:03:06 0:00:001        7 TR:PGM
 . DB2A RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
 . DB2B RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2

.-------------------------------------------------------------------------.
| TTS181 - The distributed threads associated with the Logical Unit of Work ID |
| (LUWID) ALIC.APPLDSX.AAF9DC0BFF3C were terminated by issuing -CANCEL DDF     |
| THREAD.                                                                      |
.-------------------------------------------------------------------------.
```

Figure 2.18 Thread Summary reporting a canceled set of distributed DB2 threads

Thread/STOPPER cancels all threads with the specified LUWID using DB2, MVS, TCP/IP and/or VTAM facilities, as appropriate. If the distributed thread is active within DB2, Thread/STOPPER issues a -CANCEL DDF THREAD command. As such, the authorization ID of the user of the Thread/STOPPER dialog (i.e. the user who issues the -CANCEL DDF THREAD request) should have one of the following DB2 system privileges:

- SYSOPR authority
- SYSCTRL authority
- SYSADM authority

Use the 'V' command to terminate the TCP/IP and/or VTAM sessions that underlie the distributed thread.

2.8 Canceling IBM DB2 Utility Threads

This section describes and illustrates how to cancel an IBM DB2 utility with Thread/ STOPPER. This discussion pertains only to DB2 utilities supplied by IBM.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 8 of 8
 Command ===>                                                 Scroll ===> PAGE
 TTS053 - (S)elect a thread or press ENTER to refresh summary display

   Date: xx/10/25   Time: 19:40       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2A RAI2        RLXEDIT  RAI2     DB2CALL  03:00:08 0:00:001      124 RA:DB2
 . DB2B RAI2        RLXEDIT  RAI2     DB2CALL  03:00:20 0:00:002      135 TR:PGM
 . DB2B RAIA        RLXEDIT2 RAIA     DB2CALL  02:22:38 0:00:002       56 T :PGM
 S DB2B RAI1RS      DSNUTIL  RAIC     UTILITY  00:00:00 0:00:000        0 T :DB2
 . DB2A RAI1        RLXPLANC RAI1     DB2CALL  00:01:09 0:00:000        8 RA:DB2
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:34:19 0:00:001       90 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:03:06 0:00:001        7 TR:PGM
 . DB2A RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
 . DB2B RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
```

Figure 2.19 DB2 Thread Summary including an IBM DB2 Utility

```
------------------------ Thread/STOPPER: Thread Detail -----------------------
 Command ===>
 TTS050 - Press ENTER to update statistics, enter CMD letter or hit PF3 to exit
    Thread command  ===> S               (Stats,Cancel,Dump,Force,VTAM,ISPF)
                                         (Quiesce,Resume)
DB2 Thread Identification
    DB2 subsystem   ===> DB2B            Connection Type ===> DB2 UTILITY
    Correlation ID  ===> RAI1RS          Job name        ===> RAI1RS
    DB2 plan name   ===> DSNUTIL         ASID ID         ===> 0046
    Auth ID         ===> RAIC            Connection name ===> UTILITY
    Status/Literal  ===> T               Local (non-distributed) thread
    ACE address     ===> 047F55C8        Thread token    ===> 130

 Logical Unit of Work ID
    Network ID      ===> ALIC            LU name         ===> APPLDB2B
    Thread number   ===> AAF9DD0B954B    Commit Count    ===> 1
    Create mm/dd/yy ===> 10/19/xxxx      Create hh:mm:ss ===> 19:41:56

Thread Activity (press ENTER to update statistics)
    Where executing ===> DB2             Package Name    ===> DSNUTIL
    SQL statements  ===> 403             Getpage count   ===> 6827
    Total elapsed   ===> 02:57:44.082255 Elapsed in DB2  ===> 00:00:12.805160
    TCB (PGM & DB2) ===> N/P             TCB (DB2 only)  ===> 00:00:06.048142
    I/O wait time   ===> 00:00:00.000000 Lock wait time  ===> 00:00:00.003168
```

Figure 2.20 Detail panel for a DB2 Utility Thread

Canceling DB2 utility threads with Thread/STOPPER proceeds in much the same way as for DB2 application threads. Once again, key an 'S' into the thread's row selection field (as highlighted and underlined in Figure 2.19) to display the Thread Detail panel illustrated in Figure 2.20.

The Cancel Thread prompt appears in underlined type in Figure 2.19. Key a 'C' into this field and press ENTER in order to cancel the DB2 utility. Alternatively, press PF3 to bypass cancellation and return immediately to the Thread Summary display.

If you elect to cancel the DB2 utility jobstep, Thread/STOPPER displays the Utility Termination panel illustrated in Figure 2.21. This panel prompts you to press ENTER to confirm the -TERM utility request or to press the END key to return immediately to the Thread Summary panel.

> ***NOTE:*** *There may be times when there is more than one utility thread active with the same DB2 authorization ID. Section 2.8.1 describes how to deal with such circumstances.*

```
-------------------- Thread/STOPPER: Confirm -TERM UTILITY --------------------
Command ===>
TTSØ82 - Confirm DB2 utility termination or enter END command to bypass
DB2 Utility Identification
   MVS system ===> PRD1
   DB2 SSID   ===> DSX
   Utility ID ===> RAI5.RAI1RS
   User       ===> RAI5
   Statement  ===> 77
   Utility    ===> RUNSTATS
   Phase      ===> RUNSTATS
   Count      ===> 9
   Status     ===> ACTIVE

Instructions:
   Press ENTER key to confirm the -TERM UTILITY request
         or
   Enter END command to bypass DB2 utility termination
```

Figure 2.21 IBM DB2 Utility Detail

Once the DB2 utility thread is canceled, the Thread Summary panel is redisplayed with the utility thread's row selection field flagged with a 'T' to denote terminated. Further information regarding the canceled utility thread appears in the message window at the bottom of the display.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ------ Row 1 to 8 of 8
 Command ===>                                                  Scroll ===> PAGE

   Date: xx/10/25   Time: 19:40       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2A RAI2        RLXEDIT  RAI2     DB2CALL  03:00:08 0:00:001      124 RA:DB2
 . DB2B RAI2        RLXEDIT  RAI2     DB2CALL  03:00:20 0:00:002      135 TR:PGM
 . DB2B RAIA        RLXEDIT2 RAIA     DB2CALL  02:22:38 0:00:002       56 T :PGM
 T DB2B RAI1RS      DSNUTIL  RAIC     UTILITY  00:00:00 0:00:000        0 T :DB2
 . DB2A RAI1        RLXPLANC RAI1     DB2CALL  00:01:09 0:00:000        8 RA:DB2
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:34:19 0:00:001       90 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:03:06 0:00:001        7 TR:PGM
 . DB2A RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
 . DB2B RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2

.-------------------------------------------------------------------------.
| TTS076 - IBM DB2 utility RUNSTATS, ID RAIC.RAI1RS, User RAIC was         |
| successfully terminated at a cleanup point                               |
.-------------------------------------------------------------------------.
```

Figure 2.22 Thread Summary reporting a canceled DB2 Utility thread

2.8.1 More than one Utility Thread with the same AUTH ID

This section describes and illustrates how to cancel an IBM DB2 utility via the Thread/ STOPPER Dialog when there is more than one utility thread with the same Auth ID. As always, the process starts by keying an ‘S’ into the thread’s row selection field -- as illustrated in Figure 2.23.

The Cancel Thread prompt appears in underlined type in Figure 2.24. Key a ‘C’ into this field and press ENTER in order to cancel the DB2 utility. Alternatively, press PF3 to bypass cancellation and return immediately to the Thread/Summary display.

```
---------------------- Thread/STOPPER: DB2 Thread Summary ---- Row 1 to 8 of 8
 Command ===>                                                 Scroll ===> PAGE

   Date: xx/10/25   Time: 19:40       MVS system: PRD1

   DB2  Correlation DB2 Plan Auth ID  Connect  Elapsed  TCB time # of SQL Status
                                      Name     HH:MM:SS H:MM:SST Requests :Where

 . DB2A RAI2        RLXEDIT  RAI2     DB2CALL  03:00:08 0:00:001      124 RA:DB2
 . DB2B RAI2        RLXEDIT  RAI2     DB2CALL  03:00:20 0:00:002      135 TR:PGM
 . DB2B RAIA        RLXEDIT2 RAIA     DB2CALL  02:22:38 0:00:002       56 T :PGM
 S DB2B RAI1RS      DSNUTIL  RAIC     UTILITY  00:00:00 0:00:000        0 T :DB2
 . DB2A RAI1        RLXPLANC RAI1     DB2CALL  00:01:09 0:00:000        8 RA:DB2
 . DB2B RAI5        RLXPLANC RAI5     DB2CALL  00:34:19 0:00:001       90 T :PGM
 . DB2B RAI1        RLXPLANC RAI1     DB2CALL  00:03:06 0:00:001        7 TR:PGM
 . DB2A RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
 . DB2B RAIC        TTSPLAN  RAIC     DB2CALL  00:00:00 0:00:000        0 T :DB2
```

Figure 2.23 DB2 Thread Detail

```
------------------------ Thread/STOPPER: Thread Detail ------------------------
 Command ===>
 TTS050 - Press ENTER to update statistics, enter CMD letter or hit PF3 to exit
    Thread command  ===> S               (Stats,Cancel,Dump,Force,VTAM,ISPF)
                                         (Quiesce,Resume)
DB2 Thread Identification
    DB2 subsystem   ===> DB2B            Connection Type ===> DB2 UTILITY
    Correlation ID  ===> RAI1RS          Job name        ===> RAI1RS
    DB2 plan name   ===> DSNUTIL         ASID ID         ===> 0046
    Auth ID         ===> RAIC            Connection name ===> UTILITY
    Status/Literal  ===> T               Local (non-distributed) thread
    ACE address     ===> 047F55C8        Thread token    ===> 130

 Logical Unit of Work ID
    Network ID      ===> ALIC            LU name         ===> APPLDB2B
    Thread number   ===> AAF9DD0B954B    Commit Count    ===> 1
    Create mm/dd/yy ===> 10/19/xxxx      Create hh:mm:ss ===> 19:41:56

 Thread Activity (press ENTER to update statistics)
    Where executing ===> DB2             Package Name    ===> DSNUTIL
    SQL statements  ===> 403             Getpage count   ===> 6827
    Total elapsed   ===> 02:57:44.082255 Elapsed in DB2  ===> 00:00:12.805160
    TCB (PGM & DB2) ===> N/P             TCB (DB2 only)  ===> 00:00:06.048142
    I/O wait time   ===> 00:00:00.000000 Lock wait time  ===> 00:00:00.003168
```

Figure 2.24 Detail panel for a DB2 Utility Thread

When there are multiple DB2 utilities with the same Auth ID, Thread/STOPPER displays the Utility Summary panel illustrated in Figure 2.25. Key an 'S' into the row selection field of the DB2 utility jobstep you wish to cancel.

```
------------------- Thread/STOPPER: IBM DB2 Utility Summary ------- Row 1 of 1
Command ===>                                                  Scroll ===> PAGE

Local MVS System Information
   Date / Time    ===> xx/10/19   19:54
   MVS system     ===> PRD1

   DB2   Utility ID    User      Statement Utility   Phase     Count  Status
   DB2B  RAI5.UTIL1    RAI5             37 RUNSTATS  RUNSTATS      9  ACTIVE
   DB2B  RAI5.UTIL2    RAI5             77 RUNSTATS  RUNSTATS      9  ACTIVE
 S DB2B  RAI5.UTIL3    RAI5             67 RUNSTATS  RUNSTATS      9  ACTIVE
   DB2B  RAI5.UTIL4    RAI5             45 RUNSTATS  RUNSTATS      9  ACTIVE

.-------------------------------------------------------------------------.
|TTS083 - Select which DB2 utility you wish to terminate by keying an "S" in |
|its row selection field                                                  |
.-------------------------------------------------------------------------.
```

Figure 2.25 Summary of DB2 Utility threads with the same AUTH ID

Once you identify a DB2 utility for cancellation, Thread/STOPPER displays the Utility Termination panel illustrated in Figure 2.26. You are prompted to either press ENTER to confirm the -TERM utility request, or to press the END key to bypass termination and return to the Thread Summary immediately.

```
-------------------- Thread/STOPPER: Confirm -TERM UTILITY --------------------
Command ===>
TTS082 - Confirm DB2 utility termination or enter END command to bypass
DB2 Utility Identification
   MVS system ===> PRD1
   DB2 SSID   ===> DB2B
   Utility ID ===> RAI5.UTIL3
   User       ===> RAI5
   Statement  ===> 77
   Utility    ===> RUNSTATS
   Phase      ===> RUNSTATS
   Count      ===> 9
   Status     ===> ACTIVE

Instructions:
   Press ENTER key to confirm the -TERM UTILITY request
         or
   Enter END command to bypass DB2 utility termination
```

Figure 2.26 IBM DB2 Utility Detail

Chapter 3

Batch and Console Commands

This chapter describes the ***commands*** you can issue through the Thread/STOPPER Batch and Console facilities. It also describes the ***qualifiers*** used to specify the thread(s) upon which a command will act. The syntax of all Thread/STOPPER commands is the command name followed by one or more keyword(value) operands:

```
command keyword1(value1) keyword2(value2) ...
```

For example, consider the following command string:

```
DISPLAY SSID(DB2T) PLAN(PLANØ1)
```

In this example, the ***command*** is DISPLAY while SSID and PLAN are ***qualifiers***. The SSID qualifier identifies the DB2 subsystem in which a thread exists while the PLAN qualifier specifies the plan name associated with a set of thread(s).

This chapter enumerates the commands and qualifiers common to both the Thread/STOPPER Batch and Console Facilities. The next chapter (Chapter 4) describes and illustrates how to specify commands to the Thread/STOPPER Batch Facility. (It reads them from a TTSIN file you define in your batch jobstep). Chapter 5 provides annotated examples in which the Thread/STOPPER Console Facility reads and executes commands entered from an MVS console.

3.1 Thread/STOPPER commands

You direct Thread/STOPPER to perform services -- like displaying and canceling threads -- through ***commands***. This section describes each Thread/STOPPER command in turn, in alphabetical order.

> ***NOTE:*** *All Thread/STOPPER commands can be abbreviated to their first 3 characters. For example, the CANCEL command can be abbreviated as CAN while the DISPLAY command can be abbreviated as DIS.*

3.1.1 CANCEL command

The CANCEL command causes Thread/STOPPER to terminate the DB2 thread(s) that satisfy the qualification criteria you specify. For example, the following command string cancels a thread on DB2 subsystem DB2T whose plan name is PLAN01.

```
CANCEL SSID(DB2T) PLAN(PLANØ1)
```

As a precaution, Thread/STOPPER rejects any CANCEL command which would cancel more than a single thread. The only exception is when the Thread/STOPPER Batch Facility is invoked with CANCEL_MULTIPLE as an execution parameter (as described in Chapter 4).

> ***NOTE:*** *Be careful to use the CANCEL_MULTIPLE option judiciously since you may inadvertently cancel many (even all) active DB2 threads.*

FORCE keyword of CANCEL command

The FORCE keyword of CANCEL can be used after a conventional thread cancellation request fails. FORCE bypasses a number of checks the CANCEL command normally makes to determine whether a thread is eligible to be canceled. (See Section 3.3). The FORCE option of CANCEL should be used judiciously.

```
CANCEL SSID(DB2T) PLAN(PLANØ1) FORCE
```

TCPIP keyword of the CANCEL command

The TCPIP keyword of the CANCEL command requests that a TCPIP '**DROP connection**' command be issued to terminate the TCPIP connection that underlies a distributed thread. The TCPIP option is useful when a distributed thread is currently suspended in TCPIP or a previous CANCEL command directed towards a distributed thread has failed. For example, the following command string terminates the TCPIP session that underlies the distributed thread whose plan name is PLAN01.

```
CANCEL SSID(DB2T) PLAN(PLANØ1) TCPIP
```

VTAM keyword of the CANCEL command

The VTAM keyword of the CANCEL command requests that the VTAM command **'V NET,TERM'** be issued to terminate the VTAM session(s) that underlie a distributed thread. The VTAM option is useful when a distributed thread is currently suspended in VTAM or a previous CANCEL command failed. For example, the following command string terminates the VTAM session that underlies the distributed thread whose plan name is PLAN01.

```
CANCEL SSID(DB2T) PLAN(PLAN01) VTAM
```

3.1.1.1 Canceling inactive DB2 threads

Database Access Threads (DBAT) that are in inactive status are *not* tracked (nor reported) by the DB2 IFI interface. As such, inactive Database Access Threads require special action to cancel them. A DBAT thread is identified by the thread token or LUWID associated with it. You can display inactive threads by issuing a DB2
-DISPLAY THREAD command which specifies `TYPE(INACTIVE)`:

```
-DISPLAY THREAD(*)TYPE(INACTIVE)
```

The -DISPLAY THREAD command displays a report similar to the following:

```
DSNV401I  -DB2T DISPLAY THREAD REPORT FOLLOWS -
DSNV424I  -DB2T INACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   R2       0 ICMClient.ex CM@JCH   DISTSERV 0044 55730
 V437-WORKSTATION=S157267, USERID=cm@jch,
      APPLICATION NAME=ICMClient.exe
 V445-GA000020.K893.070522150308=55730 ACCESSING DATA FOR 10.0.0.32
 V447--LOCATION         SESSID           A ST TIME
 V448--10.0.0.32        446:37704        W R2 0714210080927
```

You can try to cancel the DBAT thread (whose token is 55730 in the example), via the following DB2 command:

```
-CANCEL DDF THREAD(55730)
```

DB2 indicates its acceptance of the CANCEL request. However, the thread will remain (with inactive status). Unless and until subsequent workstation action (re)activates the thread, DB2 does not terminate the thread nor remove it.

However, the immediate removal of an inactive thread may occasionally be required, such as when a client application is inaccessible. For these cases, the Thread/STOPPER Batch and Console facilities have implemented the following form of the CANCEL command:

```
CANCEL SSID(ssid) TYPE(INACTIVE | INA) { LUWID(token | luwid ) }
```

The TYPE(INACTIVE) keyword specifies the CANCEL should operate on inactive threads. If you omit the LUWID parameter, then all inactive threads will be canceled, otherwise only the thread with the given LUWID (or thread token) will be canceled.

Example 1: Cancel all inactive threads on DB2 subsystem DB2T

```
CANCEL SSID(DB2T) TYPE(INA)
```

Example 2: Cancel the inactive thread on DB2 subsystem DB2T whose thread token is 55730:

```
CANCEL SSID(DB2T) TYPE(INA) LUWID(55730)
```

Example 3: Cancel the inactive thread on DB2 subsystem DB2T whose concatenated LUWID value is GA000020.K893.070522150308:

```
CANCEL SSID(DB2T) TYPE(INA) LUWID(GA000020.K893.070522150308)
```

When the inactive thread is canceled successfully, the TCP/IP session associated with the inactive thread is also terminated.

Example 4: Issue a CANCEL command via the Thread/CONSOLE facility:

```
@xx TTSE042 - Thread/STOPPER is waiting for work
R xx,CAN SSID(DB2T) TYPE(INA) LUWID(55730)
IEE600I REPLY TO xx IS; CAN SSID(DB2T) TYPE(INA) LUWID(55730)
+
```

DB2 issues a DSNL511I message to indicate the TCP/IP session underlying a DB2 thread has terminated:

```
DSNL511I  -DB2T DSNLIENO TCP/IP CONVERSATION FAILED 714
            TO LOCATION 10.0.0.32
            IPADDR=10.0.0.32 PORT=37704
            SOCKET=RECV RETURN CODE=3448 REASON CODE=00000000
+TTSE069 - Inactive thread with LUWID= was
GA000020.K893.070522150308
 canceled
```

NOTE: In order to successfully execute CANCEL TYPE(INACTIVE) commands, the pre-installation requirements described in Section 20.2.6 of this publication must be satisfied. Section 20.2.6 describes the requirements to cancel inactive threads in detail.

3.1.2 DB2S command

The DB2S command displays all DB2 subsystems defined to the local MVS system. If a DB2 subsystem is active then the DB2S command also displays its Version and Release level. The DB2S command requires no operands. For example:

```
DB2S
```

3.1.3 DISPLAY command

The DISPLAY command causes Thread/STOPPER to display the DB2 thread(s) that meet your qualification criteria. For example, the following command string displays all threads on DB2 subsystems DB2T and DB2P whose plan name is PLAN01.

```
DISPLAY SSID(DB2T,DB2P) PLAN(PLANØ1)
```

3.1.3.1 Displaying inactive threads

For a description of DB2 inactive threads and how they can be canceled, see section 3.1.1.1.

To display DB2 inactive threads via the Thread/STOPPER Batch and Console facilities, use the following command:

```
DISPLAY SSID(ssid) TYPE( INACTIVE | INA)
       { LUWID(thread_token | luwid) }
```

The display is the same as produced by the following DB2 command:

```
-DIS THREAD(*) TYPE(INA)  { LUWID(thread_token | luwid) }
```

appended to the Thread/STOPPER DISPLAY command for an active thread.

3.1.4 DSN command

The DSN command lets you execute the same DB2 command on multiple DB2 systems simultaneously. For example, the following command string displays all in-doubt threads on DB2 subsystems DB2T and DB2P:

```
DSN SSID(DB2T,DB2P) -DISPLAY THREAD(INDOUBT)
```

3.1.5 XLOCK command

The XLOCK command displays all DB2 threads which hold ***exclusive*** locks. The XLOCK command is useful in identifying threads that may cause contention. For example, the following command string displays all exclusive locks held by threads of DB2 subsystem DB2T:

```
XLOCK SSID(DB2T)
```

NOTE: *The XLOCK command requires no qualifiers beyond SSID (DB2 subsystem ID)*

3.1.6 LOCK command

The LOCK command displays information about the DB2 locks held by the threads that meet your qualification criteria. For example, the following command string displays the locks held by threads on subsystem DB2T with a plan name of PLAN01:

```
LOCK SSID(DB2T) PLAN(PLAN01)
```

3.1.7 STATUS command

The STATUS command displays the current settings of Thread/STOPPER execution parameters. The STATUS command requires no operands so you simply issue the command as follows:

```
STATUS
```

3.1.8 STOP command

Issue the STOP command to cause the Thread/STOPPER Console Facility to terminate. The STOP command requires no operands so you simply issue the command as follows:

```
STOP
```

3.2 Specifying Thread Qualification Criteria

Use Thread/STOPPER qualification keywords to specify the thread(s) upon which your commands will act. The syntax for all Thread/STOPPER qualifications is the qualifier keyword followed by the qualifying value within parentheses as in: `keyword(value)`

Thread/STOPPER supports the use of standard SQL wildcard characters in qualifiers. The underscore character '_' is a placeholder which matches any ***one*** character while the percent sign character '%' matches ***any*** number of characters.

For example, PLAN(PLAN%) selects all threads whose plan names start with the characters 'PLAN'. The qualifier CORR(USER_0T%) selects all threads whose correlation IDs start with the string 'USER', followed by any single character, followed by the string '0T', followed by any other characters.

If you wish the qualification to be treated exactly as specified, do not use wildcard characters. For example if you specify AUTH(TECHS1) then only authorization ID TECHS1 will be selected. In contrast, the authorization ID TECHS1A will not be selected.

This section describes each Thread/STOPPER qualifier keyword in turn, in alphabetical order.

> ***NOTE:*** *The SSID keyword (which identifies the DB2 subsystem) is always required since it has no default value. Nor may the SSID value be partially qualified.*

3.2.1 ACE qualifier

The ACE qualifier identifies the Agent Control Element associated with a DB2 thread. The ACE value is an address comprised of 8 hexadecimal digits. Together with the SSID qualifier, the ACE uniquely identifies a DB2 thread. The Thread/STOPPER DISPLAY command provides the ACE address of each thread it monitors. You can reference these ACE addresses in CANCEL commands you subsequently issue.

```
Example:  CANCEL SSID(DB2T) ACE(Ø8704808)
```

***NOTE**: Wildcard characters are not permitted in ACE specification.*

3.2.2 ASID qualifier

The ASID qualifier provides the Address Space Identifier of a thread's allied address space. ASID values are specified as 4 hexadecimal digits.

```
Example:  CANCEL SSID(DB2T) ASID(Ø12A)
```

***NOTE**: Wildcard characters are not permitted in ASID specification.*

3.2.3 AUTH qualifier

The AUTH keyword identifies the primary authorization ID of the desired thread(s). AUTH may be up to 8 characters long.

```
Example:  DISPLAY SSID(DB2T) AUTH(DB2USER)
```

***NOTE**: AUTH qualifier can be specified using '%' and '_' pattern characters.*

3.2.4 CONN qualifier

The Connection Name qualifier identifies the 8 character DB2 connection type. Possible values include TSO, BATCH, DB2CALL and UTILITY.

Threads originating in CICS and IMS address spaces may have additional connection names -- IMS-ID (for IMS threads) and CICS-ID (for CICS threads) while . DB2 internal threads have a CONN value of the DB2 SSID. For distributed database access threads that employ the DB2 private protocol and for threads using the DRDA protocol, CONN should contain the connection name of the thread at the requesting location. For threads using the DRDA protocol from a non-DB2 requester, CONN should contain the constant 'SERVER'.

The following example displays all threads on DB2T connected through the Call Attachment Facility:

```
Example:  DISPLAY SSID(DB2T) CONN(DB2CALL)
```

***NOTE**: CONN qualifier can be specified using '%' and '_' pattern characters.*

3.2.5 CORR qualifier

The CORR keyword denotes a field of up to 12 characters in length whose significant characters identify the Correlation ID of the desired thread(s). The following example displays all threads on DB2T whose correlation IDs start with the characters 'RAI'.

```
Example:  DISPLAY SSID(DB2T) CORR(RAI%)
```

***NOTE**: CORR qualifier can be specified using '%' and '_' pattern characters.*

3.2.6 LOCN qualifier

The Location Name identifies the name of the location from which distributed agents originate. Location name may be up to 16 characters long.

```
Example:  DISPLAY SSID(DB2T) LOCN(LOCATIONDB2T)
```

***NOTE**: LOCN qualifier can be specified using '%' and '_' pattern characters.*

3.2.7 LUWI qualifier

The LUWI keyword identifies a thread by its logical-unit-of-work ID. A full LUWI value is comprised of four concatenated components: a Network ID, a LU name, a uniqueness value and a Commit count. It is very difficult to correctly enter a LUWI as a single 64-byte hexadecimal value. Instead, RAI recommends that you specify the value of each component of the LUWI *individually* as described in subsections 3.2.7.1 through 3.2.7.4.

***NOTE:** If the value of a LUWI component is not specified, then a value of binary zeros is assumed. Such a value is treated as a generic (wildcard) selector for any value.*

3.2.7.1 The NETID qualifier

The NETID qualifier is the first component of the logical-unit-of-work ID. It specifies the SNA Network ID associated with a thread. NETID can be between 1 and 8 characters.

```
Example: CANCEL SSID(DB2T) NETID(IBMNET)
```

3.2.7.2 The LUNAME qualifier

The LUNAME qualifier is the second component of the logical-unit-of-work ID. It specifies an SNA Logical Unit Name between 1 and 8 characters long.

```
Example: CANCEL SSID(DB2T) LUNAME(LUDB2T)
```

3.2.7.3 The UNIQ qualifier

The UNIQ qualifier is the third component of the logical-unit-of-work ID. It specifies the Unique Value associated with a particular thread and Logical-Unit-of-Work. UNIQ qualifiers are exactly 12 hexadecimal digits long.

```
Example:  DISPLAY SSID(DB2T) UNIQ(AB034568A188)
```

3.2.7.4 The CCNT qualifier

The fourth component of the logical-unit-of-work ID specifies a Commit Count value. Commit count values are between 1 and 4 hexadecimal digits long.

```
Example:  DISPLAY SSID(DB2T) CCNT(1)
```

3.2.8 OPID qualifier

The OPID qualifier specifies the original authorization ID associated with a set of thread(s). OPID values are between 1 and 8 characters long.

```
Example:  DISPLAY SSID(DB2T) OPID(SIGNON)
```

***NOTE**: OPID qualifier can be specified using '%' and '_' pattern characters.*

3.2.9 PLAN qualifier

This keyword specifies the DB2 application plan which is used as a thread selection qualifier.

```
Example:  DISPLAY SSID(DB2T) PLAN(PLAN01)
```

The Plan qualifier denotes an 8 byte character field whose significant characters identify the name of the DB2 application plan associated with a set of threads.

***NOTE**: PLAN qualifier can be specified using '%' and '_' pattern characters.*

3.2.10 SSID qualifier

The SSID qualifier is required with all Thread/STOPPER commands. It identifies the name of the DB2 Subsystem in which a set of thread(s) exist as well as where a Thread/STOPPER command is to execute. DB2 subsystem names may be up to 4 characters long.

```
Example:  DISPLAY SSID(DB2T,DB2P)
```

3.3 LOCKS keyword and its qualifiers

Use the LOCKS keyword and its associated qualifiers to specify the DB2 resources of concern, and by inference the thread(s) upon which a Thread/STOPPER DISPLAY or CANCEL command should act.

This section first describes the LOCKS operand and then its associated qualifiers in turn, in alphabetical order.

3.3.1 LOCKS (S | IS | X | IX) keyword

The LOCKS keyword can be used with the DISPLAY and CANCEL commands. It identifies the threads that hold one of the following locks:

S - Thread owns a resource in shared mode
IS - Thread intends to share a resource
X - Thread owns a resource in exclusive mode
IX - Thread intends to own a resource exclusively

If the lock type parameter is omitted then ***all*** lock types will be selected.

For example, the following command string displays all exclusive locks held by threads onthe DB2 subsystems DB2T and DB2P.

```
Example:  DISPLAY SSID(DB2T,DB2P) LOCKS(X)
```

NOTE: *TheLOCKS keyword requires specification of one of the qualifiers: DBNAME, or DBNAME and TSNAME, or TBNAME.*

3.3.2 DBNAME qualifier

The DBNAME qualifier identifies all threads holding locks on a DB2 database.

```
Example:  CANCEL SSID(DB2T) DBNAME(DSNDBØ4) LOCKS(X)
```

This command directs Thread/STOPPER to cancel all threads which hold exclusive locks on the database DSNDB04.

3.3.3 TBNAME qualifier

The TBNAME qualifier identifies the DB2 table name and is specified as ***owner.table***.

```
Example:  CANCEL SSID(DB2T) TBNAME(PAYROLL.EMPLOYEE_TABLE) LOCKS
```

This command directs Thread/STOPPER to cancel all threads which hold any lock on the table named PAYROLL.EMPLOYEE_TABLE

3.3.4 TSNAME qualifier

The TSNAME qualifier must always be specified together with DBNAME qualifier. It identifies all threads holding locks on the specified DB2 table space. The TSNAME specifies 1 to 8 characters name of a DB2 table space.

```
Example:  CANCEL SSID(DB2P) DBNAME(PRODDB) TSNAME(EMPLOYTS) LOCKS
```

This command directs Thread/STOPPER to cancel all threads holding locks on the table space named EMPLOYTS within the DB2 database named PRODDB.

3.4 Thread cancellation strategy

Thread/STOPPER Batch and Console facilities implement the following CANCEL strategy based upon the DB2 thread type:

- If a DB2 Utility thread, then issue the *–TERM UTIL* DB2 command.
- If the FORCE keyword is specified, then issue an abend in the allied address space under the thread's Task Control Block (TCB). This cancellation mode should only be used as a last resort when all other cancellation methods have failed and you wish to forcibly remove the thread.
- If the TCPIP keyword is specified, then a *VARY TCPIP DROP* command is issued to terminate the TCP/IP connection that underlies this DDF thread. This command should be issued only after both –CANCEL DDF and –CANCEL THREAD commands are issued (as described below). Issue this command when the thread is 'hung' in TCP/IP.
- If the VTAM keyword is specified then a *V NET,TERM,SESS* command is issued to terminate the parallel VTAM session that underlies this DDF thread. This command

should be issued only after both the -CANCEL DDF and –CANCEL THREAD commands are issued (as described below). Issue this command when the thread is 'hung' in VTAM.

- If the thread status is at the plan level (as opposed to having SIGNON or IDENTIFY status) and the plan name is DISTSERV, then issue a *-CANCEL DDF* command.

- If the thread attachment is CICS, then identify the task ID associated with the thread and issue the CICS command: *CEMT SET TASK (taskid) FORCE*

3.5 Data Sharing Group (DSG) operation

Thread/STOPPER determines if a DB2 subsystem is a member of a Data Sharing Group (DSG). If so, then DISPLAY and CANCEL commands are executed on each of the active members of the Data Sharing Group of which the local DB2 subsystem is a member. This means that in a DSG environment, a single copy of Thread/STOPPER can manage all the threads of the Data Sharing Group.

The SSID keyword must specify the name of a DB2 subsystem active on the MVS image in which Thread/STOPPER is executing. All Thread/STOPPER commands are available for all members of DB2 Data Sharing Group with the following exceptions:

- The FORCE, VTAM, TCPIP and CICS keywords only apply to members of the Data Sharing Group that reside on the local MVS image

- CANCEL notifications are sent only to TSO User IDs active on the local MVS image

IBM has steadily improved the DB2 CANCEL command with each new version of DB2 for z/OS. As such, the need for special cancellation strategies beyond –CANCEL THREAD and –CANCEL DDF are greatly reduced.

Chapter 4

The Thread/STOPPER Batch Facility

The Thread/STOPPER Batch Facility allows you to specify a single Thread/STOPPER command or set of commands within the context of a batch jobstep. This chapter describes and illustrates how to make requests of the Batch Facility using the commands and qualifiers common to both the Thread/STOPPER Batch and Console Facilities. (These common commands and qualifiers were described in Chapter 3).

This chapter also documents the initialization parameters to invoke the Batch and Console Facility programs. These parameters govern operation of both programs during a particular run. Several parameters unique to the Thread/STOPPER Console Facility are documented separately in the discussion of the Console program that appears in Chapter 5.

4.1 Preparing JCL for the Thread/STOPPER Batch Facility

This first section describes the JCL needed to run the Thread/STOPPER Batch Facility as a step of a batch jobstream. Figure 4.1 illustrates the JCL distributed as member TTSRB of the TTSCNTL library. The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow the figure.

```
//jobname  JOB ......          (1)
//TTSBATCH EXEC TTSPROC,       (2)
//         PROG=TTSB           (3)
//TTSIN    DD  *               (4)
CANCEL SSID(ssid) PLAN(someplan)
//
```

Figure 4.1 JCL to run the Thread/STOPPER Batch Facility

(1) Define a valid job statement.

(2) Invoke the Thread/SERIES catalogued procedure named TTSPROC which defines common JCL used to run several Thread/SERIES programs. This procedure is distributed as member TTSPROC of the TTSCNTL library and is typically copied into one of your site's catalogued procedure libraries during Thread/SERIES installation. Alternatively, TTSPROC can be used as an instream procedure. Lastly, the TTSCNTL dataset which contains TTSPROC can be defined as a JCL procedure library (on a job basis) through a JCLLIB statement or some functionally equivalent dynamic PROCLIB facility.

(3) Run the Thread/STOPPER Batch Facility program named TTSB.

(4) Thread/STOPPER reads your Batch Facility commands from the file named TTSIN. Figure 4.1 illustrates how these commands may be specified instream with your jobstep following the TTSIN DD statement. Alternatively, the TTSIN DD statement can reference a dataset which contains Thread/STOPPER commands.

> ***NOTE:*** *You can specify several Batch Facility commands in a single TTSIN input stream*

4.2 Thread/STOPPER Execution Parameters

This section describes the execution-time parameters that govern the operation of the Thread/STOPPER Batch and Console Facility programs. These parameters must be specified in the startup JCL that invokes the Thread/STOPPER Batch or Console Facilities. The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow the figure.

```
//jobname  JOB
//TTSBRUN  EXEC TTSPROC,                                    (1)
//         PROG=TTSB,                                       (2)
//         PARMS='keyword1(value1) keyword2(value2)'        (3)
```

Figure 4.2 Specifying execution time parameters

(1) Invoke the general Thread/SERIES procedure

(2) Run the Thread/STOPPER Batch Facility program named TTSB.

(3) Execution-time parameters are specified through the PARMS operand with which the TTSPROC procedure is invoked. The syntax of these execution-time parameters are of the following general form:

keyword1(value1) keyword2(value2) ...

Each keyword value must be specified precisely. That is, no wildcard characters such as '_' or '%' are permitted.

This section describes each Thread/STOPPER execution-time parameter in turn, in alphabetical order.

4.2.1 ACS keyword

The ACS keyword identifies the name of the member of the TTSCNTL library which contains JCL to invoke the Thread/STOPPER Audit Facility. The Audit Facility ensures that complete details about the cancellation of a particular thread are recorded in the Thread_Audit table, even when cancellation takes an indefinite period of time to complete successfully. You need specify this operand ***only*** if the member containing this JCL has been renamed from its default value of TTSRACS.

Example:

```
//TTSBRUN  EXEC TTSPROC,
//         PROG=TTSB,
//         PARMS='ACS(TTSRACS)'
```

4.2.2 AUDIT keyword

The AUDIT keyword specifies -- (YES) or (NO) -- whether to invoke the Thread/STOPPER Audit Facility. A THREAD_AUDIT table must have been created at installation time in order to use the Audit Facility.

Example:
```
//          PARMS='AUDIT(NO)'
```

4.2.3 CANCEL_MULTIPLE keyword (Thread/STOPPER Batch Facility only)

The CANCEL_MULTIPLE parameter specifies -- (YES) or (NO) -- whether you can cancel more than one thread with a single command in the event that more than one thread meets your qualification criteria. The default is NO. Use this keyword sparingly since you may inadvertently cancel many (even all) DB2 threads.

Example:
```
//          PARMS= 'CANCEL_MULTIPLE(YES)'
```

4.2.4 CAN_DUMP keyword

The CAN_DUMP keyword specifies -- (YES) or (NO) -- whether Thread/STOPPER should obtain a DUMP of a canceled thread's allied thread address space. The default is NO.

Example:
```
//          PARMS=' CAN_DUMP(YES)'
```

4.2.5 CAN_RETRY keyword

The CAN_RETRY keyword specifies -- (YES) or (NO) -- whether to allow retry processing in the MVS task associated with a canceled thread. CAN_RETRY applies when the task associated with a thread issues the ESTAE macro and specifies a recovery and retry routine. The default is YES. In this case, the MVS task will be permitted to pass control to its retry routine and continue processing.

Example:
```
//          PARMS=' CAN_RETRY(YES)'
```

4.2.6 CAN_STEP keyword

The CAN_STEP keyword specifies -- (YES) or (NO) -- whether the jobstep task associated with a canceled thread should be abnormally terminated. The default is NO. This allows a TSO user for example to remain in session even if one of their DB2 thread tasks is canceled.

Example:
```
//          PARMS=' CAN_STEP(YES)'
```

4.2.7 The LIMIT keyword

The LIMIT keyword defines the maximum number of output lines the DISPLAY command can produce. The default is 9999 lines.

Example:
```
//         PARMS=‘ LIMIT(50)’
```

4.3 Batch Facility Examples

This section provides several annotated examples of Thread/STOPPER Batch Facility jobstreams. To remind, the syntax of all Thread/STOPPER commands is the command name followed by one or more qualifier keyword(value) operands:

```
command qualifier1(value1) qualifier2(value2)
```

Example 1: Display all active threads on subsystems DB2T and DB2P whose plan names start with the string ‘ABC’.

```
//jobname   JOB
//TTSBATCH  EXEC TTSPROC,
//          PROG=TTSB
//TTSIN     DD  *
  DISPLAY SSID(DB2T,DB2P) PLAN(ABC%)
//
```

Example 2: Cancel all threads on DB2 subsystem DB2T with a plan name of ABC. Ensure that Thread/STOPPER honors the CANCEL command if more than one thread meets this qualification criteria by specifying CANCEL_MULTIPLE(YES) as a parameter with which the Batch Facility is invoked.

```
//jobname   JOB
//TTSBATCH  EXEC TTSPROC,
//          PROG=TTSB,
//          PARMS=‘CANCEL_MULTIPLE’
//TTSIN     DD  *
  CANCEL SSID(DB2A) PLAN(ABC)
//
```

Chapter 5

The Thread/STOPPER Console Facility

This chapter describes how Thread/STOPPER commands may be entered from an MVS console. This chapter first describes how to invoke the Thread/STOPPER Console Facility and continues with a discussion of the invocation parameters unique to the Console Facility. Lastly, a series of annotated examples are presented to illustrate an operator's interaction with the Thread/STOPPER Console Facility.

5.1 Invocation Parameters unique to the Console Facility

Section 4.2 described the common execution-time parameters that govern the operation of Thread/STOPPER's Batch and Console Facilities. These parameters must be specified in the startup JCL used to invoke the Thread/STOPPER Batch and Console Facilities. This section describes the invocation parameters unique to the Console Facility. These unique parameters are described in alphabetical order in the subsections which follow.

5.1.1 CANCEL_MULTIPLE parameter

The Thread/STOPPER Console Facility ***ignores*** any CANCEL_MULTIPLE parameter specification you code. In the context of the Thread/STOPPER *Batch Facility*, the CANCEL_MULTIPLE parameter specifies whether you can cancel more than one thread with a single command (in the event that more than one thread meets your qualification criteria). In contrast, the *Console Facility* permits you to cancel only one thread at a time.

5.1.2 DESCRCDE parameter

The DESCRCDE keyword defines the MVS Console Message Descriptor Code(s) to be assigned to any WTO messages issued by the Console Facility. You can specify a list of one or more of the descriptor codes shown in Figure 5.1. The DESCRCDE is optional and has no default value.

```
Example:

//TTSCON   EXEC TTSPROC,    Invoke the Thread/SERIES procedure
//         PROG=TTSCON,     Run the Thread/STOPPER Console program
//         PARMS=' DESCRCDE(1,2,3)'
```

Desc Code	Description	Desc Code	Description
1	System failure	9	Operator request
2	Immediate action required	10	Dynamic status display
3	Eventual action required	11	Critical eventual action
4	System status	12	Reserved
5	Immediate command response	13	Reserved
6	Job status	15	Reserved
7	Application program	14	Reserved
8	Out-of-line message	16	Reserved

Figure 5.1 MVS Descriptor Codes

5.1.3 ORIGIN keyword

The ORIGIN keyword specifies how operators and authorized users will communicate with the Console Facility. ORIGIN refers to where commands originate.

WTOR

Coding ORIGIN(WTOR) specifies that operator communication with the Console Facility is accomplished by responding to an outstanding WTOR reply number. For example, when activated or idle, the Console Facility waits on a reply to a command prompt like the following:

```
@xx Thread/STOPPER is ready and waiting for work
```

where xx is any number from 0 to 99 assigned by MVS. Thread/STOPPER commands are issued by replying to the outstanding message number. For example:

```
R xx,DISPLAY SSID(DB2T) PLAN(ABC%)
```

QEDIT

Coding ORIGIN(QEDIT) specifies that operator communication with the Console Facility is accomplished through MVS MODIFY commands. Suppose, for example, the Console Facility is submitted as a job named TTSCON. Then, to issue a Thread/STOPPER command, you need to enter an MVS MODIFY command -- such as the following -- from the MVS Console:

```
MODIFY TTSCON, DISPLAY SSID(DB2T) PLAN(ABC%)
```

NOTE: *In contrast to the WTOR option, there is no message number xx displayed through the MODIFY interface.*

5.1.4 ROUTECDE keyword

The ROUTECDE keyword defines the MVS Console Message Routing Codes to be assigned to WTO and WTOR messages issued by the Console Facility. The default is ROUTECDE(11). You can specify a list of one or more of the routing codes shown in Figure 5.2:

```
Example:   ROUTECDE(1,2,3)
```

Route Code	Description	Route Code	Description
1	Master console action	9	System security
2	Master console information	10	System error/maintenance
3	Tape pool	11	Programmer information
4	Direct access pool	12	Emulators
5	Tape library	13	User defined
6	Disk library	15	User defined
7	Unit record pool	14	User defined
8	Teleprocessing control	16	User defined

Figure 5.2 MVS Routing Codes

5.2 Invoking the Console Facility

The Thread/STOPPER Console Facility can be executed as a job or started task. Section 20.10.2 describes how the JCL to invoke the Console Facility is set up at installation time while Section 20.10.3 describes DB2 Authorization ID considerations for started tasks. Thereafter, you can invoke the Thread/STOPPER Console Facility as a started task by executing the following MVS console command:

```
S TTSRCON.TTS
```

where TTSRCON specifies the name of the started procedure for the Thread/STOPPER Console Facility. Alternatively, you can invoke the Console Facility as a standard job by submitting JCL such as:

```
//job      JOB
//TTSCON   EXEC TTSPROC,    Invoke the Thread/SERIES procedure
//         PROG=TTSCON ,    Run the Thread/STOPPER Console program
//         PARMS='ORIGIN(WTOR)'     Specify runtime parameters
//
```

5.3 Console Facility Command Examples

This section provides a series of annotated examples in which Thread/STOPPER commands are issued to the Console Facility. In each example, the command is issued as a console reply to WTOR number 88.

> ***NOTE:*** *Thread/STOPPER commands can also be issued through the MVS MODIFY interface if ORIGIN(QEDIT) is specified as an invocation parameter on the Console Facility startup procedure. See Section 5.1.3.*

5.3.1 CANCEL command example

This example assumes you first issued a Thread/STOPPER DISPLAY command to determine the ACE (DB2 Agent Control Element) value associated with the thread you want to cancel. Once this ACE address is obtained, you can issue the following CANCEL command:

```
@88 Thread/STOPPER is ready and waiting for work
R 88,CANCEL SSID(DB2T) ACE(Ø88B1158)

@CANCEL request for Thread SSID=DB2T,ACE=Ø88B1158,PLAN=EXAMPLE,AUTH=RAIØØ1 will be
terminated by asynchronous request
@Thread/STOPPER Audit Facility job submitted
@89 Thread/STOPPER is ready and waiting for work
```

This example cancels the application thread on DB2 subsystem DB2T (whose ACE address is 088B1158) via the Thread/STOPPER CANCEL command. After canceling the thread, the Console Facility submits an Audit Facility job to monitor the thread's status and log its successful cancellation in the THREAD_AUDIT table.

5.3.2 DB2S command example

The DB2S command displays all DB2 subsystems defined to the local MVS system. If a subsystem is active, then its DB2 Version and Release level are also displayed:

```
@25 Thread/STOPPER is ready and waiting for work
R 25,DB2S
SSID  Status    CmdPrefix   Procname  Release  Group
-------------------------------------------------
DB7A  Active    DB7A        DB7AMSTR  710
DB7B  Active    DB7B        DB7BMSTR  710
DSN8  Active    DSN8        DSN8MSTR  810
DB8A  Inactive  DB8A        DB8AMSTR  ---
DB8B  Inactive  DB8B        DB8BMSTR  ---
D9GA  Active    D9GA        D9GAMSTR  910      D9G
D9GB  Inactive  D9GB        D9GBMSTR  ---      D9G
-------------------------------------------------
```

5.3.3 DISPLAY command examples

The DISPLAY command writes information to the MVS Console about the thread(s) that meet the qualification criteria you specify. The output includes an ACE address (8 hexadecimal digits) which can be used to uniquely identify a thread on a subsequent CANCEL command. For example, the following DISPLAY command shows all threads whose authorization ID starts with the characters RAI:

```
@25 Thread/STOPPER is ready and waiting for work
R 25,DIS SSID(DB8B) AUTH(RAI%)
DB2  ACE      DB2 Plan Primary  Connect  Elapsed  TCB time # of SQL Status
SSID                   auth ID  name     HH:MM:SS H:MM:SST requests :Where
-------------------------------------------------------------------------
DB8B 0800B8F8 RLXvrmCS RAI024   DB2CALL  07:27:02 0:00:018      644 T :PGM
DB8B 0800B578 RLXvrmCS RAI025   DB2CALL  04:00:31 0:00:006      216 T :PGM
DB8B 0800BC78 TTSPvrm  RAI025   DB2CALL  00:00:00 0:00:000        0 T :DB2
-------------------------------------------------------------------------
```

The following DISPLAY command shows all threads that hold locks on database DSNDB04. Subsystem D7GA is a member of the Data Sharing group that includes subsystem D8GB.

```
R 25,DIS SSID(D7GA) DBNAME(DSNDB04) LOCKS
DB2  ACE      DB2 Plan Primary  Connect  Elapsed  TCB time # of SQL Status
SSID                   auth ID  name     HH:MM:SS H:MM:SST requests :Where
------------------------------------------------------------------------------
D9GA 084C03B8 RLXvrmCS RAI027   DB2CALL  00:02:41 0:00:000       13 T :PGM
D8GB 084C03B8 RLXvrmCS RAIB23   DB2CALL  00:02:41 0:00:000       13 T :PGM
D8GB 084C0578 RLXvrmCS RAIB23   DB2CALL  00:02:41 0:00:000       13 T :PGM
------------------------------------------------------------------------------
```

5.3.4 DSN command example

The Thread/STOPPER DSN command lets you execute any DB2 command on several DB2 subsystems simultaneously. In the following example, the DB2 -DISPLAY THREAD(*) command is issued on the DB2 subsystems named DB2T and DB2P.

```
@27 Thread/STOPPER is ready and waiting for work
R 27,DSN SSID(DB8B,D9GA) -DISPLAY THREAD(*)

DSNV401I DB8B DISPLAY THREAD REPORT FOLLOWS -
DSNV402I DB8B ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
DB2CALL  T  *     4 RAI2TTSB     RAI028   TTSPvrm  0019   937
DB2CALL  T      228 RAI025       RAI025   RLXvrmCS 007B   909
DB2CALL  T      835 RAI024       RAI024   RLXvrmCS 0086   884
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I DB7B DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
DSNV401I  -D9GA DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D9GA ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
DB2CALL  T  *     5 RAI2TTSB     RAI028   TTSPvrm  0019   548
DB2CALL  T       15 RAI023       RAI023   RLXvrmCS 007E   536
DB2CALL  T       15 RAI023       RAI023   RLXvrmCS 007E   537
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -D9GA DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
DSNV401I  -D9GB DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D9GB ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
DB2CALL  T       15 RAIB23       RAIB23   RLXvrmCS 0055   195
DB2CALL  T       15 RAIB23       RAIB23   RLXvrmCS 0055   196
```

5.3.5 XLOCK command example

The XLOCK command displays the exclusive locks held by application threads that meet your qualification criteria. The following example displays the resources held exclusively by threads on subsystem DB2T whose plan name is RLXvrmCS:

```
@32 TTSE042 - Thread/STOPPER is waiting for work
R 32,XLOCK SSID(D9GA) PLAN(RLXvrmCS)

XLOCK SSID(D9GA)
LOCKS HELD ON DB2 SUBSYSTEM: D9GA
Ace      Token    Resource        Stat Resource Name                  Duration
-------------------------------------------------------------------------------
076EFA88 083BE1AC PageSet         X    0107 0002                      COMMIT
-------------------------------------------------------------------------------
```

> ***NOTE**: The column labeled '**Resource Name**' shows two hexadecimal numbers. The first is the DBID and the second is PSID of the resource. Refer to the DB2 catalog table SYSIBM.SYSTABLESPACE to determine the name of the Table Space represented by these two values.*

5.3.6 LOCK command display

The LOCK command displays the locks held by the application threads that meet your qualification criteria.

```
@24 TTSE042 - Thread/STOPPER is waiting for work
R 24,LOCK SSID(D9GA) PLAN(RLXvrmCS)

LOCKS HELD ON DB2 SUBSYSTEM: D9GA
Ace      Token    Resource        Stat Resource Name                  Duration
-------------------------------------------------------------------------------
076F08C8 0814EE48 PageSet         IS   0004 0002                      COMMIT
076F08C8 0814EDC8 DataBase        S    0004 0000                      COMMIT
076F08C8 0814ED48 PageSet         IS   0006 0009                      COMMIT
076F08C8 0814DAC8 SkelPkgTable    S    RLXVRM.RLXSQL6.16E1542C0B8BCFF COMMIT
076F08C8 0814DB48 SkelCursorTable S    RLXvrmCS                       PLAN
076F08C8 0814EE48 PageSet         IS   0004 0002                      COMMIT
076F08C8 0814EDC8 DataBase        S    0004 0000                      COMMIT
076F08C8 0814ED48 PageSet         IS   0006 0009                      COMMIT
076F08C8 0814DAC8 SkelPkgTable    S    RLXVRM.RLXSQL6.16E1542C0B8BCFF COMMIT
076F08C8 0814DB48 SkelCursorTable S    RLXvrmCS                       PLAN
076EF1C8 0814EEC8 SkelCursorTable S    TTSPvrm                        PLAN
-------------------------------------------------------------------------------
```

In this example, the application thread whose plan name is RLXvrmCS is holding shared locks on Table Space with DBID 0004 and PSID 0002 (DSNDB04), a shared lock on Database with DBID 0004 (DSNDB04), etc.

You can determine the DB2 table name associated with a pair of DBID / OBID values by querying the catalog table SYSIBM.SYSTABLES. Use the decimal values of DBID and OBID as search criteria. The DB2 Diagnosis Guide and Reference (Publication LY27-9536) provides an explanation of the lock types and states that appear in LOCK command output, along with a discussion of DBID, OBID and PSID values.

5.3.7 STATUS command display

The STATUS command displays the current values of the Thread/STOPPER run-time parameters. These include run-time defaults as well as any explicit execution parameters specified when the Console Facility was invoked (as described in Section 5.2). The STATUS command also displays the values of Thread/STOPPER system parameters defined at installation time.

```
@88 Thread/STOPPER is ready and waiting for work
R 88,STATUS
RUN-TIME PARAMETERS
 ORIGIN............ WTOR
 CAN_STEP.......... NO
 CAN_TERM.......... YES
 CAN_DUMP.......... NO
 CAN_RETRY......... YES
 CAN_STEP.......... NO
 LIMIT............. 9999

 SYSTEM PARAMETERS
 TSDAC#............ 222
 TSDARC#........... 00DB2000
 TSDWAIT#.......... 0000005:
 TSDPLAN........... TTSPLAN
 TSDACD#........... 004E
 TSDARCD#.......... 00E50013
 TSDVAB#........... RAIAPL
 TSDOPW............ RAI
```

Chapter 7

Thread Audit View Facility

Thread/SERIES maintains an audit trail which describes DB2 threads and actions taken vis-a-vis those threads by various Thread/SERIES components (such as the Thread/ STOPPER dialog or the Thread/SENTRY automated monitor). The audit trail is maintained in a table named THREAD_AUDIT that resides within the same DB2 sub-system in which the audited thread was executing. Appendix A provides detailed descriptions of the columns which comprise the Thread_Audit table. Some columns of the Thread_Audit table provide statistics and details about the thread while other columns document who took action against a thread, when and for what reason.

This chapter describes how to use the Thread Audit View Facility to browse this audit trail. Section 7.1 describes how to invoke the Thread Audit View Facility via the TTSRUN command while Section 7.2 describes and illustrates the dialog panels presented by the View Facility.

7.1 Invoking the Thread Audit View Facility

Section 20.16 describes how to setup the Thread Audit View Facility so it can be invoked as an ISPF dialog. The REXX exec TTSRUN described in Section 20.16 manages all Thread/SERIES libraries dynamically *within* ISPF and invokes the Thread Audit View Facility as well as the Thread/STOPPER Dialog.

Alternatively, the Thread Audit View Facility may have been setup as an ISPF selection menu option (as described in Section 20.18.) In this case, see your Thread/SERIES product administrator for details concerning invocation of the Thread Audit View Facility.

7.2 Dialog flow

This section describes and illustrates the panels displayed by TTSVIEW. The dialog first presents a Query by Example panel with which you can identify the Thread Audit records you wish to view. The Query By Example (QBE) panel illustrated in Figure 7.1 allows you to select rows from the Thread_Audit table without the need to construct SQL SELECT statements. You can also specify an optional sort order and direction for each column.

```
Host/Edit --- Query By Example: TTSvrm.THREAD_AUDIT --------------------------
Command ===>

Use symbols % and _ to activate LIKE predicate logic. Use  > and < as a first
character in numeric column types.  ENTER to continue or END to exit dialog

Ord Dir Seq Column Name/Type             Column Value
... .    23 Plan Name..............(C8) --------
... .    24 PGM Name...............(C8) --------
... .    22 Correlation............(C12) ------------
... .    25 Auth ID................(C8) --------
... .    26 Connection.............(C8) --------
... .     7 Policy ID..............(C8) --------
... .     8 Policy Reason..........(C24) ------------------------
... .     2 Unique Value...........(C12) ------------
... .     3 Token..................(I4) --------------
... .     5 Action Taken...........(C12) ------------
... .    10 Action Date............(DT10) ----------
... .    11 Action Timestamp......(TS26) --------------------------
```

Figure 7.1 Query by Example panel presented by TTSVIEW dialog

The Column headings in Figure 7.1 have the following meaning:

Ord contains a decimal number (1,2,3...n) to specify an ordering of the rows of the query result. This information is used to build the SQL query's ORDER BY clause as illustrated in the example below.

Dir specifies sort direction: **A**scending or **D**escending. Sort direction for all columns is ASCending by default. Thus, ascending is assumed when you assign Ord for this column and no value is entered for Dir.

Column Value specifies a value of the column to be used to select table rows. You can use **%** and **_** characters to select table rows using generic patterns via the SQL LIKE predicate. Note that when the `_' (underscore) is used, the entire column value must be filled with `_'. For numerical column values you can use > and < operators in the first position of the column value to create a comparison predicate.

The columns of the Thread_Audit table for which you can provide search criteria are as follows:

Plan Name identifies the 1 to 8 character name of the DB2 application plan associated with the thread. The TTSVIEW dialog folds whatever characters you specify here to uppercase.

Program Name identifies the name of the DB2 package or DBRM that is currently executing within the plan. The TTSVIEW dialog folds whatever characters you specify for 'program name' to uppercase.

Correlation ID denotes a 1 to 12 character recovery 'correlation-id' associated with the thread. Both upper and lower case characters are significant for the Correlation ID pattern.

Authorization ID identifies the 8 character DB2 primary authorization associated with the thread. The TTSVIEW dialog folds whatever characters you specify here to uppercase.

Connection Name identifies the 8 character DB2 connection type. Possible values include BATCH, DB2CALL, RRSAF, SERVER, TSO, and UTILITY. Threads originating in CICS and IMS address spaces may have additional connection names. The TTSVIEW dialog folds whatever characters you specify for a connection name to uppercase.

Policy ID identifies the site defined policy whose violation led Thread/SENTRY to take an automatic action. Alternatively, enter the value 'MANUAL' to select those threads against which a Thread/SERIES action was initiated manually. Both upper and lower case characters are significant in the pattern specified for 'Policy ID'.

Policy Reason contains a character string which briefly describes what threshold was violated to trigger a policy defined action. Policy reason strings are comprised of a category prefix followed by the specific threshold that was violated. The currently defined set of policy categories and their corresponding prefixes include:

Life-of-thread thresholds	LOT
Unit-of-work thresholds	UOW
Interval based minimum thresholds	IMIN
Interval based maximum thresholds	IMAX
Inactive thread thresholds	IDLE

Unique Value specifies a 12 digit hexadecimal value that uniquely identifies a thread within a particular DB2 subsystem. The TTSVIEW dialog folds whatever alphabetic characters you specify in the Unique Value string to uppercase.

Thread Token specifies the shorthand integer value DB2 assigns to each thread. The token is one to five decimal digits. Note that threads connected to DB2 subsystems at releases *prior* to Version 4.1 have no thread token assigned. Instead, their thread token value appears as 0.

Action identifies the action (such as cancel or warning) taken against a thread. 'Action' identifies either the user requested action issued through one of the Thread/STOPPER facilities, or the policy defined action triggered automatically by Thread/SENTRY. The value you enter for Action is case sensitive so enter an exact Action value like 'Cancel', 'Warn' or 'Force' or an Action pattern like 'C%', 'W%' or 'F%'.

Action Date indicates the date when an audited action took place. Both '%' and '_' characters are valid in the pattern for Action Date.

Action Timestamp reflects the current timestamp when a row was inserted into the Thread_Audit table. Both '%' and '_' characters are valid in the pattern for Action Date.

Once you specify values for all search criteria, press ENTER to create and execute the SQL query. Alternatively, press the END key to exit the dialog. Figure 7.2 illustrates a sample summary display reflecting audited actions against threads with the plan name 'TTSIvrm'.

```
Col 1 of 181 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 6
Command ===>                                                 Scroll ===> HALF

-    23        24           22             25               26              7
Rcmd Plan Name Program Name Correlation ID Authorization ID Connection Name Pol
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           POL
'''' -         -            -              -                -               POL
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI020           BATCH           POL
'''' -         -            -              -                -               POL
```

Figure 7.2 Thread Audit table Tabular Display

7.3 ROW Commands

You can enter Row commands in the **Rcmd** field of the Summary Display panel as illustrated in Figure 7.3 in which the Rcmd field is underlined.

```
Col 1 of 181 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 6
Command ===>                                                  Scroll ===> HALF

-    23        24           22             25               26              7
Rcmd Plan Name Program Name Correlation ID Authorization ID Connection Name Pol
S''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           POL
'''' -         -            -              -                -               POL
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI020           BATCH           POL
'''' -         -            -              -                -               POL
```

Figure 7.3 Row Command entered in the Rcmd field

S (Select)

Use the 'S' row command to display the columns of a particular row on a detail panel like the one illustrated in Figure 7.4. Once a specific row is displayed, you can press the UP (PF7) and DOWN (PF8) keys to bring additional columns of the selected Thread_Audit row into view. Then, press the End key (typically PF3) to return to the summary display of query result rows.

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 6
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''  23 Plan Name (C8)............. N TTSIvrm
''  24 Program Name (C8)........... N TTSIVP
''  22 Correlation ID (C12)........ N RAI7IVPE
''  25 Authorization ID (C8)....... N RAI021
''  26 Connection Name (C8)........ N BATCH
''   7 Policy ID (C8).............. N MANUAL
''   8 Policy Reason (C24)......... N Manually selected thread
''   2 Unique Value (C12).......... N B1A02DEE43B0
''   3 Thread Token (I4)........... N  1402
''   5 Action (C12)................ N Cancel
''  10 Action Date (DT10).......... N 01/08/20xx
''  11 Action Timestamp (TS26)..... N 20xx-01-08-17.50.30.153421
''   1 SSID (C4)................... N DB2E
''   4 ACE Address (C8)............ N 079BCA18
''   6 Action Status (C12)......... N Pending
''   9 Action Method (C40)......... N DB2 CANCEL THREAD issued
''  12 Action ACEE (C8)............ N RAI021
''  13 Action SSID (C8)............ N RAI021
''  14 Action Grp (C8)............. N RAIGRP
''  15 Action Term (C8)............ N RAILU021
''  16 Action Appl (C8)............ N N/P
''  17 Action Surr (C8)............ N N/P
''  18 Action CPU (C12)............ N 282373882003
''  19 Creation Date (C10)......... N 01/08/20xx
```

```
''  2Ø Creation Time (C8)..... N 17:46:14
''  21 Creation Micro (C7)..... N 6690565
''  27 Operator (C8)..... N RAI021
''  28 SMF ID (C8)..... N RALP
''  29 ASID (C4)..... N ØØ38
''  3Ø Jobname (C8)..... N RAI7IVPE
''  31 Where (C4)..... N PGM
''  32 Accounting Token (C22)..... N N/P
''  33 SQL Count (S2)..... N     122
''  34 Status Code (C2)..... N T
''  35 Net ID (C8)..... N USAMSIØA
''  36 LU Name (C8)..... N DB2APP3
''  37 Commit # (I4)..... N        Ø
''  38 SQL DML # (I4)..... N      122
''  39 Getpages (I4)..... N         14
''  4Ø Status Literal (C4Ø)..... N Local (non-distributed) thread
''  41 Connection Code (I4)..... N              1
''  42 Connecting System (C12)..... N TSO
''  43 TCB Addr (C8)..... N ØØØØØØØØ
''  44 Elapsed Total (C15)..... N ØØ:Ø4:14.Ø55481
''  45 Elapsed DB2 (C15)..... N ØØ:ØØ:ØØ.261288
''  46 Class 1 Time (C15)..... N ØØ:ØØ:ØØ.Ø33104
''  47 Class 2 Time (C15)..... N ØØ:ØØ:ØØ.Ø23257
''  48 SRB Time (C15)..... N ØØ:ØØ:ØØ.ØØØ742
''  49 IO Wait Time (C15)..... N ØØ:ØØ:ØØ.ØØØØØØ
''  5Ø Lock Wait Time (C15)..... N ØØ:ØØ:ØØ.ØØØØØØ
''  51 Distributed Location (C16).. N N/P
''  52 APPC ID (C8)..... N N/P
''  53 Distributed Session (C16)... N N/P
''  65 QMDA Product (C3)..... N DSN
''  66 QMDA Version (C2)..... N Ø4
''  67 QMDA Release (C2)..... N Ø1
''  68 QMDA Mod Level (C1)..... N Ø
''  69 QMDA Location (C16)..... N RADB2E
''  7Ø QMDA Net ID (C8)..... N USAMSIØA
''  71 QMDA LU Name (C8)..... N DB2APP3
''  72 QMDA Connect Name (C8)..... N BATCH
''  73 QMDA Connect Type (C8)..... N BATCH
''  74 QMDA Correlation (C12)..... N RAI7IVPE
''  75 QMDA AuthID (C8)..... N RAI021
''  76 QMDA Plan (C8)..... N TTSIVRM
''  77 Client Platform (C18)..... N ..................
''  78 Client Application (C2Ø).... N .....................
''  79 Client AuthID (C8)..... N ........
''  8Ø Account String Length (S2).. N       9
''  81 Accounting String (V2ØØ).... N RAIØ 1234
             cont(ØØ2) ØØØ41-ØØØ8Ø
             cont(ØØ3) ØØØ81-ØØ12Ø
             cont(ØØ4) ØØ121-ØØ16Ø
             cont(ØØ5) ØØ161-ØØ2ØØ
```

Figure 7.4 Selecting a Row for Display

X (Exclude)

The ‘X’ row command excludes from display the row on which the command was entered such that it can no longer be viewed.

7.4 Primary Commands

The Primary commands supported by the Thread Audit View Facility are those entered in the **Command** field of either the summary or detail panels. The TTSVIEW primary commands are presented in this section in alphabetical order and include full command syntax and usage notes where applicable.

ACTION

The ACTION command displays a view of the Thread_Audit table comprised of those columns which pertain to actions taken vis-a-vis selected threads. Figure 7.5 illustrates the Action view of the Thread_Audit table (in tabular, summary mode) while Figure 7.6 illustrates the Action detail panel.

```
Col 1 of 461 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    2            23       5        6            11                      7
Rcmd Unique Value Plan     Action>> Status       Action Timestamp    >> Policy
'''' B1A02DEE43B0 TTSIvrm  Cancel   Pending      20xx-01-08-17.50.30.15 MANUAL
'''' B1A02DEE43B0 TTSIvrm  Cancel   Successful   20xx-01-08-17.51.23.88 MANUAL
'''' B1A02FA6EC25 TTSIvrm  Cancel   Pending      20xx-01-08-17.55.05.55 POLICY1
'''' B1A02FA6EC25 -        Cancel   Successful   20xx-01-08-17.55.47.67 POLICY1
'''' B1A53BF777A1 TTSIvrm  Force    Pending      20xx-01-12-18.16.12.90 POLICY1
'''' B1A53BF777A1 -        Force    Successful   20xx-01-12-18.16.44.98 POLICY1
'''' B1A658A4074C TTSIvrm  Force    Pending      20xx-01-13-15.48.21.01 POLICY0
'''' B1A658A4074C -        Force    Successful   20xx-01-13-15.48.54.37 POLICY0
```

Figure 7.5 ACTION Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   2 Unique Value (C12)..... N B1A02DEE43B0
''  23 Plan (C8)............... N TTSIvrm
''   5 Action (C12)............ N Cancel
''   6 Status (C12)............ N Pending
''  11 Action Timestamp (TS26)..... N 20xx-01-08-17.50.30.153421
''   7 Policy ID (C8)........... N MANUAL
''  12 Action ACEEID (C8)..... N RAI021
''  10 Action Date (DT10)..... N 01/08/20xx
''  13 Action DB2AUTH (C8)..... N RAI021
''  14 Action Grp (C8)......... N RAIGRP
''  15 Action Term (C8)........ N RAILU021
```

```
''  16 Action Appl (C8)............ N N/P
''  28 SMF ID (C8)................. N RALP
''   1 SSID (C4)................... N DB2E
''  30 Jobname (C8)................ N RAI7IVPE
''   4 ACE Addr (C8)............... N 079BCA18
''   3 Token (I4).................. N  1402
''  31 Where (C4).................. N PGM
''  27 Operator (C8)............... N RAI021
''  26 Connection (C8)............. N BATCH
''  25 Auth ID (C8)................ N RAI021
''  22 Correlation (C12)........... N RAI7IVPE
''   8 Policy Reason (C24)......... N Manually selected thread
''   9 Action Method (C40)......... N DB2 CANCEL THREAD issued
''  17 Action Surr (C8)............ N N/P
''  18 Action CPU (C12)............ N 282373882003
''  19 Creation Date (C10)......... N 01/08/20xx
''  20 Creation Time (C8).......... N 17:46:14
''  21 Creation MS (C7)............ N 6690565
''  24 Pgm Name (C8)............... N TTSIVP
''  29 ASID (C4)................... N 0038
''  32 Accounting Token (C22)...... N N/P
''  33 SQL Count (S2).............. N     122
''  34 Status Code (C2)............ N T
''  35 Net ID (C8)................. N USAMSI0A
''  36 LU Name (C8)................ N DB2APP3
''  37 Commit # (I4)............... N        0
''  38 SQL DML # (I4).............. N      122
```

Figure 7.6 ACTION Detail

AUDIT

The AUDIT command displays a view of the Thread_Audit table in which the 'audit related' columns take precedence. Figure 7.7 illustrates the Audit Summary while Figure 7.8 shows the Audit detail display.

```
Col 1 of 490 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    2            5        6            11                      23       26
Rcmd Unique Value Action>> Action Stat  Action Timestamp    >> Plan     Connect
'''' B1A02DEE43B0 Cancel   Pending      20xx-01-08-17.50.30.15 TTSIvrm  BATCH
'''' B1A02DEE43B0 Cancel   Successful   20xx-01-08-17.51.23.88 TTSIvrm  BATCH
'''' B1A02FA6EC25 Cancel   Pending      20xx-01-08-17.55.05.55 TTSIvrm  BATCH
'''' B1A02FA6EC25 Cancel   Successful   20xx-01-08-17.55.47.67 -        -
'''' B1A53BF777A1 Force    Pending      20xx-01-12-18.16.12.90 TTSIvrm  BATCH
'''' B1A53BF777A1 Force    Successful   20xx-01-12-18.16.44.98 -        -
'''' B1A658A4074C Force    Pending      20xx-01-13-15.48.21.01 TTSIvrm  BATCH
'''' B1A658A4074C Force    Successful   20xx-01-13-15.48.54.37 -        -
```

Figure 7.7 Audit Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   2 Unique Value (C12)..... N B1A02DEE43B0
''   5 Action (C12)........... N Cancel
''   6 Action Stat (C12)...... N Pending
''  11 Action Timestamp (TS26)..... N 20xx-01-08-17.50.30.153421
''  23 Plan (C8)............... N TTSIvrm
''  26 Connection (C8)......... N BATCH
''  25 Auth ID (C8)............ N RAI021
''  22 Correlation (C12)....... N RAI7IVPE
''   7 Policy ID (C8)........... N MANUAL
''   8 Policy Reason (C24)..... N Manually selected thread
''   9 Action Method (C40)..... N DB2 CANCEL THREAD issued
''  10 Action Date (DT10)...... N 01/08/20xx
''  12 Action ACEE (C8)........ N RAI021
''  13 Action DB2AUTH (C8)..... N RAI021
''  14 Action Grp (C8)......... N RAIGRP
''  15 Action Term (C8)........ N RAILU021
''  16 Action Appl (C8)........ N N/P
''  17 Action Surr (C8)........ N N/P
''  18 Action CPU (C12)........ N 282373882003
''   3 Token (I4).............. N  1402
''  19 Creation Date (C10)..... N 01/08/20xx
''  20 Creation Time (C8)...... N 17:46:14
''  21 Creation Micro (C7)..... N 6690565
''  24 Pgm Name (C8)........... N TTSIVP
''  30 Jobname (C8)............ N RAI7IVPE
''  31 Where (C4).............. N PGM
''  27 Operator (C8)........... N RAI021
''  28 SMF ID (C8)............. N RALP
''  29 ASID (C4)............... N 0038
''  32 Accounting Token (C22)...... N N/P
''  33 SQL Count (S2).......... N     122
''  34 Status Code (C2)........ N T
''  35 Net ID (C8)............. N USAMSI0A
''  36 LU Name (C8)............ N DB2APP3
''  37 Commit # (I4)........... N        0
''  38 SQL DML # (I4).......... N      122
''  39 Getpages (I4)........... N         14
''  40 Status Literal (C40)..... N Local (non-distributed) thread
```

Figure 7.8 Audit Detail

DB2ACCT

The DB2ACCT command displays a view of the Thread_Audit table in which the columns related to MVS and DDF accounting data take precedence. Figure 7.9 illustrates the DB2ACCT Summary, while Figure 7.10 shows the DB2ACCT detail display.

> ***NOTE:*** *Meaningful MVS and DDF accounting information is only provided for those threads whose PRODUCT_NAME value is 'DSN'. That is, threads whose accounting strings are generated by DB2 for z/OS.*

```
Col 1 of 241 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-               3 65           66           67           68             69
Rcmd Thread Token QMDA Product QMDA Version QMDA Release QMDA Mod Level QMDA Lo
''''         1402 DSN          Ø4           Ø1           Ø              RADB2E
''''         1402 DSN          Ø4           Ø1           Ø              RADB2E
''''         1405 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
''''          148 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
''''          2Ø1 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
```

Figure 7.9 DB2ACCT Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   3 Thread Token (I4)........... N  1402
''  65 QMDA Product (C3)........... N DSN
''  66 QMDA Version (C2)........... N Ø4
''  67 QMDA Release (C2)........... N Ø1
''  68 QMDA Mod Level (C1)......... N Ø
''  69 QMDA Location (C16)......... N RADB2E
''  7Ø QMDA NetID (C8)............. N USAMSIØA
''  71 QMDA LU Name (C8)........... N DB2APP3
''  72 QMDA Connect Name (C8)...... N BATCH
''  73 QMDA Connect Type (C8)...... N BATCH
''  74 QMDA Correlation (C12)...... N RAI7IVPE
''  75 QMDA AuthID (C8)............ N RAIØ21
''  76 QMDA Plan (C8).............. N TTSIvrm
''  8Ø Account String Length (S2).. N       9
''  81 Accounting String (V2ØØ).... N RAIØ 1234
             cont(ØØ2) ØØØ41-ØØØ8Ø
             cont(ØØ3) ØØØ81-ØØ12Ø
             cont(ØØ4) ØØ121-ØØ16Ø
```

Figure 7.10 DB2ACCT Detail

QBE

The QBE command produces the initial view of the Thread_Audit table that results after you press Enter from the Query by Example panel. Figure 7.11 illustrates the QBE Summary, while Figure 7.12 shows the QBE detail display.

```
Col 1 of 181 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    23        24           22             25               26              7
Rcmd Plan Name Program Name Correlation ID Authorization ID Connection Name Pol
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           MAN
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           POL
'''' -         -            -              -                -               POL
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI020           BATCH           POL
'''' -         -            -              -                -               POL
'''' TTSIvrm   TTSIVP       RAI7IVPE       RAI021           BATCH           POL
'''' -         -            -              -                -               POL
```

Figure 7.11 QBE Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''  23 Plan Name (C8)............. N TTSIvrm
''  24 Program Name (C8)........... N TTSIVP
''  22 Correlation ID (C12)........ N RAI7IVPE
''  25 Authorization ID (C8)....... N RAI021
''  26 Connection Name (C8)........ N BATCH
''   7 Policy ID (C8).............. N MANUAL
''   8 Policy Reason (C24)......... N Manually selected thread
''   2 Unique Value (C12).......... N B1A02DEE43B0
''   3 Thread Token (I4)........... N  1402
''   5 Action (C12)................ N Cancel
''  10 Action Date (DT10).......... N 01/08/20xx
''  11 Action Timestamp (TS26)..... N 20xx-01-08-17.50.30.153421
''   1 SSID (C4)................... N DB2E
''   4 ACE Address (C8)............ N 079BCA18
''   6 Action Status (C12)......... N Pending
''   9 Action Method (C40)......... N DB2 CANCEL THREAD issued
''  12 Action ACEE (C8)............ N RAI021
''  13 Action SSID (C8)............ N RAI021
''  14 Action Grp (C8)............. N RAIGRP
''  15 Action Term (C8)............ N RAILU021
''  16 Action Appl (C8)............ N N/P
''  17 Action Surr (C8)............ N N/P
''  18 Action CPU (C12)............ N 282373882003
''  19 Creation Date (C10)......... N 01/08/20xx
''  20 Creation Time (C8).......... N 17:46:14
```

```
''  21 Creation Micro (C7)......... N 6690565
''  27 Operator (C8)............... N RAI021
''  28 SMF ID (C8)................. N RALP
''  29 ASID (C4)................... N 0038
''  30 Jobname (C8)................ N RAI7IVPE
''  31 Where (C4).................. N PGM
''  32 Accounting Token (C22)...... N N/P
''  33 SQL Count (S2).............. N     122
''  34 Status Code (C2)............ N T
''  35 Net ID (C8)................. N USAMSI0A
''  36 LU Name (C8)................ N DB2APP3
''  37 Commit # (I4)............... N        0
''  38 SQL DML # (I4).............. N      122
''  39 Getpages (I4)............... N         14
''  40 Status Literal (C40)........ N Local (non-distributed) thread
''  41 Connection Code (I4)........ N              1
''  42 Connecting System (C12)..... N TSO
''  43 TCB Addr (C8)............... N 00000000
''  44 Elapsed Total (C15)......... N 00:04:14.055481
''  45 Elapsed DB2 (C15)........... N 00:00:00.261288
''  46 Class 1 Time (C15).......... N 00:00:00.033104
''  47 Class 2 Time (C15).......... N 00:00:00.023257
''  48 SRB Time (C15).............. N 00:00:00.000742
''  49 IO Wait Time (C15).......... N 00:00:00.000000
''  50 Lock Wait Time (C15)........ N 00:00:00.000000
''  51 Distributed Location (C16).. N N/P
''  52 APPC ID (C8)................ N N/P
''  53 Distributed Session (C16)... N N/P
''  65 QMDA Product (C3)........... N DSN
''  66 QMDA Version (C2)........... N 04
''  67 QMDA Release (C2)........... N 01
''  68 QMDA Mod Level (C1)......... N 0
''  69 QMDA Location (C16)......... N RADB2E
''  70 QMDA Net ID (C8)............ N USAMSI0A
''  71 QMDA LU Name (C8)........... N DB2APP3
''  72 QMDA Connect Name (C8)...... N BATCH
''  73 QMDA Connect Type (C8)...... N BATCH
''  74 QMDA Correlation (C12)...... N RAI7IVPE
''  75 QMDA AuthID (C8)............ N RAI021
''  76 QMDA Plan (C8).............. N TTSIVRM
''  77 Client Platform (C18)....... N ..................
''  78 Client Application (C20).... N ......................
''  79 Client AuthID (C8).......... N ........
''  80 Account String Length (S2).. N       9
''  81 Accounting String (V200).... N RAI0 1234
           cont(002) 00041-00080
           cont(003) 00081-00120
           cont(004) 00121-00160
           cont(005) 00161-00200
```

Figure 7.12 QBE Detail

QMDA

The QMDA command displays a view of the Thread_Audit table in which the columns related to MVS and DDF accounting data take precedence. Figure 7.13 illustrates the QMDA Summary, while Figure 7.14 shows the QMDA detail display.

```
Col 1 of 295 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-               3 65           66           67           68             69
Rcmd Thread Token QMDA Product QMDA Version QMDA Release QMDA Mod Level QMDA Lo
''''         1402 DSN          Ø4           Ø1           Ø              RADB2E
''''         1402 DSN          Ø4           Ø1           Ø              RADB2E
''''         1405 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
''''          148 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
''''          201 DSN          Ø4           Ø1           Ø              RADB2E
''''            - -            -            -            -              -
```

Figure 7.13 QMDA Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   3 Thread Token (I4)........... N  1402
''  65 QMDA Product (C3)........... N DSN
''  66 QMDA Version (C2)........... N Ø4
''  67 QMDA Release (C2)........... N Ø1
''  68 QMDA Mod Level (C1)......... N Ø
''  69 QMDA Location (C16)......... N RADB2E
''  70 QMDA NetID (C8)............. N USAMSIØA
''  71 QMDA LU Name (C8)........... N DB2APP3
''  72 QMDA Connect Name (C8)...... N BATCH
''  73 QMDA Connect Type (C8)...... N BATCH
''  74 QMDA Correlation (C12)...... N RAI7IVPE
''  75 QMDA AuthID (C8)............ N RAIØ21
''  76 QMDA Plan (C8).............. N TTSIvrm
''  77 Client Platform (C18)....... N ..................
''  78 Client Application (C20).... N ....................
''  79 Client AuthID (C8).......... N ........
''  80 Account String Length (S2).. N       9
''  81 Accounting String (V200).... N RAIØ 1234
             cont(002) 00041-00080
             cont(003) 00081-00120
             cont(004) 00121-00160
             cont(005) 00161-00200
```

Figure 7.14 QMDA Detail

SQLACCT

The SQLACCT command displays a view of the Thread_Audit table in which the columns related to MVS and DDF accounting data take precedence. Figure 7.15 illustrates the SQLACCT Summary while Figure 7.16 shows the SQLACCT detail display.

> ***NOTE**: Meaningful MVS and DDF accounting information is only provided for those threads whose PRODUCT_NAME value is 'SQL'. That is, threads whose accounting strings are created by DB2 client server products such as DB2 for Windows NT, DB2 for OS/2 and DB2 for various Unix implementations.*

```
Col 1 of 249 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                 Scroll ===> HALF

-               3 65           66           67           68             77
Rcmd Thread Token QMDA Product QMDA Version QMDA Release QMDA Mod Level Client
''''         1402 DSN          Ø4           Ø1           Ø
''''         1402 DSN          Ø4           Ø1           Ø
''''         1405 DSN          Ø4           Ø1           Ø
''''            - -            -            -            -              -
''''          148 DSN          Ø4           Ø1           Ø
''''            - -            -            -            -              -
''''          2Ø1 DSN          Ø4           Ø1           Ø
''''            - -            -            -            -              -
```

Figure 7.15 SQLACCT Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                 Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   3 Thread Token (I4)........... N  1402
''  65 QMDA Product (C3)........... N DSN
''  66 QMDA Version (C2)........... N Ø4
''  67 QMDA Release (C2)........... N Ø1
''  68 QMDA Mod Level (C1)......... N Ø
''  77 Client Platform (C18)....... N ..................
''  78 Client Application (C2Ø).... N .....................
''  79 Client AuthID (C8).......... N ........
''  8Ø Account String Length (S2).. N       9
''  81 Accounting String (V2ØØ).... N RAIØ 1234
             cont(ØØ2) ØØØ41-ØØØ8Ø
             cont(ØØ3) ØØØ81-ØØ12Ø
             cont(ØØ4) ØØ121-ØØ16Ø
             cont(ØØ5) ØØ161-ØØ2ØØ
''   5 Action (C12)................ N Cancel
''   6 Action Stat (C12)........... N Pending
''   7 Policy ID (C8).............. N MANUAL
''   8 Policy Reason (C24)......... N Manually selected thread
''   2 Unique Value (C12).......... N B1AØ2DEE43BØ
```

Figure 7.16 SQLACCT Detail

SYSTEM

The SYSTEM command displays a view of the Thread_Audit table which features the 'System related' columns. Figure 7.17 illustrates the System related Summary panel, while Figure 7.18 shows the System Detail display.

```
Col 1 of 458 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    2            5        28       1    30       23            3 18
Rcmd Unique Value Action>> SMF ID   SSID Jobname  Plan Name Token Action CPU
'''' B1A02DEE43B0 Cancel   RALP     DB2E RAI7IVPE TTSIvrm    1402 282373882003
'''' B1A02DEE43B0 Cancel   RALP     DB2E RAI7IVPE TTSIvrm    1402 280373882003
'''' B1A02FA6EC25 Cancel   RALP     DB2E RAI7IVPE TTSIvrm    1405 282373882003
'''' B1A02FA6EC25 Cancel   -        -    -        -             - -
'''' B1A53BF777A1 Force    RALP     DB2E RAI7IVPE TTSIvrm     148 281373882003
'''' B1A53BF777A1 Force    -        -    -        -             - -
'''' B1A658A4074C Force    RALP     DB2E RAI7IVPE TTSIvrm     201 280373882003
'''' B1A658A4074C Force    -        -    -        -             - -
```

Figure 7.17 System Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   2 Unique Value (C12)..... N B1A02DEE43B0
''   5 Action (C12)............ N Cancel
''  28 SMF ID (C8)............. N RALP
''   1 SSID (C4)............... N DB2E
''  30 Jobname (C8)............ N RAI7IVPE
''  23 Plan Name (C8).......... N TTSIvrm
''   3 Token (I4).............. N  1402
''  18 Action CPU (C12)....... N 282373882003
''   6 Action Status (C12)..... N Pending
''  11 Action Time (TS26)..... N 20xx-01-08-17.50.30.153421
''  31 Where (C4)............. N PGM
''   4 ACE Addr (C8)........... N 079BCA18
''  27 Operator (C8)........... N RAI021
''  26 Connection (C8)......... N BATCH
''  25 Auth ID (C8)............ N RAI021
''  22 Corrrelation (C12)..... N RAI7IVPE
''   7 Policy ID (C8)........... N MANUAL
''   8 Policy Reason (C24)..... N Manually selected thread
''   9 Action Method (C40)..... N DB2 CANCEL THREAD issued
''  10 Action Date (DT10)..... N 01/08/20xx
''  12 Action ACEE (C8)....... N RAI021
''  13 Action SSID (C8)....... N RAI021
''  14 Action Grp (C8)........ N RAIGRP
''  15 Action Term (C8)....... N RAILU021
''  16 Action Appl (C8)....... N N/P
''  17 Action Surr (C8)....... N N/P
```

```
''  19 Creation Date (C1Ø)......... N Ø1/Ø8/2Øxx
''  2Ø Creation Time (C8).......... N 17:46:14
''  21 Creation MS (C7)............ N 6690565
''  24 PGM Name (C8)............... N TTSIVP
''  29 ASID (C4)................... N ØØ38
''  32 Accounting Token (C22)...... N N/P
''  33 SQL Count (S2).............. N     122
''  34 Status Code (C2)............ N T
''  35 Net ID (C8)................. N USAMSIØA
''  36 LU Name (C8)................ N DB2APP3
''  37 Commit (I4)................. N        Ø
''  38 SQL DML # (I4).............. N      122
```

Figure 7.18 System Detail

TABLE

The TABLE command displays the base view of the Thread_Audit table in which all columns appear in the order in which they are defined to DB2. Figure 7.19 illustrates the Table Summary, while Figures 7.20 through 7.22 show the Table detail display.

```
Col 1 of 1267 --------- Data Object: TTSvrm.THREAD_AUDIT -------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    1    2                          3 4        5            6             7
Rcmd SSID Uniqueness Value Thread Token ACE Addr Action Taken Action Status Pol
'''' DB2E B1A02DEE43BØ             1402 Ø79BCA18 Cancel       Pending       MAN
'''' DB2E B1A02DEE43BØ             1402 Ø79BCA18 Cancel       Successful    MAN
'''' DB2E B1A02FA6EC25             1405 Ø79BCA18 Cancel       Pending       POL
'''' -    B1A02FA6EC25                - -        Cancel       Successful    POL
'''' DB2E B1A53BF777A1              148 Ø784CA18 Force        Pending       POL
'''' -    B1A53BF777A1                - -        Force        Successful    POL
'''' DB2E B1A658A4Ø74C              2Ø1 Ø7B561A8 Force        Pending       POL
'''' -    B1A658A4Ø74C                - -        Force        Successful    POL
```

Figure 7.19 Table Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   1 SSID (C4)................. N DB2E
''   2 Uniqueness Value (C12)..... N B1A02DEE43B0
''   3 Thread Token (I4).......... N         1402
''   4 ACE Addr (C8).............. N 079BCA18
''   5 Action Taken (C12)......... N Cancel
''   6 Action Status (C12)........ N Pending
''   7 Policy ID (C8)............. N MANUAL
''   8 Policy Reason (C24)........ N Manually selected thread
''   9 Action Method (C40)........ N DB2 CANCEL THREAD issued
''  10 Action Date (DT10)........ N 01/08/20xx
''  11 Action Timestamp (TS26)..... N 20xx-01-08-17.50.30.153421
''  12 Action ACEE ID (C8)........ N RAI021
''  13 Action DB2AUTH ID (C8)..... N RAI021
''  14 Action Group_Name (C8)..... N RAIGRP
''  15 Action TERM ID (C8)........ N RAILU021
''  16 Action APPL ID (C8)........ N N/P
''  17 Action SURR ID (C8)........ N N/P
''  18 Action CPU ID (C12)........ N 282373882003
''  19 Creation Date (C10)........ N 01/08/20xx
''  20 Creation Time (C8)......... N 17:46:14
''  21 Creation Microsecs (C7)..... N 6690565
''  22 Correlation Name (C12)..... N RAI7IVPE
''  23 Plan Name (C8)............. N TTSIvrm
''  24 Program_Name (C8).......... N TTSIVP
''  25 Authorization ID (C8)...... N RAI021
''  26 Connection Name (C8)....... N BATCH
''  27 Original Operator (C8)..... N RAI021
''  28 MVS System (C8)............ N RALP
''  29 Address Space ID (C4)...... N 0038
''  30 Thread Jobname (C8)........ N RAI7IVPE
''  31 Where Executing (C4)....... N PGM
''  32 Accounting Token (C22)..... N N/P
''  33 Display SQL Count (S2)..... N     122
''  34 Status Code (C2)........... N T
''  35 Network ID (C8)............ N USAMSI0A
''  36 LU Name (C8)............... N DB2APP3
''  37 Commit Count (I4).......... N              0
''  38 SQL DML Count (I4)......... N            122
```

Figure 7.20 Table Detail - Part I

```
Col 39 of 81 ----------- DB2 Table: TTSvrm.THREAD_AUDIT ----------- Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''  39 Getpages Issued (I4)....... N             14
''  40 Status Literal (C40)....... N Local (non-distributed) thread
''  41 Connection Code (I4)....... N              1
''  42 Connecting System (C12)..... N TSO
''  43 TCB Address (C8)........... N 00000000
```

```
''  44 Total Elapsed Time (C15).... N 00:04:14.055481
''  45 DB2 Elapsed Time (C15)...... N 00:00:00.261288
''  46 Class1 TCB Time (C15)....... N 00:00:00.033104
''  47 Class2 TCB_Time (C15)....... N 00:00:00.023257
''  48 Home SRB Time (C15)......... N 00:00:00.000742
''  49 IO Wait Time (C15).......... N 00:00:00.000000
''  50 Lock Wait Time (C15)........ N 00:00:00.000000
''  51 Dist Location (C16)......... N N/P
''  52 Dist_APPC ID (C8)........... N N/P
''  53 Dist Session ID (C16)....... N N/P
''  54 THREAD_FLAG1 (C1)........... N D
''  55 THREAD_FLAG2 (C1)........... N
''  56 THREAD_FLAG3 (C1)........... N
''  57 THREAD_FLAG4 (C1)........... N T
''  58 THREAD_FLAG5 (C1)........... N A
''  59 THREAD_FLAG6 (C1)........... N
''  60 THREAD_FLAG7 (C1)........... N W
''  61 THREAD_FLAG8 (C1)........... N
''  62 THREAD_FLAG9 (C1)........... N P
''  63 THREAD_FLAGA (C1)........... N
''  64 THREAD_FLAGB (C1)........... N C
''  65 QMDA Product (C3)........... N DSN
''  66 QMDA Version (C2)........... N 04
''  67 QMDA Release (C2)........... N 01
''  68 QMDA Mod Level (C1)......... N 0
''  69 QMDA Location (C16)......... N RADB2E
''  70 QMDA NetID (C8)............. N USAMSI0A
''  71 QMDA LU Name (C8)........... N DB2APP3
''  72 QMDA Connect Name (C8)...... N BATCH
''  73 QMDA Connect Type (C8)...... N BATCH
''  74 QMDA Correlation (C12)...... N RAI7IVPE
''  75 QMDA AuthID (C8)............ N RAI021
''  76 QMDA Plan (C8).............. N TTSIvrm
```

Figure 7.21 Table Detail – Part II

```
Col 48 of 81 ----------- DB2 Table: TTSvrm.THREAD_AUDIT ----------- Row 1 of 8
Command ===>                                                 Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''  77 Client Platform (C18)..... N ..................
''  78 Client Application (C20).... N ..................
''  79 Client AuthID (C8).......... N ........
''  80 Account String Length (S2).. N       9
''  81 Accounting String (V200).... N RAI0 1234
             cont(002) 00041-00080
             cont(003) 00081-00120
             cont(004) 00121-00160
             cont(005) 00161-00200
```

Figure 7.22 Table Detail – Part III

TIME

The TIME command displays a view of the Thread_Audit table in which the 'timing related' columns take precedence. Figure 7.23 illustrates the Timing Summary while Figure 7.24 shows the Timing detail displayx.

```
Col 1 of 490 --------- Data Object: TTSvrm.THREAD_AUDIT --------- Row 1 from 8
Command ===>                                                  Scroll ===> HALF

-    2            23        44              45              46              47
Rcmd Unique Value Plan Name Total Time      DB2 Time        Class1 Time     Cla
'''' B1A02DEE43B0 TTSIvrm   00:04:14.055481 00:00:00.261288 00:00:00.033104 00:
'''' B1A02DEE43B0 TTSIvrm   00:04:19.754915 00:00:00.261288 00:00:00.033104 00:
'''' B1A02FA6EC25 TTSIvrm   00:01:07.521893 00:00:00.040195 00:00:00.013172 00:
'''' B1A02FA6EC25 -         -               -               -               -
'''' B1A53BF777A1 TTSIvrm   00:00:30.942471 00:00:00.430579 00:00:00.008735 00:
'''' B1A53BF777A1 -         -               -               -               -
'''' B1A658A4074C TTSIvrm   00:19:03.321219 00:00:00.350691 00:00:00.129463 00:
'''' B1A658A4074C -         -               -               -               -
```

Figure 7.23 Time Summary

```
Col 1 of 81 ------------ DB2 Table: TTSvrm.THREAD_AUDIT ------------ Row 1 of 8
Command ===>                                                  Scroll ===> HALF

Row Cmd ===> _ Status ==>
Rc Seq Column name/Type          Null Column Value
''   2 Unique Value (C12)..... N B1A02DEE43B0
''  23 Plan Name (C8)........... N TTSIvrm
''  44 Total Time (C15)......... N 00:04:14.055481
''  45 DB2 Time (C15)........... N 00:00:00.261288
''  46 Class1 Time (C15)........ N 00:00:00.033104
''  47 Class2 Time (C15)........ N 00:00:00.023257
''  48 SRB Time (C15)........... N 00:00:00.000742
''  49 IO Wait Time (C15)....... N 00:00:00.000000
''  50 Lock Wait Time (C15)..... N 00:00:00.000000
''   5 Action Taken (C12)....... N Cancel
''  11 Action Timestamp (TS26).. N 20xx-01-08-17.50.30.153421
''  28 SMF ID (C8).............. N RALP
''   1 SSID (C4)................ N DB2E
''  30 Jobname (C8)............. N RAI7IVPE
''   4 ACE Addr (C8)............ N 079BCA18
''   3 Token (I4)............... N  1402
''   6 Action Status (C12)...... N Pending
''  31 Where (C4)............... N PGM
''  27 Operator (C8)............ N RAI021
''  26 Connection (C8).......... N BATCH
''  25 Auth ID (C8)............. N RAI021
''  22 Correlation (C12)........ N RAI7IVPE
''   7 Policy ID (C8)........... N MANUAL
```

```
''   8 Policy Reason (C24)......... N Manually selected thread
''   9 Action Method (C40)......... N DB2 CANCEL THREAD issued
''  10 Action Date (DT10).......... N 01/08/20xx
''  12 Action ACEE (C8)............ N RAI021
''  13 Action SSID (C8)............ N RAI021
''  14 Action GRP (C8)............. N RAIGRP
''  15 Action Term (C8)............ N RAILU021
''  16 Action Appl (C8)............ N N/P
''  17 Action Surr (C8)............ N N/P
''  18 Action CPU (C12)............ N 282373882003
''  19 Creation Date (C10)......... N 01/08/20xx
''  20 Creation Time (C8).......... N 17:46:14
''  21 Creation MS (C7)............ N 6690565
''  24 PGM Name (C8)............... N TTSIVP
''  29 ASID (C4)................... N 0038
```

Figure 7.24 Time Detail

Chapter 10

Thread/SENTRY Overview and Policy Definition

This chapter describes the operation and use of Thread/SENTRY, a product which provides automated monitoring and policy enforcement for DB2 threads. Thread/SENTRY makes it unnecessary to manually identify DB2 problem threads or to explicitly cancel them. Instead, you can define what constitutes a 'problem thread' in terms of threshold measurements such as elapsed time, CPU time, number of SQL statements executed, number of locks held, etc. Thread/SENTRY detects such problem threads *automatically* based on criteria you pre-define.

Thread/SENTRY (occasionally referred to herein as "the monitor") wakes up periodically (at an interval you specify) to monitor DB2 thread activity. You can direct Thread/SENTRY to issue a warning, cancel a thread or invoke a site written exit routine whenever Thread/SENTRY detects a violation. You can also direct Thread/SENTRY to quiesce or resume the address space associated with a DB2 thread. Thread/SENTRY evaluates DB2 threads in real-time and enforces your site-defined policies automatically. Moreover, Thread/SENTRY can monitor multiple DB2 subsystems (on the same MVS image) simultaneously -- ***even DB2 subsystems at different version and release levels.***

Section 10.1 presents an overview of Thread/SENTRY features and capabilities. Section 10.2 summarizes the statements with which to define your site policies while Section 10.3 describes and illustrates the individual Thread/SENTRY control statements in detail. The JCL with which to run Thread/SENTRY and the console commands that control its operation are described and illustrated in Chapter 11.

10.1 Thread/SENTRY Overview

Thread/SENTRY provides a dedicated facility to *automatically* detect problem DB2 threads. Thread/SENTRY can *gracefully cancel* these problem threads and do so with minimal impact. Thread/SENTRY lets you deal with the inevitable operational mistakes, network failures, application errors and runaway queries that create 'problem' DB2 threads -- *without* disrupting large numbers of users.

Thread/SENTRY terminates a DB2 thread without canceling the thread's allied address space. Thread cancellation via Thread/SENTRY is more granular and far less drastic than recycling an entire CICS or IMS transaction processing region's connection to DB2 via STOP and START commands. Thread/SENTRY is also more granular in operation than the MVS CANCEL and FORCE commands. While these commands terminate an entire job, TSO session or started task, Thread/SENTRY cancels just the DB2 thread or at most the MVS task associated with a DB2 thread.

Moreover, Thread/SENTRY works in cases where DB2's own CANCEL THREAD and CANCEL DDF THREAD commands are ineffective. This can occur when the thread is hung up in the application or within the communications network (TCP/IP and/or SNA/VTAM).

Thread/SENTRY supports various methods of thread cancellation using DB2, MVS TCP/IP and/or SNA/VTAM commands -- both individually and in combination. Thread/SENTRY automatically determines the correct method to use, based on the current state of the DB2 thread. Thread/SENTRY utilizes the same thread cancellation facilities as does Thread/STOPPER.

Thread/SENTRY is an enterprise solution that supports all *local* and *distributed* DB2 applications connected through the various DB2 attachment facilities. These include CICS and IMS transactions, client/server applications connected through gateways, TSO sessions, QMF users and DB2 batch jobs.

10.2 Thread/SENTRY Statement Summary

Your organization can define what constitutes a DB2 'problem thread' in terms of threshold values. These limits define policies that Thread/SENTRY enforces automatically. Administrators define site policies through Thread/SENTRY ***statements***.

This section lists the Thread/SENTRY control statements and briefly describes their function. Subsequent sections describe each Thread/SENTRY statement in turn. Each section describes the syntax and operands of one control statement and provides one or more annotated examples to illustrate its use.

Thread/SENTRY processes the following statement types:

The **MONITOR** statement defines characteristics of the Thread/SENTRY application itself, such as the monitoring interval and the names of the DB2 subsystems to be monitored.

The **EXCLUDE** statement lets you define threads that Thread/SENTRY should *ignore*. Excluded threads are neither monitored nor canceled.

Note: See note under LIMIT.

The **LIMIT** statement permits the specification of various minimum and maximum thresholds for resource consumption by threads that is observed over

(1) the life a of thread,
(2) the period encompassing a single unit-of-recovery; or
(3) fixed intervals of elapsed time (termed minmax intervals).

Thread activity includes such measures as elapsed time, CPU time, number of SQL statements executed, etc. When a maximum threshold is exceeded (or a minimum threshold is not reached), Thread/SENTRY performs the action defined on the LIMIT statement. The ACTION operand of the LIMIT statement can direct Thread/SENTRY to issue a warning; cancel, quiesce or resume the thread; or call a site-written exit routine that decides what to do.

> Note: The EXCLUDE and LIMIT statements are all **maskable,** through the SQL wild card characters '%' and '_'. This lets you identify **which** threads are subject to evaluation on the basis of variables such as plan name, Auth ID, correlation name, etc. For example, you can restrict the application of a policy to CICS threads through a specification such as: CONNECTION(CICS%). Other operands of the EXCLUDE and LIMIT statements let you restrict **when** these policies should be evaluated to specific days and/or times.

The **DEFAULT** statement and its operands let you specify default values for a set of EXCLUDE and LIMIT statements that follow it. The DEFAULT statement is not executable as such. The values defined on a DEFAULT statement remain in effect until ***(1)*** another DEFAULT statement is encountered or ***(2)*** no additional Thread/SENTRY control statements remain to be processed.

Whenever possible, Thread/SENTRY notifies users whose threads are subject to warning or cancellation (unless these notifications are specifically suppressed as described in the discussion of _WTO below). Thread/SENTRY can also alert *additional* administrative user(s) whenever it issues warnings or cancellations. The **NOTIFY_LIST** statement lets you define a list of administrative IDs to be notified when a policy violation takes place. You can assign a symbolic name to this notification list which can appear in the previously defined LIMIT statements. When Thread/SENTRY detects a policy violation, it can notify all the IDs specified in the notification list.

However, when the referenced NOTIFY_LIST symbolic name starts with the special string `_WTO` (as in `_WTOABC`), Thread/SENTRY writes the user notification message to the

operator (via a WTO), rather than send a notification message to the owner of the thread. In contrast, the notifications sent to administrative users specified in the NOTIFY_LIST are processed without change (i.e. not effected by the _WTO special string).

Structure of Control Statements

Figure 10.1 illustrates the hierarchical structure of Thread/SENTRY control statements. (Statements are indented to better portray the statement hierarchy. This indentation is not required.) Note that Thread/SENTRY policy statements within a category are processed in a FIFO (first in, first out) sequence.

```
MONITOR
   DEFAULT
      EXCLUDE
      LIMIT
   DEFAULT
      LIMIT
      LIMIT
      LIMIT
   NOTIFY_LIST
```

Figure 10.1 Thread/SENTRY hierarchical statement structure

These statements are read in from a control file during Thread/SENTRY initialization. You can also refresh this control file, dynamically *during execution*. (See the RULE_REFRESH command in Section 11.2.3).

Thread/SENTRY statements and their operands will steadily evolve over time to expose new thread variables and threshold measures to automated monitoring and governance. Thread/SENTRY maintains your policy definitions in control file(s) rather than DB2 tables to make it unnecessary to ALTER DB2 tables and rebind the Thread/SENTRY plans each time the syntax is enhanced.

We expect the evolution of Thread/SENTRY's statement syntax will also tend to reduce the number of exit programs an organization has to develop in order to implement their site specific policies.

10.2.1 Thread Selection Criteria

Thread/SENTRY applies AND logic to select the threads to be subject to a given policy. For example, suppose a LIMIT policy specifies both PLAN name and AUTH ID values. Thread/SENTRY will evaluate only those threads whose plan name ***and*** AuthID match those defined by the policy. As another example, suppose an EXCLUDE policy specifies both a CONNECTION name value as well as a DAYS operand which specifies the policy should only be in force during weekdays (Monday through Friday). In such a case, Thread/SENTRY will exclude only those threads which meet both criteria: their CONNECTION name matches that defined by the policy ***and*** the thread is executing on a weekday. A thread is subject to a given policy only if the thread matches all the WHAT and WHEN criteria specified by the policy.

Once Thread/SENTRY selects a thread for evaluation, it employs either AND or OR logic with respect to the threshold operands that may trigger a policy violation. (The use of AND or OR logic for a given policy is governed by the LOGIC operand described in Section 10.7.2.) For example, a policy might limit the CPU time a thread can consume as well as the number of SQL calls the thread can issue. When AND logic is applied, Thread/SENTRY takes the action specified for the policy when the thread violates both the CPU limit ***and*** the SQL call limit. Alternatively, when OR logic is applied, Thread/SENTRY takes action when the thread violates either the CPU limit ***or*** the SQL call limit.

10.2.2 Specifying Time Values

Thread/SENTRY accepts time values either as an integer number of seconds or as formatted values that include hours, minutes and seconds.

When just digits are specified (without colon or period separators), then the entire time value is construed to be seconds as in TIME(5) or TIME(10) -- e.g., TIME(1800) denotes 1800 seconds or thirty minutes.

Alternatively, the presence of either the period ('.') or colon (':') denotes a formatted time comprised of hours, minutes and seconds. The following formats are supported: (where hh = hours, mm = minutes and ss = seconds.)

```
ISO     hh.mm.ss
USA     hh:mm:ss
EUR     hh.mm.ss
JIS     hh:mm:ss
```

For example 2:30 denotes 2 hours and 30 minutes while 10.45.15 denotes 10 hours, 45 minutes and 15 seconds. Similarly, 19:00 denotes 19 hundred hours or 7 PM.

10.3 Thread/SENTRY Control Statements

The following sections describes the various Thread/SENTRY control statements and their operands. The following notational conventions are used in this Chapter:

> Keywords appear in UPPERCASE (for example SSID) and should be coded exactly as shown

> Site-dependent values are denoted by lowercase characters. For example: (userid)

> Operands shown in brackets [ ] are optional. You can choose from a list of values separated by the vertical bar | . For example: MODE(CANCEL|WARN). One or none of the values may be chosen; the default values are underscored.

> Operands shown in braces { } are alternatives; one must be chosen.

> An ellipsis (. . .) indicates the parameter may be repeated to specify additional items of the same category.

You can code comments within the delimiters /* and */ to document Thread/SENTRY control statements and their operands.

10.4 MONITOR Statement

The MONITOR statement defines operational characteristics of Thread/SENTRY itself, such as its wakeup interval and the names of the DB2 subsystems to be monitored. The MONITOR statement can be defined in a control file, or its operands can be passed as parameters to the Thread/SENTRY program through the JCL EXEC statement. Alternatively, you can at installation time assemble and link-edit a set of MONITOR statement values into a load module which defines the Thread/SENTRY defaults. Operands of the MONITOR statement (in alphabetical order) include the following:

```
DESCRCDE(descriptor_code)
```

The DESCRCDE operand applies only if Thread/SENTRY output is directed to the operator via OUTPUT(WTO). It denotes an MCS descriptor code for WTO requests issued by Thread/SENTRY.

```
ESTAE(YES|NO)
```

Governs whether abend trapping and MVS task recovery are enabled for the Thread/SENTRY main and subtasks. ESTAE(YES) is the default and should not be changed unless you are directed to do so by vendor support personnel.

`EXIT_MODULE(load_module_name)`

Identifies the name of the load module which contains the Thread/SERIES Table of Exit Routine definitions to be used during this run. Each entry within the Thread/SERIES Table of Exit Routines defines either a site written or vendor supplied exit routine as defined by the TTS#TXR macro. See Appendix E for details.

`IDLE_THREAD(YES|NO)`

The IDLE_THREAD operand governs whether or not Thread/SENTRY monitors inactive DDF threads (i.e. IDLE threads) for policy violations. Thread/SENTRY incurs significant overhead to obtain information about idle threads so the default is IDLE_THREAD(NO).

`MCS_CONSOLE(YES|NO)`

The MCS_CONSOLE operand governs whether or not Thread/SENTRY activates an MCS console and listens for selected messages. Specify MCS_CONSOLE(YES) to enable Thread/SENTRY to automatically detect active log full conditions in one or more monitored DB2 subsystem(s). The default is MCS_CONSOLE(NO).

When Thread/SENTRY detects message DSNJ110E, it quiesces the DB2 batch jobs connected to the DB2 subsystem whose active log is filling up. Thread/SENTRY resumes these batch jobs when the log off-load completes successfully (denoted by message DSNJ139I.)

`MESSAGE_MODULE(module_name)`

The MESSAGE_MODULE operand specifies the name of a load module which contains site written messages that Thread/SENTRY will issue when selected policies are violated. See the discussion of the MSGID operand of the LIMIT policy statement in Section 10.7.2 for further details. Appendix C in this publication describes how to compose site written messages and assemble / link edit them as a load module.

`MSG_DISPLAY(TERSE|VERBOSE|DEBUG)`

The MSG_DISPLAY operand governs which Thread/SENTRY messages should be printed as well as what messages should be suppressed. A specification of TERSE directs Thread/SENTRY to print just essential messages while VERBOSE requests that all messages related to standard processing (including warnings and informational messages) should appear. The DEBUG specification causes all TERSE and VERBOSE messages to be externalized along with additional messages that may be useful for debugging purposes. MSG_DISPLAY(TERSE) is the default.

`MINMAX_INTERVAL(time)`

Specifies the default time duration (in wall clock terms) used to determine the level of thread activity during some observation period. Thread/SENTRY accumulates thread activity during this observation interval and takes action whenever it detects a violation. When this MINMAX interval completes, all activity counters are reset to initialize a new observation period. The categories of activity monitored are requested via LIMIT statement operands prefixed with **IMAX** or **IMIN** (interval maximum and interval minimum). The default MINMAX_INTERVAL defined here applies to any LIMIT statement that does not explicitly specify a MINMAX_INTERVAL. The MINMAX_INTERVAL is ignored for any LIMIT policy that contains neither IMAX nor IMIN operands.

> ***NOTE:*** *The MINMAX_INTERVAL should specify a duration at least* ***double*** *the value of the WAKEUP_INTERVAL. For example, a WAKEUP_INTERVAL of 60 seconds can support MINMAX_ INTERVAL values of 120 seconds or more.*

`MODE(CANCEL|WARN)`

The MODE operand governs whether Thread/SENTRY actually cancels threads for which violations are detected or merely issues a warning. Warn mode can be used to audit all violations but stops short of thread cancellation. When MODE(WARN) is specified, it overrides any LIMIT statements (or other policies) which specify ACTION(CANCEL). The default operational mode is CANCEL.

`MONITOR_PLAN(planname)`

The MONITOR_PLAN operand specifies the name of the Thread/SENTRY application plan. The default plan name in Version 3.1 is TTSM310.

`NONSTD_MODULE(load_module_name)`

Identifies the name of the load module which contains the Thread/SERIES Table of Non-Standard Processing definitions to be used during this run. Each entry within this table defines a thread or set of threads for which non-standard processing is required. Each entry also identifies the exit routine to be called when such a thread is encountered. See Appendix E for details.

`NOTIFY_ENABLED(scope_option,(enable_list))`

The NOTIFY_ENABLED operand identifies the Thread/SENTRY actions for which notification messages should be sent. It also identifies the users and/or administrators who will receive these notification messages.

scope_option specifies the global criterion to be applied to any list of users, administrators and policy actions which appear in any ***enable_list*** which follows.

`ALL` Both administrators and affected users will receive notification messages whenever Thread/SENTRY detects a policy violation.

`NONE` No notification messages will be sent to either administrators or users, regardless of which policy is violated or the action defined for a policy.

`ONLY` Thread/SENTRY will ***only*** send notification messages to the action/recipient pairs whose names explicitly appear in the ***enable_list*** which follows.

`EXCEPT` Thread/SENTRY will send notification messages to both administrators and affected users for all actions ***except*** the action/recipient pairs whose names explicitly appear in the ***enable_list*** which follows

enable_list `(CANCEL_USER,CANCEL_ADMIN,WARNING_USER, WARNING_ADMIN,PENDING_USER,PENDING_ADMIN)`

`CANCEL_USER` Thread/SENTRY should send notification messages to canceled users.

`CANCEL_ADMIN` Thread/SENTRY should send notification messages to the set of administrative IDs associated with a NOTIFY_LIST in the event a thread is canceled.

`WARN_USER` Thread/SENTRY should send notification messages to a user whose thread violates a policy which specifies an action of WARN.

`WARN_ADMIN` Thread/SENTRY should send notification messages to the set of administrative IDs associated with a NOTIFY_LIST in the event a thread violates a policy which specifies an action of WARN.

`PENDING_USER` Thread/SENTRY should send notification messages to users for whom thread cancellation has been pending longer than the PENDING_INTERVAL described below.

`PENDING_ADMIN` Thread/SENTRY should send notification messages to the set of administrative IDs associated with a NOTIFY_LIST in the event a cancellation is pending longer than the PENDING_INTERVAL defined below. This can occur for threads which violate a policy which specifies an action of CANCEL.

NOTIFY_ENABLED(ALL) is the default.

Example 1

Direct Thread/SENTRY to send notification messages to administrators and affected users for all policy violations, whenever Thread/SENTRY detects a policy violation.

```
MONITOR
   NOTIFY_ENABLED(ALL)
```

Example 2

Direct Thread/SENTRY to only send notification messages under the following circumstances: Whenever a thread is canceled, notify both administrators and affected users. When a policy is violated for which an action of WARN is specified, notify just the list of administrator IDs defined (or defaulted) for the violated policy.

```
MONITOR
     NOTIFY_ENABLED(ONLY,(CANCEL_USER,CANCEL_ADMIN,WARN_ADMIN))
```

`NOTIFY_FREQUENCY(integer)`

The integer value specified by the NOTIFY_FREQUENCY operand governs how often *logical* requests to send notifications and/or insert rows into the Thread_Audit table will be *physically* honored. Thread/SENTRY accumulates logical requests until the total is equal to the value specified by NOTIFY_FREQUENCY. At that point, Thread/SENTRY actually sends notifications to users and administrators and inserts a row into the Thread_Audit table. For example, to direct Thread/SENTRY to physically honor one notification request for every 10 requests received, set the NOTIFY_FREQUENCY to 10.

Thread/SENTRY always honors the first request. NOTIFY_FREQUENCY is applied to requests associated with pending actions (which are triggered by the PENDING_INTERVAL operand described below). The default frequency is 10.

`NOTIFY_INACTIVE_ADMIN(BOTH|OFF|WARN|CANCEL)`

The NOTIFY_INACTIVE_ADMIN operand identifies the Thread/SENTRY actions for which notification messages should be sent to the set of administrative IDs associated with a NOTIFY_LIST in the event a thread violates a policy, even to those IDs that are not logged in to z/OS system at the time of violation.

BOTH Thread/SENTRY should send notification messages to any logged off administrative ID in the event a thread is canceled, or a thread violates a policy which specifies an action of WARN.

OFF No notification messages will be sent to any logged off administrator, regardless of which policy is violated or the action defined for a policy.

WARN Thread/SENTRY should send notification messages to any logged off administrative ID in the event a thread violates a policy which specifies an action of WARN.

CANCEL Thread/SENTRY should send notification messages to any logged off administrative ID in the event a thread is canceled.

`NOTIFY_INACTIVE_USER(BOTH|OFF|WARN|CANCEL)`

The NOTIFY_INACTIVE_USER operand identifies the Thread/SENTRY actions for which notification messages should be sent to any user ID whose thread violates a policy, even to those IDs that are not logged in to z/OS system at the time of violation.

BOTH Thread/SENTRY should send notification messages to any logged off user ID whose thread is canceled, or a thread violates a policy which specifies an action of WARN.

OFF No notification messages will be sent to any logged off user ID, regardless of which policy is violated or the action defined for a policy.

WARN Thread/SENTRY should send notification messages to any logged off user ID whose thread violates a policy which specifies an action of WARN.

CANCEL Thread/SENTRY should send notification messages to any logged off user ID whose thread is canceled.

`NOTIFY_LIST(listname)`

The NOTIFY_LIST operand specifies the symbolic name of a list of administrative IDs to be notified when Thread/SENTRY detects a policy violation.

Ordinarily the owner of the violating thread is notified of the action taken by Thread/SENTRY. However, when `listname` starts with the special string `_WTO` (as in `_WTOABC`), Thread/SENTRY writes the user notification message to the operator (via a WTO), rather than send a notification message to the owner of the thread. Notifications sent to administrative users specified in the NOTIFY_LIST are not effected by the `_WTO` special string.

`NOTIFY_MAXIMUM(integer)`

The NOTIFY_MAXIMUM operand limits the number of notifications that Thread/SENTRY will send to users and administrators for a particular thread. Once the specified maximum is reached, Thread/SENTRY *disables* further notifications for that thread.

Only physically honored notification requests increment the count limited by the NOTIFY_MAXIMUM value. The default NOTIFY_MAXIMUM is 3.

`NOTIFY_ON_WARNING(YES|NO)`

The NOTIFY_ON_WARNING operand directs Thread/SENTRY to issue (YES) or suppress (NO) notifications for any pending warning request.

`OUTDD(WTO|ddname)`

The OUTDD operand specifies where Thread/SENTRY should direct its informational, warning and error messages. The report destination can reference the DDname of a file or specify the keyword WTO that denotes Write to the Operator. By default, Thread/SENTRY routes this output to a file named TTSOUT. *This default DDname **should not** be overridden.*

`PENDING_INTERVAL(time)`

The PENDING_INTERVAL operand specifies how long (in seconds) Thread/SENTRY should wait for a thread to terminate normally or abnormally. When the pending interval expires without Thread/SENTRY confirming that the thread no longer exists, Thread/SENTRY logically requests that additional notifications be sent to the affected user and those administrators specified to receive notification messages (as described by the NOTIFY_ENABLED operand). In addition, Thread/SENTRY logically requests the insertion of a row into the Thread_Audit table. Logical requests are honored in accordance with the NOTIFY_FREQUENCY operand described above. These actions reflect the fact that thread termination is pending but not yet complete. The default PENDING_INTERVAL is 60 seconds.

Example: Direct Thread/SENTRY to send notification messages to administrators and affected users if thread cancellation is not confirmed within 30 seconds.

```
MONITOR
   PENDING_INTERVAL(30)
```

`RESET_WARNING(YES|NO)`

The RESET_WARNING operand directs Thread/SENTRY to reset (YES) or maintain (NO) a pending warning request when a target thread no longer violates the policy that triggered the original warning. Both the RESET_WARNING and NOTIFY_ON_WARNING operands must be set to YES in order for Thread/SENTRY to examine pending violations and possibly reset them.

`ROUTECDE(routing_code)`

The ROUTECDE operand applies only if Thread/SENTRY output is directed to the operator via an output specification of OUTPUT(WTO). It denotes an MCS routing code for the WTO macros issued by Thread/SENTRY.

`SNAPDD(ddname)`

The SNAPDD operand specifies where Thread/SENTRY should direct output from the SNAP command. By default, Thread/SENTRY routes this output to a file named TTSTRACE.

`SSID(ssid_list)`

The SSID operand identifies the DB2 subsystem or list of DB2 subsystems to be monitored.

`SUM_PARALLEL_ELAPSED(YES|NO)`

The SUM_PARALLEL_ELAPSED operand governs whether Thread/SENTRY will summarize elapsed wall clock time as well as elapsed time within DB2 for a set of parallel threads. By specifying NO (or accepting the default), neither elapsed wall clock time nor elapsed time within DB2 are summarized. Rather, these two elapsed time values are those from the originating, coordinator thread.

To direct Thread/SENTRY to summarize elapsed wall clock time and elapsed time within DB2 for an originating thread and the 1 to N parallel threads it coordinates, specify SUM_PARALLEL_ELAPSED(YES).

`WAKEUP_INTERVAL(time)`

The WAKEUP_INTERVAL operand specifies the monitoring interval in seconds (i.e. how much time elapses before Thread/SENTRY wakes up to monitor DB2 threads and enforce your policies). The default wakeup_interval is 60 seconds.

Example: Monitor the DB2 subsystems named DB2A, DB2B and DB2C. Thread/SENTRY should 'wake up' every 60 seconds to evaluate thread activity:

```
MONITOR
   SSID(DB2A,DB2B,DB2C)
   WAKEUP_INTERVAL(60)
```

`WTO_HRDCPY(NO|YES)`

The WTO_HRDCPY parameter specifies whether Thread/SENTRY should write WTO messages to hard copy only. The WTO_HRDCPY command option can be specified in the form WTO_HRDCPY(option) where option is either YES or NO. The value YES directs Thread/SENTRY to write WTO messages to hard copy *only* while an option of NO causes WTO messages to be written to the console as well as to hard copy.

10.5 The DEFAULT Statement

The **DEFAULT** statement and its operands let you specify default values for a set of EXCLUDE and LIMIT statements that follow it. These default values remain in effect until ***(1)*** another DEFAULT statement is encountered or ***(2)*** no more Thread/SENTRY statements remain to be processed.

`SSID(name)`

The SSID operand identifies the name of the default DB2 subsystem to be associated with any EXCLUDE or LIMIT statements that follow, which do not specify a subsystem ID explicitly.

`LOGIC(AND|OR)`

The LOGIC operand specifies whether a single threshold or set of thresholds must be violated before Thread/SENTRY performs the action associated with any LIMIT statements that follow, which do not specify a LOGIC operand explicitly.

`AND` The policy action is triggered only when ***all*** thresholds specified by the policy are violated. For example, both threshold 1 and threshold 2 must be exceeded before Thread/SENTRY takes action.

`OR` The policy action is triggered if ***any*** threshold specified by the policy is violated. For example, Thread/SENTRY takes action when either threshold 1 or threshold 2 are exceeded.

`NOTIFY_LIST(listname)`

The NOTIFY_LIST operand specifies the symbolic name of a list of administrative IDs to be notified when Thread/SENTRY detects a policy violation.

Ordinarily, the owner of the violating thread is notified of the action taken by Thread/SENTRY. However, when listname starts with the special string _WTO (as in _WTOABC), Thread/SENTRY writes the user notification message to the operator (via a WTO), rather than send a notification message to the owner of the thread. Notifications sent to administrative users specified in the NOTIFY_LIST are not effected by the `_WTO` special string.

`START(time)`

The START operand specifies the default starting time for the application of a policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY evaluates the policy regardless of the time of day.

`END(time)`

The END operand specifies the default ending time for the application of a policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY evaluates the policy regardless of the time of day.

`DAYS(1,2,3,4,5,6,7,ALL,WEEKDAYS)`

The DAYS operand specifies the default day(s) of the week when a policy is effective. If a DAYS value is neither defaulted nor explicitly specified, Thread/SENTRY evaluates the policy regardless of the day of the week.

`MINMAX_INTERVAL(time)`

Specifies the default time duration (in wall clock terms) used to determine the level of thread activity during some observation period. The monitor accumulates thread activity during this observation interval and takes action whenever it detects a violation. When this MINMAX interval completes, all activity counters are reset to initialize a new observation period. The categories of activity monitored are requested via LIMIT statement operands prefixed with **IMAX** or **IMIN** (interval maximum and interval minimum). The default MINMAX_INTERVAL defined here applies to any LIMIT statement that does not explicitly specify a MINMAX_INTERVAL. The MINMAX_INTERVAL is ignored for any LIMIT policy that contains neither IMAX nor IMIN operands.

> ***NOTE:*** *The MINMAX_INTERVAL should specify a duration at least* ***double*** *the value of the WAKEUP_INTERVAL. The default WAKEUP_INTERVAL is set to 60 seconds (see the MONITOR statement above) which supports MINMAX_INTERVAL values of 120 seconds or more.*

10.6 The EXCLUDE Statement

The **EXCLUDE** statement defines threads that are not subject to evaluation and cancellation by Thread/SENTRY. The following operands of the EXCLUDE statement identify the thread(s) to which an exclusion policy applies.

> ***NOTE:*** *Values for the operands identified as maskable can specify the SQL wild card characters % and _ for pattern matching purposes. For example CONNECTION(CICS%)*

`POLICY_ID(pid)`

The POLICY_ID operand specifies a symbolic name (of one to eight characters in length) for this exclusion policy. This operand ***must*** be specified.

10.6.1 What threads are subject to EXCLUDE policies

`SSID(ssids)`

The SSID operand specifies the name of one or more DB2 subsystems in which this exclusion policy is to be effective. The list of subsystem names should be separated by commas.

`PLAN(name)`

The PLAN operand identifies the names of the DB2 application plans that should be ignored by Thread/SENTRY (provided all other WHAT and WHEN criteria are meet). PLAN name is maskable.

`PROGRAM(pgmname)`

The PROGRAM operand denotes a field of up to 8 bytes long whose significant characters identify the name of a currently executing DB2 package or DBRM. Thread/SENTRY should ignore such a program (provided all other WHAT and WHEN criteria are meet). A Class 7 trace must be continuously active in order for Thread/SENTRY to exclude threads on the basis of the currently active package or DBRM name. The PROGRAM value is maskable.

`AUTHID(name)`

The AUTHID operand specifies the primary authorization IDs of thread(s) that Thread/SENTRY should ignore (provided all other WHAT and WHEN criteria are meet). AUTHID name is maskable.

`JOBNAME(jobname)`

The JOBNAME operand denotes the names of the jobs associated with the allied address spaces from which one or more thread(s) originate. Thread/ SENTRY ignores any threads that originate in an address space that is subject to exclusion (provided all other WHAT and WHEN criteria are meet). The JOBNAME value is maskable.

`CORRELATION(name)`

The CORRELATION operand denotes a field up to 12 bytes long whose significant characters identify the Correlation ID of the thread(s) to be ignored by Thread/SENTRY (provided all other WHAT and WHEN criteria are meet). The CORRELATION value is maskable.

CONNECTION(name)

The CONNECTION operand denotes a field up to 8 bytes long whose significant characters specify the connection name associated with a thread. Thread/SENTRY ignores any threads which match this connection name (provided all other WHAT and WHEN criteria are meet). CONNECTION is a maskable value.

CONNECTION_TYPE(type)

The CONNECTION_TYPE operand denotes a field up to 8 bytes long whose significant characters specify a thread's *connection type*. Thread/SENTRY ignores any threads which match this connection type (provided all other WHAT and WHEN criteria are meet). CONNECTION_TYPE is ***not*** a maskable value. Rather, CONNECTION_TYPE must specify one of the values in the left hand column below (derived from the mapping macro DSNDQWHC supplied with DB2). Each QWHCxxxx value denotes the connection type described in the corresponding right hand column:

QWHCTSO	TSO foreground and background
QWHCDB2C	DB2 call attach
QWHCDLIB	DL/I batch
QWHCCICS	CICS attach
QWHCIMSB	IMS attach BMP
QWHCIMSM	IMS attach MPP
QWHCDUW	System directed access
QWHCRUW	Application directed access
QWHCICTL	IMS control region
QWHCTBMP	IMS transaction BMP
QWHCUTIL	DB2 Utilities
QWHCTRRS	RRSAF Attach
TSOFORE	TSO foreground
TSOBATCH	TSO batch

10.6.2 What threads are subject to EXCLUDE policies on the basis of MVS and DDF Accounting Data Associated with the Thread

NOTE: *The following operands are applicable to all threads for which MVS and DDF accounting information is present.*

`PRODUCT_NAME(name)`

The PRODUCT_NAME operand identifies the product that generated the accounting string. Thread/SENTRY ignores any threads which match this product name value (provided all other WHAT and WHEN criteria are meet). The product identifier may assume one the following three character values:

DSN	denotes DB2 for z/OS
ARI	denotes SQL/DS or DB2 for VM
SQL	denotes DB2 client/server
QSQ	denotes DB2/400

`PRODUCT_VERSION(vv)`

The PRODUCT_VERSION operand is a two character value which identifies the version of the product that generated the accounting string. Thread/SENTRY ignores any threads which match this version value (provided all other WHAT and WHEN criteria are meet).

`PRODUCT_RELEASE(rr)`

The PRODUCT_RELEASE operand is a two character value which identifies the release level of the product that generated the accounting string. Thread/SENTRY ignores any threads which match this release value (provided all other WHAT and WHEN criteria are meet).

`PRODUCT_MOD(m)`

The PRODUCT_MOD operand is a single character value which identifies the modification level of the product that generated the accounting string. Thread/SENTRY ignores any threads which match this modification level value (provided all other WHAT and WHEN criteria are meet).

NOTE: The following operands refer to MVS and DDF accounting information for those threads whose PRODUCT_NAME value is 'DSN'. That is, threads whose accounting strings are generated by either DB2 for OS/390 or DB2 for MVS/ESA.

`QMDA_LOCATION(name)`

The QMDA_LOCATION operand identifies the DB2 location name for the DB2 system that created the accounting string. A name of up to 16 characters in length may be specified. Thread/SENTRY ignores any threads which match this *maskable* location name value (provided all other WHAT and WHEN criteria are meet).

`QMDA_NETID(name)`

The QMDA_NETID operand identifies the SNA NETID of the DB2 system that created the accounting string. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this *maskable* SNA Net ID value (provided all other WHAT and WHEN criteria are meet).

`QMDA_LUNAME(name)`

The QMDA_LUNAME operand identifies the SNA LU name of the DB2 system that created the accounting string. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this LU name (provided all other WHAT and WHEN criteria are meet). QMDA_LUNAME is a maskable value.

`QMDA_CONNECTION(name)`

The QMDA_CONNECTION operand identifies the DB2 Connection Name at the DB2 system where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this *maskable* QMDA_CONNECTION value (provided all other WHAT and WHEN criteria are meet).

`QMDA_CONNECTION_TYPE(name)`

The QMDA_CONNECTION_TYPE operand identifies the DB2 Connection Type at the DB2 system where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this QMDA_CONNECTION_TYPE (provided all other WHAT and WHEN criteria are meet). This operand accepts a maskable value.

`QMDA_CORRELATION(name)`

The QMDA_CORRELATION operand identifies the DB2 Correlation ID at the DB2 system where the SQL application is running. A name of up to 12 characters in length may be specified. Thread/SENTRY ignores any threads which match this correlation name (provided all other WHAT and WHEN criteria are meet). QMDA_CORRELATION is a maskable value.

`QMDA_AUTHID(name)`

The QMDA_AUTHID operand identifies the DB2 authorization ID that the SQL application used, prior to name translation and prior to driving the connection exit at the DB2 site where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this Authorization ID (provided all other WHAT and WHEN criteria are meet). QMDA_AUTHID is a maskable value.

`QMDA_PLAN(name)`

The QMDA_PLAN operand identifies the DB2 PLAN that the SQL application used at the DB2 site running the SQL application. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this plan name (provided all other WHAT and WHEN criteria are meet). QMDA_PLAN is a maskable value.

NOTE: The following operands refer to MVS and DDF accounting information for those threads whose PRODUCT_NAME value is 'SQL'. That is, threads whose accounting strings are created by DB2 client server products such as DB2 for Windows NT, DB2 for OS/2 and DB2 for various Unix implementations**.**

`CLIENT_PLATFORM(name)`

The CLIENT_PLATFORM operand identifies the Client Platform where the SQL application is running. A name of up to 18 characters in length may be specified. Some example values are 'NT', 'AIX' and 'OS/2'. Thread/SENTRY ignores any threads which match this platform name (provided all other WHAT and WHEN criteria are meet). CLIENT_PLATFORM is a maskable value.

`CLIENT_APPLICATION(name)`

The CLIENT_APPLICATION operand identifies the name of the client SQL application. A name of up to 20 characters in length (such as 'PAYROLL') may be specified. Thread/SENTRY ignores any threads which match this client application name (provided all other WHAT and WHEN criteria are meet). CLIENT_APPLICATION is a maskable value.

`CLIENT_AUTHID(name)`

The CLIENT_AUTHID operand identifies the authorization ID of the client application process. A name of up to 8 characters in length may be specified. Thread/SENTRY ignores any threads which match this client Authorization ID (provided all other WHAT and WHEN criteria are meet). CLIENT_AUTHID is a maskable value.

NOTE: *The following operands apply to <u>all</u> threads for which accounting strings are available. See Section 10.7.9.1 for examples of LIMIT policies which make use of these accounting string operands.*

`ACCOUNT_SUBSTRING1(pattern string)`

The ACCOUNT_SUBSTRING1 operand specifies a pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY ignores threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNTG_SUBSTRING1 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START1(integer)`

ACCOUNT_START1 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING1 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH1(integer)`

The ACCOUNT_LENGTH1 operand specifies an explicit length for the comparison between the pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING1 is implicitly derived in a manner consistent with the SQL LIKE predicate as follows:

- The underscore sign (_) represents a single arbitrary character
- The percent sign (%) represents a string of zero or more arbitrary characters
- Any other character represents a single occurrence of itself

`ACCOUNT_SUBSTRING2(pattern string)`

The ACCOUNT_SUBSTRING2 operand specifies a second pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY ignores any threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNT_SUBSTRING2 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START2(integer)`

ACCOUNT_START2 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING2 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH2(integer)`

The ACCOUNT_LENGTH2 operand specifies an explicit length for the comparison between the second pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING2 is implicitly derived as described in the documentation for the ACCOUNT_LENGTH1 operand.

`ACCOUNT_SUBSTRING3(pattern string)`

The ACCOUNT_SUBSTRING3 operand specifies a third pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY ignores any threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNT_SUBSTRING3 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START3(integer)`

ACCOUNT_START3 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING3 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH3(integer)`

The ACCOUNT_LENGTH3 operand specifies an explicit length for the comparison between the third pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING3 is implicitly derived as described in the documentation for the ACCOUNT_LENGTH1 operand.

10.6.3 When to apply Exclusion Policies

The following operands specify when to apply an exclusion policy.

`START(time)`

The START operand specifies the starting time for the application of this exclusion policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY applies this exclusion policy regardless of the time of day.

`END(time)`

The END operand specifies an explicit ending time for the application of this policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY evaluates this policy regardless of the time of day.

`DAYS(1,2,3,4,5,6,7,ALL,WEEKDAYS)`

The DAYS operand specifies the day(s) of the week for which this exclusion policy is effective. If a DAYS value is neither defaulted nor explicitly specified, Thread/SENTRY evaluates this policy regardless of the day of the week.

10.7 The LIMIT Statement

The **LIMIT** statement enables you to specify resource consumption thresholds for thread activity. You can define various minimum and maximum levels that apply over

(1) the life a of thread,
(2) the period encompassing a single unit-of-recovery; or
(3) some period of elapsed time (termed a MINMAX interval).

Thread activity includes such measures as elapsed time, CPU time, number of SQL statements executed, etc. When a maximum threshold is exceeded (or a minimum threshold is not reached), Thread/SENTRY performs the action defined on the LIMIT statement. The ACTION operand of the LIMIT statement can direct Thread/SENTRY to issue a warning, cancel the thread or call a site-written or vendor supplied exit routine to decide what to do. Operands of the LIMIT statement identify which limits are to be evaluated and enforced.

Parallel Thread Support

When Thread/SENTRY detects a set of parallel threads (i.e. an *originating* thread and the 1 - n *parallel* threads it coordinates), Thread/SENTRY creates a fictitious *composite* thread that contains summarized statistics for the originating thread and its associated parallel threads. Thread/SENTRY automatically applies LIMIT policies to the summary statistics maintained in this *composite* thread. The identity of the composite thread is the same as that of the originating, coordinator thread with respect to values such as plan name, authid, etc.

In the following example, the LIMIT policy name POLICY1 will cancel a thread whose plan name is PAYROLL in either of the following cases: 1 - PAYROLL runs as a non parallel thread and issues more than 10,000 GETPAGE requests or 2 - PAYROLL runs in parallel and the sum of the GETPAGE requests issued by the originating thread and the parallel threads it coordinates exceed 10,000.

```
LIMIT
   POLICY_ID(POLICY1)
   PLAN(PAYROLL)          <- same plan name for originator and composite
   MAX_GETPAGES(10000)    <- maximum GETPAGES for coordinator + parallel
```

NOTE: By default, elapsed wall clock time for the application as well as elapsed time within DB2 are **not** summarized. Rather, these two elapsed time values are those from the originating, coordinator thread. To summarize elapsed wall clock time and elapsed time within DB2, you can specify SUM_PARALLEL_ELAPSED(YES) as an operand of the MONITOR statement as described in Section 10.4.

10.7.1 What threads are subject to LIMIT policies

The following operands of the LIMIT statement identify the thread(s) to which the policy applies. If these qualifiers are omitted, Thread/SENTRY will apply this LIMIT policy to all threads *except* those explicitly excluded through EXCLUDE policy statements.

> ***NOTE:*** *Values for the operands identified as maskable can specify the SQL wild card characters % and _ for pattern matching purposes. For example CONNECTION(CICS%)*

`POLICY_ID(pid)`

The POLICY_ID operand specifies a symbolic name (of one to eight characters in length) for this LIMIT policy. This operand ***must*** be specified.

`SSID(ssids)`

The SSID operand specifies the name of one or more DB2 subsystems in which this LIMIT policy is to be effective.

`PLAN(name)`

The PLAN operand identifies the names of the DB2 application plan(s) to which this LIMIT policy is applied (provided all other WHAT and WHEN criteria are meet). PLAN name is maskable.

`PROGRAM(pgmname)`

The PROGRAM operand identifies the name of a currently executing DB2 package or DBRM to which this LIMIT policy is applied (provided all other WHAT and WHEN criteria are meet). A Class 7 trace must be continuously active in order for Thread/SENTRY to select threads on the basis of program name. PROGRAM name is maskable.

`AUTHID(name)`

The AUTHID operand specifies the primary authorization IDs of thread(s) to which this LIMIT policy will be applied (provided all other WHAT and WHEN criteria are meet). AUTHID is maskable.

`JOBNAME(jobname)`

The JOBNAME operand denotes the names of the jobs associated with the allied address spaces from which one or more thread(s) originate. Thread/ SENTRY will apply this LIMIT policy to any thread that originates in an address space whose jobname matches the jobname pattern (provided all other WHAT and WHEN criteria are meet). The JOBNAME value is maskable.

`CORRELATION(name)`

The CORRELATION operand denotes a field up to 12 bytes in length whose significant characters identify the Correlation ID of the thread(s) to which this LIMIT policy will be applied (provided all other WHAT and WHEN criteria are meet). The CORRELATION value is maskable.

`CONNECTION(name)`

The CONNECTION operand denotes a field up to 8 bytes long whose significant characters specify the connection name associated with a thread. Thread/SENTRY will apply this LIMIT policy to any threads which match this connection name (provided all other WHAT and WHEN criteria are meet). CONNECTION is a maskable value.

`CONNECTION_TYPE(type)`

The CONNECTION_TYPE operand denotes a field up to 8 bytes long whose significant characters specify a thread's *connection type*. Thread/SENTRY will apply this LIMIT policy to any threads which match this connection type (provided all other WHAT and WHEN criteria are meet). CONNECTION_TYPE is ***not*** a maskable value. Rather, it must specify one of the values in the left hand column below (derived from the mapping macro DSNDQWHC supplied with DB2). Each QWHCxxxx value denotes the connection type described in the corresponding right hand column:

`QWHCTSO`	TSO foreground and background
`QWHCDB2C`	DB2 call attach
`QWHCDLIB`	DL/I batch
`QWHCCICS`	CICS attach
`QWHCIMSB`	IMS attach BMP
`QWHCIMSM`	IMS attach MPP
`QWHCDUW`	System directed access
`QWHCRUW`	Application directed access
`QWHCICTL`	IMS control region
`QWHCTBMP`	IMS transaction BMP
`QWHCUTIL`	DB2 Utitlities
`QWHCTRRS`	RRSAF Attach
`TSOFORE`	TSO foreground
`TSOBATCH`	TSO batch

10.7.1.1 What threads are subject to LIMIT policies on the basis of MVS and DDF Accounting Data Associated with the Thread

NOTE: The following operands are applicable to all threads for which MVS and DDF accounting information is present.

`PRODUCT_NAME(name)`

The PRODUCT_NAME operand identifies the product that generated the accounting string. Thread/SENTRY will apply this LIMIT policy to any threads which match this product name value (provided all other WHAT and WHEN criteria are meet). The product identifier may assume one the following three character values:

DSN denotes DB2 for z/OS
ARI denotes SQL/DS or DB2 for VM
SQL denotes DB2 client/server
QSQ denotes DB2/400

`PRODUCT_VERSION(vv)`

The PRODUCT_VERSION operand is a two character value which identifies the version of the product that generated the accounting string. Thread/SENTRY will apply this LIMIT policy to any threads which match this version value (provided all other WHAT and WHEN criteria are meet).

`PRODUCT_RELEASE(rr)`

The PRODUCT_RELEASE operand is a two character value which identifies the release level of the product that generated the accounting string. Thread/SENTRY will apply this LIMIT policy to any threads which match this release value (provided all other WHAT and WHEN criteria are meet).

`PRODUCT_MOD(m)`

The PRODUCT_MOD operand is a single character value which identifies the modification level of the product that generated the accounting string. Thread/SENTRY will apply this LIMIT policy to any threads which match this modification level value (provided all other WHAT and WHEN criteria are meet).

NOTE: The following operands refer to MVS and DDF accounting information for those threads whose PRODUCT_NAME value is 'DSN'. That is, threads whose accounting strings are generated by either DB2 for OS/390 or DB2 for MVS/ESA**.**

`QMDA_LOCATION(name)`

The QMDA_LOCATION operand identifies the DB2 location name for the DB2 system that created the accounting string. A name of up to 16 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this *maskable* location name value (provided all other WHAT and WHEN criteria are meet).

`QMDA_NETID(name)`

The QMDA_NETID operand identifies the SNA NETID of the DB2 system that created the accounting string. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this *maskable* SNA Net ID value (provided all other WHAT and WHEN criteria are meet).

`QMDA_LUNAME(name)`

The QMDA_LUNAME operand identifies the SNA LU name of the DB2 system that created the accounting string. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this LU name (provided all other WHAT and WHEN criteria are meet). QMDA_LUNAME is a maskable value.

`QMDA_CONNECTION(name)`

The QMDA_CONNECTION operand identifies the DB2 Connection Name at the DB2 system where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this *maskable* QMDA_CONNECTION value (provided all other WHAT and WHEN criteria are meet).

`QMDA_CONNECTION_TYPE(name)`

The QMDA_CONNECTION_TYPE operand identifies the DB2 Connection Type at the DB2 system where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this QMDA_CONNECTION_TYPE (provided all other WHAT and WHEN criteria are meet). This operand accepts a maskable value.

`QMDA_CORRELATION(name)`

The QMDA_CORRELATION operand identifies the DB2 Correlation ID at the DB2 system where the SQL application is running. A name of up to 12 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this correlation name (provided all other WHAT and WHEN criteria are meet). QMDA_CORRELATION is a maskable value.

`QMDA_AUTHID(name)`

The QMDA_AUTHID operand identifies the DB2 authorization ID that the SQL application used, prior to name translation and prior to driving the connection exit at the DB2 site where the SQL application is running. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this Authorization ID (provided all other WHAT and WHEN criteria are meet). QMDA_AUTHID is a maskable value.

`QMDA_PLAN(name)`

The QMDA_PLAN operand identifies the DB2 PLAN that the SQL application used at the DB2 site running the SQL application. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this plan name (provided all other WHAT and WHEN criteria are meet). QMDA_PLAN is a maskable value.

NOTE: The following operands refer to MVS and DDF accounting information for those threads whose PRODUCT_NAME value is 'SQL'. That is, threads whose accounting strings are created by DB2 client server products such as DB2 for Windows NT, DB2 for OS/2 and DB2 for various Unix implementations.

`CLIENT_PLATFORM(name)`

The CLIENT_PLATFORM operand identifies the Client Platform where the SQL application is running. A name of up to 18 characters in length may be specified. Some example values are 'NT', 'AIX' and 'OS/2'. Thread/SENTRY will apply this LIMIT policy to any threads which match this platform name (provided all other WHAT and WHEN criteria are meet). CLIENT_PLATFORM is a maskable value.

`CLIENT_APPLICATION(name)`

The CLIENT_APPLICATION operand identifies the name of the client SQL application. A name of up to 20 characters in length (such as 'PAYROLL') may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this client application name (provided all other WHAT and WHEN criteria are meet). CLIENT_APPLICATION is a maskable value.

`CLIENT_AUTHID(name)`

The CLIENT_AUTHID operand identifies the authorization ID of the client application process. A name of up to 8 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads which match this client Authorization ID (provided all other WHAT and WHEN criteria are meet). CLIENT_AUTHID is a maskable value.

NOTE: The following operands apply to all threads for which accounting strings are available. See Section 10.7.9.1 for examples of LIMIT policies which make use of these accounting string operands.

`ACCOUNT_SUBSTRING1(pattern string)`

The ACCOUNT_SUBSTRING1 operand specifies a pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNT_SUBSTRING1 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START1(integer)`

ACCOUNT_START1 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING1 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH1(integer)`

The ACCOUNT_LENGTH1 operand specifies an explicit length for the comparison between the pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING1 is implicitly derived in a manner consistent with the SQL LIKE predicate as follows:

- The underscore sign (_) represents a single arbitrary character
- The percent sign (%) represents a string of zero or more arbitrary characters
- Any other character represents a single occurrence of itself

`ACCOUNT_SUBSTRING2(pattern string)`

The ACCOUNT_SUBSTRING2 operand specifies a second pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNT_SUBSTRING2 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START2(integer)`

ACCOUNT_START2 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING2 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH2(integer)`

The ACCOUNT_LENGTH2 operand specifies an explicit length for the comparison between the second pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING2 is implicitly derived as described in the documentation for the ACCOUNT_LENGTH1 operand.

`ACCOUNT_SUBSTRING3(pattern string)`

The ACCOUNT_SUBSTRING3 operand specifies a third pattern substring that may occur within the accounting string associated with a thread. A maskable pattern of up to 200 characters in length may be specified. Thread/SENTRY will apply this LIMIT policy to any threads whose accounting string (or portion thereof) matches the pattern specified by ACCOUNT_SUBSTRING3 (provided all other WHAT and WHEN criteria are meet).

`ACCOUNT_START3(integer)`

ACCOUNT_START3 specifies the starting character position within the thread's accounting string where pattern matching with the value specified by ACCOUNT_SUBSTRING3 should begin. If omitted, the default starting position is 1.

`ACCOUNT_LENGTH3(integer)`

The ACCOUNT_LENGTH3 operand specifies an explicit length for the comparison between the third pattern string and the accounting substring. If omitted, the length of the pattern specified by ACCOUNT_SUBSTRING3 is implicitly derived as described in the documentation for the ACCOUNT_LENGTH1 operand.

10.7.2 Other Operands of the LIMIT Statement

```
ACTION(WARN|CANCEL|DUMP|FORCE|KILL|QUIESCE|RESUME|VTAM|PGM(program))
```

The ACTION operand specifies what Thread/SENTRY should do if this LIMIT policy is violated. Thread/SENTRY supports the following actions:

`CANCEL`	the thread
`DUMP`	cancel the thread with a dump
`FORCE`	the thread to terminate. This is the most drastic action and should be used very judiciously.
`KILL`	FORCE PURGE the CICS transaction (applicable to CICS transactions only)
`PGM`	invoke a vendor supplied or installation-written exit program and pass it information about the thread for which a violation was detected. See Appendix E for a discussion of Exit routines.
`QUIESCE`	the MVS address space associated with the thread. The thread address space effectively becomes non-dispatchable with the lowest possible performance characteristics.
`RESUME`	the MVS address space associated with the thread. The thread address space resumes execution with its original (pre-quiesced) performance characteristics.
`WARN`	issue a warning. This is the default.

The ACTION operand of the LIMIT statement accepts multiple actions, separated by comas. Thread/SENTRY performs one action per wakeup interval so long as the problem DB2 thread persists. When specified, such action escalates from `WARNING`, to `CANCEL`, to `FORCE` and lastly to `KILL`.

Examples:

`ACTION(WARNING,FORCE)`	Thread/SENTRY will issue warnings up to the number of notifications limited by the NOTIFY_FREQUENCY and NOTIFY_MAXIMUM operands. Then, if the target thread still violates the current policy, Thread/SENTRY will FORCE the thread to terminate.
`ACTION(WARNING,CANCEL,FORCE)`	Thread/SENTRY will issue warnings up to the number of notifications limited by the NOTIFY_FREQUENCY and NOTIFY_MAXIMUM operands. Then, if the target thread still violates the current policy, Thread/SENTRY will CANCEL the thread using the DB2 -CANCEL THREAD command. If the thread still exists at the next wakeup interval, Thread/SENTRY will FORCE the thread to terminate.

ADD_ZIIP_DATA(NO|YES)

The ADD_ZIIP_DATA operand specifies whether the CPU time consumed on an IBM specialty engine should be included in CLASS1 and CLASS2 CPU times thresholds calculations.

AUDIT(YES|NO)

The AUDIT operand specifies whether the event associated with the ACTION operand should be recorded in the THREAD_AUDIT table. The default value is YES.

LOGIC(AND|OR)

The LOGIC operand specifies whether a single threshold or set of thresholds must be violated before Thread/SENTRY performs the action defined by the LIMIT policy. The default is OR.

AND — The policy action is triggered only when ***all*** thresholds specified by the LIMIT policy are violated. For example, both threshold 1 and threshold 2 must be exceeded before Thread/SENTRY takes action.

OR — The policy action is triggered if ***any*** threshold specified by the LIMIT policy is violated. For example, Thread/SENTRY takes action when either threshold 1 or threshold 2 are exceeded.

MSGID(message_ID)

The MSGID operand specifies the name of a site written message that Thread/SENTRY should issue when this policy is violated. The site written message must be link-edited within the load module specified via the MESSAGE_MODULE operand of the MONITOR statement as described in Section 10.4. Appendix C in this publication describes how to compose site written messages and assemble / link edit them as a load module.

MINMAX_INTERVAL(time)

Specifies the duration (in wall clock terms) used to determine the level of thread activity during some observation period. The monitor accumulates thread activity during this observation interval and takes action whenever it detects a violation. When this MINMAX interval completes, all activity counters are reset to initialize a new observation period. The categories of activity monitored are requested via LIMIT statement operands prefixed with **IMAX** or **IMIN** (interval maximum and interval minimum). The MINMAX_INTERVAL is ignored for any LIMIT policy that contains neither IMAX nor IMIN operands.

NOTE: The MINMAX_INTERVAL should specify a duration at least **double** the value of the WAKEUP_INTERVAL. The default WAKEUP_INTERVAL is set to 60 seconds (see the MONITOR statement above) which supports MINMAX_INTERVAL values of 120 seconds or more.

`NOTIFY_LIST(listname)`

The NOTIFY_LIST operand specifies the symbolic name of a list of administrative IDs to be notified when Thread/SENTRY detects a policy violation.

Ordinarily, the owner of the violating thread is notified of the action taken by Thread/SENTRY. However, when `listname` starts with the special string `_WTO` (as in `_WTOABC`), Thread/SENTRY writes the user notification message to the operator (via a WTO), *rather than* send a notification message to the owner of the thread. Notifications sent to administrative users specified in the NOTIFY_LIST are not effected by the `_WTO` special string.

10.7.3 When to apply LIMIT policies

The following operands specify ***when*** to apply this LIMIT policy:

`START(time)`

The START operand specifies the starting time for the application of this LIMIT policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY applies this LIMIT policy regardless of the time of day.

`END(time)`

The END operand specifies an explicit ending time for the application of this policy. If starting and ending times are neither defaulted nor explicitly specified, Thread/SENTRY evaluates this policy regardless of the time of day.

`DAYS(1,2,3,4,5,6,7,ALL,WEEKDAYS)`

The DAYS operand specifies the day(s) of the week when this exclusion policy is effective. If a DAYS value is neither defaulted nor explicitly specified, Thread/SENTRY evaluates this policy regardless of the day of the week.

10.7.4 Life of Thread Limits

The following operands define limits for ***the life of a thread***. Each LIMIT policy can define a single threshold measurement such as MAX_ELAPSED(time) or specify several operands and their limit values. ***You do not have to specify every operand for each policy.*** Once thresholds are exceeded for a given policy, Thread/SENTRY performs the action defined by the ACTION operand. All *life-of-thread* related operands are prefixed by the character string '**MAX**'.

`MAX_BUFFER_POOL_ID(integer)`

Specifies a DB2 buffer pool ID. Thread/SENTRY LIMIT policies let you limit the number of GETPAGE requests any thread can perform against a particular DB2 buffer pool.

`MAX_ELAPSED(time)`

Specifies maximum elapsed time for the thread in wall clock terms.

`MAX_CPU(time)`

Specifies the maximum CPU time the thread may consume, both within DB2 as well as within the application program. IBM documentation sometimes refers to this as Class 1 TCB CPU time.

`MAX_SRB(time)`

Specifies the maximum SRB time that can be charged to the allied address space associated with a thread.

`MAX_DB2_ELAPSED(time)` *Requires Class 2 Accounting trace*

Specifies the maximum wall clock time that may elapse within DB2 during the execution of a thread.

`MAX_DB2_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the maximum CPU time that may be consumed by a thread while it processes within DB2. IBM documentation sometimes refers to this as Class 2 TCB CPU time.

`MAX_IO_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum elapsed time spent waiting to perform synchronous I/O under this thread.

`MAX_LOCK_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum elapsed time spent waiting for DB2 locks and latches.

`MAX_GETPAGES(integer)`

Specifies the maximum number of getpage requests the thread can issue during its lifetime. Both successful and unsuccessful requests of an *unconditional* nature as well as successful *conditional* requests count towards this limit. The getpage count is an excellent raw measure of thread activity.

`MAX_SQL(integer)`

Specifies the maximum number of SQL data manipulation language (DML) statements of all types a thread can execute during its lifetime.

`MAX_QUERY(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during its lifetime. Thread/SENTRY counts SELECT, OPEN, FETCH and CLOSE statements towards this limit.

`MAX_FETCH(integer)`

Specifies the maximum number of SQL FETCH statements a thread can execute during its lifetime.

`MAX_CHANGES(integer)`

Specifies the maximum number of SQL statements a thread can execute during its lifetime that result in changes to data. Thread/SENTRY counts DELETE, INSERT and UPDATE statements towards this limit.

`MAX_QUERIED(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during its lifetime. Thread/SENTRY counts SELECT, OPEN, multiple-rows FETCH and CLOSE statements towards this limit.

`MAX_FETCHED(integer)`

Specifies the maximum number of SQL multiple-row FETCH statements a thread can execute during its lifetime. Thread/SENTRY uses the QXRWSFETCHD field of the DB2 DSNDQXST structure to compare the number of multiple-rows fetched against the MAX_FETCHED specified limit.

`MAX_CHANGED(integer)`

Specifies the maximum number of SQL statements a thread can execute during its lifetime that result in changes to data. Thread/SENTRY counts the number of multiple-rows inserted, updated and deleted towards this limit. Thread/SENTRY uses the QXRWSINSRTD, QXRWSUPDTD and QXRWSDELETD fields of the DB2 DSNDQXST structure to compare the number of rows inserted, updated and deleted respectively, against the MAX_CHANGED specified limit.

`MAX_DEADLOCKS(integer)`

Specifies the maximum number of deadlocks detected during the life of a thread.

`MAX_SUSPENDS(integer)`

Specifies the maximum number of times during its life a thread is suspended due to lock conflicts. This count is incremented every time DB2 cannot obtain a lock and has to suspend the thread's unit-of-recovery.

`MAX_TIMEOUTS(integer)`

Specifies the maximum number of times during its life a thread is suspended for a duration that exceeds the timeout value defined for the DB2 subsystem in which the thread executes.

`MAX_ESCALATIONS_SHARED(integer)`

Specifies the maximum number of times during its life that DB2 escalates a thread's locks to shared mode.

`MAX_ESCALATIONS_EXCLUSIVE(integer)`

Specifies the maximum number of times during its life that DB2 escalates a thread's locks to exclusive mode.

`MAX_PAGE_LOCKS(integer)`

Specifies the maximum number of page locks a thread can hold concurrently during its execution.

10.7.5 Unit-of-Recovery LIMITS

The following operands define limits for a thread during a ***single unit-of-recovery***. Once this unit-of-work threshold is exceeded, Thread/SENTRY performs the action defined by the ACTION operand. Note that Unit-of-recovery limits are considerably more expensive to monitor and enforce than the cumulative life-of-thread limits defined by the MAX_ operands described above. All unit-of-recovery related operands are prefixed by the character string '**UOW**' denoting unit-of-work.

`UOW_ELAPSED(time)`

Specifies the maximum elapsed time for the thread during a single unit-of-recovery.

`UOW_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the maximum CPU time a thread can consume, both within DB2 as well as in the application program, during a single unit-of-recovery.

`UOW_SRB(time)`

Specifies the maximum SRB time that can be charged to the allied address space associated with a thread during a single unit-of-recovery.

`UOW_DB2_ELAPSED(time)` *Requires Class 2 Accounting trace*

Specifies the maximum wall clock time that may elapse within DB2 during a single unit-of-recovery.

`UOW_DB2_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the maximum CPU time a thread may consume within DB2 during a single unit-of-recovery.

`UOW_IO_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum duration a thread may wait to perform synchronous I/O during a single unit-of-recovery.

`UOW_LOCK_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum duration a thread may wait to acquire locks and latches during a single unit-of-recovery.

`UOW_GETPAGES(integer)`

Specifies the maximum number of getpage requests a thread can issue during a single unit-of-recovery.

`UOW_SQL(integer)`

Specifies the maximum number of SQL data manipulation language (DML) statements of all types a thread can execute during a single unit-of-recovery.

`UOW_QUERY(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during a single unit-of-recovery. Thread/SENTRY counts SELECT, OPEN, FETCH and CLOSE statements towards this limit.

`UOW_FETCH(integer)`

Specifies the maximum number of SQL FETCH statements a thread can execute during a single unit-of-recovery.

`UOW_CHANGES(integer)`

Specifies the maximum number of SQL statements a thread can execute during a single unit-of-recovery that result in changes to data. Thread/SENTRY counts DELETE, INSERT and UPDATE statements towards this limit.

`UOW_QUERIED(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during a single unit-of-recovery. Thread/SENTRY counts SELECT, OPEN, multiple-rows FETCH and CLOSE statements towards this limit.

`UOW_FETCHED(integer)`

Specifies the maximum number of SQL multiple-row FETCH statements a thread can execute during a single unit-of-recovery. Thread/SENTRY uses the QXRWSFETCHD field of the DB2 DSNDQXST structure to compare the number of multiple-rows fetched against the UOW_FETCHED specified limit.

`UOW_CHANGED(integer)`

Specifies the maximum number of SQL statements a thread can execute during a single unit-of-recovery that result in changes to data. Thread/SENTRY counts the number of multiple-rows inserted, updated and deleted towards this limit. Thread/SENTRY uses the QXRWSINSRTD, QXRWSUPDTD and QXRWSDELETD fields of the DB2 DSNDQXST structure to compare the number of rows inserted, updated and deleted respectively, against the UOW_CHANGED specified limit

`UOW_DEADLOCKS(integer)`

Specifies the maximum number of deadlocks detected during a single unit-of-recovery.

`UOW_SUSPENDS(integer)`

Specifies the maximum number of times (during a single unit-of-recovery) a thread is suspended due to lock conflicts. This count is incremented every time DB2 cannot obtain a lock and has to suspend the thread's unit-of-recovery.

`UOW_TIMEOUTS(integer)`

Specifies the maximum number of times during a single unit-of-recovery that a thread is suspended for a duration that exceeds the timeout value defined for the DB2 subsystem in which the thread executes.

`UOW_ESCALATIONS_SHARED(integer)`

Specifies the maximum number of times during a single unit-of-recovery that DB2 escalates a thread's locks to shared mode.

`UOW_ESCALATIONS_EXCLUSIVE(integer)`

Specifies the maximum number of times during a single unit-of-recovery that DB2 escalates a thread's locks to exclusive mode.

`UOW_PAGE_LOCKS(integer)`

Specifies the maximum number of page locks a thread can concurrently hold during a single unit-of-recovery.

10.7.6 Interval Maximum LIMITS

The following operands define upper and lower limits for thread activity as observed within a user specified period of time termed a ***MINMAX interval***. All of these operands start with the character string '**IMAX**' or '**IMIN**'. The minmax interval is defined explicitly for the current LIMIT statement via the MINMAX_INTERVAL operand or can be defaulted on the MONITOR or DEFAULT statements. Thread/SENTRY acts in accordance with the ACTION operand of the LIMIT statement when it detects an IMIN or IMAX violation. Alternatively, when a minmax interval expires without a violation, Thread/SENTRY resets the activity counters associated with the minmax interval to zero and begins recording activity statistics for a new minmax interval.

Note that minmax interval related limits are considerably more expensive to monitor and enforce than the cumulative life-of-thread limits defined by the MAX_ operands described above. In addition, ensure that the value of the minmax interval is at least ***double*** the monitor ***wake up*** interval.

`IMAX_ELAPSED(time)`

Specifies the maximum elapsed time for the thread during a single minmax interval.

`IMAX_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the maximum CPU time a thread can consume, both within DB2 as well as in the application program, during a single minmax interval.

`IMAX_SRB(time)`

Specifies the maximum SRB time that can be charged to the allied address space associated with a thread during a single minmax interval.

`IMAX_DB2_ELAPSED(time)` *Requires Class 2 Accounting trace*

Specifies the maximum wall clock time that may elapse within DB2 during a single minmax interval.

`IMAX_DB2_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the maximum CPU time a thread may consume within DB2 during a single minmax interval.

`IMAX_IO_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum duration that a thread may wait to perform synchronous I/O during a single minmax interval.

`IMAX_LOCK_WAIT(time)` *Requires Class 3 Accounting trace*

Specifies the maximum duration that a thread may wait to acquire locks and latches during a single minmax interval.

`IMAX_GETPAGES(integer)`

Specifies the maximum number of getpage requests a thread may issue during a single minmax interval.

`IMAX_SQL(integer)`

Specifies the maximum number of SQL data manipulation language (DML) statements of all types a thread can execute during a single minmax interval.

`IMAX_QUERY(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during a single minmax interval. Thread/SENTRY counts SELECT, OPEN, FETCH and CLOSE statements towards this limit.

`IMAX_FETCH(integer)`

Specifies the maximum number of SQL FETCH statements a thread can execute during a single minmax interval.

`IMAX_CHANGES(integer)`

Specifies the maximum number of SQL statements that result in changes to data that a thread can execute during a single minmax interval. Thread/SENTRY counts DELETE, INSERT and UPDATE statements towards this limit.

`IMAX_QUERIED(integer)`

Specifies the maximum number of SQL query type statements a thread can execute during a single minmax interval. Thread/SENTRY counts SELECT, OPEN, multiple-rows FETCH and CLOSE statements towards this limit.

`IMAX_FETCHED(integer)`

Specifies the maximum number of SQL multiple-row FETCH statements a thread can execute during a single minmax interval. Thread/SENTRY uses the QXRWSFETCHD field of the DB2 DSNDQXST structure to compare the number of multiple-rows fetched against the IMAX_FETCHED specified limit.

`IMAX_CHANGED(integer)`

Specifies the maximum number of SQL statements that result in changes to data that a thread can execute during a single minmax interval. Thread/SENTRY counts the number of multiple-rows inserted, updated and deleted towards this limit. Thread/SENTRY uses the QXRWSINSRTD, QXRWSUPDTD and QXRWSDELETD fields of the DB2 DSNDQXST structure to compare the number of rows inserted, updated and deleted respectively, against the IMAX_CHANGED specified limit.

`IMAX_DEADLOCKS(integer)`

Specifies the maximum number of deadlocks tolerated during a single minmax interval.

`IMAX_SUSPENDS(integer)`

Specifies the maximum number of times (during a single minmax interval) that a thread can be suspended due to lock conflicts. This count is incremented every time DB2 cannot obtain a lock and has to suspend the thread's unit-of-recovery.

`IMAX_TIMEOUTS(integer)`

Specifies the maximum number of times (during a single minmax interval) that a thread can be suspended for a duration that exceeds the timeout value defined for the DB2 subsystem in which the thread executes.

`IMAX_ESCALATIONS_SHARED(integer)`

Specifies the maximum number of times (during a single minmax interval) that DB2 may escalate a thread's locks to shared mode before Thread/SENTRY takes the action specified by the ACTION operand.

`IMAX_ESCALATIONS_EXCLUSIVE(integer)`

Specifies the maximum number of times (during a single minmax interval) that DB2 may escalate a thread's locks to exclusive mode before Thread/SENTRY takes the action specified by the ACTION operand.

`IMAX_PAGE_LOCKS(integer)`

Specifies the maximum number of page locks a thread may concurrently hold during a single minmax interval.

10.7.7 Interval Minimums and IDLE Threads

The following operands describe minimum levels of activity for threads during a minmax observation interval. If a thread does not exhibit the minimum level(s) of activity described by these operands (because a thread is idle as an example), the thread is considered to be in violation of the LIMIT policy with which these IMIN operand(s) are associated.

`IMIN_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the minimum CPU time a thread must consume, both within DB2 as well as in the application program, during a single minmax interval.

`IMIN_SRB(time)`

Specifies the minimum SRB time that must be charged to the allied address space associated with a thread during a single minmax interval.

`IMIN_DB2_ELAPSED(time)` *Requires Class 2 Accounting trace*

Specifies the minimum wall clock time that must elapse within DB2 during a single minmax interval.

`IMIN_DB2_CPU(time)` *Requires Class 2 Accounting trace*

Specifies the minimum CPU time a thread must consume within DB2 during a single minmax interval.

`IMIN_GETPAGES(integer)`

Specifies the minimum number of getpage requests a thread must issue during a single minmax interval.

`IMIN_COMMIT(integer)`

Specifies the minimum number of SQL COMMIT statements a thread must execute during a single minmax interval.

`IMIN_SQL(integer)`

Specifies the minimum number of SQL data manipulation language (DML) statements of all types a thread must execute during a single minmax interval.

`IMIN_QUERY(integer)`

Specifies the minimum number of SQL query type statements a thread must execute during a single minmax interval. Thread/SENTRY counts SELECT, OPEN, FETCH and CLOSE statements towards this limit.

`IMIN_FETCH(integer)`

Specifies the minimum number of SQL FETCH statements a thread must execute during a single minmax interval.

`IMIN_CHANGES(integer)`

Specifies the minimum number of SQL statements that result in changes to data that a thread must execute during a single minmax interval. Thread/SENTRY counts DELETE, INSERT and UPDATE statements towards this limit.

`IMIN_QUERIED(integer)`

Specifies the minimum number of SQL query type statements a thread must execute during a single minmax interval. Thread/SENTRY counts SELECT, OPEN, multiple-rows FETCH and CLOSE statements towards this limit.

`IMIN_FETCHED(integer)`

Specifies the minimum number of SQL multiple-row FETCH statements a thread must execute during a single minmax interval. Thread/SENTRY uses the QXRWSFETCHD field of the DB2 DSNDQXST structure to compare the number of multiple-rows fetched against the IMIN_FETCHED specified limit.

`IMIN_CHANGED(integer)`

Specifies the minimum number of SQL statements that result in changes to data that a thread must execute during a single minmax interval. Thread/SENTRY counts the number of multiple-rows inserted, updated and deleted towards this limit. Thread/SENTRY uses the QXRWSINSRTD, QXRWSUPDTD and QXRWSDELETD fields of the DB2 DSNDQXST structure to compare the number of rows inserted, updated and deleted respectively, against the IMIN_CHANGED specified limit.

10.7.8 Operands for Inactive Database Access Threads

The following operands provide a means to govern *inactive* database access threads defined as a database access thread which holds no cursors, locks or database resources. DB2 permits such threads to remain idle indefinitely.

`IDLE_LASTMSG(time)`

Specifies how much time can elapse without an inactive database access thread either sending or receiving a message.

10.7.9 LIMIT Policy Examples

Example 1

Define a thread LIMIT policy (named POLICY1) to apply to all Human Resources applications whose plan names start with the characters ‘HR’. Cancel any thread that consumes more than 10 seconds of CPU time or issues more than 30,000 GETPAGE requests from the time it begins to execute. Record any cancellations in the THREAD_AUDIT table and cite POLICY1 as the rule that was violated.

```
LIMIT
   POLICY_ID(POLICY1)
   PLAN(HR%)
   MAX_CPU(10)
   MAX_GETPAGES(30000)
   ACTION(CANCEL)
   AUDIT(YES)
```

Example 2

Define a thread LIMIT policy (named POLICY2) that will issue a warning when ***any*** thread issues more than ten thousand FETCH statements. These warnings need not be recorded in Thread/SENTRY's THREAD_AUDIT table.

```
LIMIT
   POLICY_ID(POLICY2)
   MAX_FETCH(10000)
   ACTION(WARN)
   AUDIT(NO)
```

Example 3

Define a thread LIMIT policy (named POLICY3) to apply to all Human Resources applications whose plan names start with the characters 'HR'. Cancel any thread that fails to issue a single SQL COMMIT within a 2 minute (120 second) interval. Record any cancellations in the THREAD_AUDIT table and cite POLICY3 as the rule that was violated. Note that a 2 minute minmax period may only be specified when the WAKEUP_INTERVAL operand of the MONITOR statement (see above) is less than or equal to 1 minute.

```
LIMIT
   POLICY_ID(POLICY3)
   PLAN(HR%)
   MINMAX_INTERVAL(120)
   IMIN_COMMIT(1)
   ACTION(CANCEL)
   AUDIT(YES)
```

Example 4

Define a thread LIMIT policy (named POLICY4) that will issue a warning when ***any*** thread fails to issue a getpage request for 30 minutes (1800 seconds). These warnings need not be recorded in Thread/SENTRY's THREAD_AUDIT table. This request is compatible with a WAKEUP_INTERVAL specification of 15 minutes or less.

```
LIMIT
   POLICY_ID(POLICY4)
   MINMAX_INTERVAL(1800)
   IMIN_GETPAGES(1)
   ACTION(WARN)
   AUDIT(NO)
```

10.7.9.1 LIMIT Policy Examples that reference MVS and DDF Accounting Data

Example 5

Define a thread LIMIT policy (named POLICY5) to apply to client SQL applications running on a Windows NT platform where the application name is 'PAYROLL'. Cancel any thread that consumes more than 10 seconds of CPU time or issues more than 30,000 GETPAGE requests from the time it begins to execute. Record any cancellations in the THREAD_AUDIT table and cite POLICY5 as the rule that was violated.

```
LIMIT
   POLICY_ID(POLICY5)
   CLIENT_PLATFORM(NT)
   CLIENT_APPLICATION(PAYROLL)
   MAX_CPU(10)
   MAX_GETPAGES(30000)
   ACTION(CANCEL)
   AUDIT(YES)
```

Example 6

Define a LIMIT policy (named POLICY6) to apply to threads whose accounting string has the characters 'Low Priority' embedded in it starting at character 10. Cancel any thread that consumes more than 5 seconds of CPU time from the time it begins to execute. Record any cancellations in the THREAD_AUDIT table and cite POLICY6 as the rule that was violated.

```
LIMIT
   POLICY_ID(POLICY6)
   ACCOUNT_SUBSTRING1(Low Priority)
   ACCOUNT_START1(10)
   MAX_CPU(5)
   ACTION(CANCEL)
   AUDIT(YES)
```

Example 7

Define a LIMIT policy (named POLICY7) to apply to DB2 for OS/390 threads originating at the location named 'REGION1' and has the following patterns embedded in its accounting string:

The characters 'Group2' embedded in the accounting string starting at position 5
Any substring starting at position 15 which matches the pattern 'Priority__5'.

Cancel any thread meeting the above criteria which in addition consumes more than 15 seconds of CPU time or issues more than 20,000 GETPAGE requests from the time it begins to execute. Record any cancellations in the THREAD_AUDIT table and cite POLICY7 as the rule that was violated.

```
LIMIT
   POLICY_ID(POLICY6)
   PRODUCT_NAME(DSN)
   QMDA_LOCATION(REGION1)
   ACCOUNT_SUBSTRING1(Group2)
   ACCOUNT_START1(5)
   ACCOUNT_SUBSTRING2(Priority__5)
   ACCOUNT_START2(15)
   MAX_CPU(15)
   MAX_GETPAGES(20000)
   ACTION(CANCEL)
   AUDIT(YES)
```

10.8 The NOTIFY_LIST Statement

The NOTIFY_LIST statement lets you define a list of administrative IDs to be notified when Thread/SENTRY detects a policy violation. The notices these IDs receive are *in addition to* the automatic notification Thread/SENTRY sends to the owners of effected threads when user notification is enabled (see the NOTIFY_ENABLED operand of the NOTIFY_LIST or MONITOR statement). The list may include any mix of administrators logged on to terminals connected through local TSO, CICS or IMS systems.

`NAME(name)`

The NAME operand specifies the symbolic name by which this list of administrative IDs is known. Any number of LIMIT policies can explicitly refer to this list of user IDs by name. You can also define (on the MONITOR and DEFAULT statements) a default list of administrator IDs to be notified.

NOTE: When this symbolic name starts with the special string `_WTO` (as in `_WTOABC`), Thread/SENTRY writes notification messages that would ordinarily be directed to the owner of the thread to the operator instead (via a WTO). rather than send a notification message. In contrast, the notifications sent to administrative users specified in the NOTIFY_LIST are processed without change (i.e. not effected by the _WTO special string).

```
NOTIFY_ENABLED(scope_option,(enable_list))
NOTIFY_FREQUENCY(integer)
NOTIFY_MAXIMUM(integer)
```

The NOTIFY_ENABLED, NOTIFY_FREQUENCY and NOTIFY_MAXIMUM operands within the NOTIFY_LIST statement are used to override the corresponding defaults of the MONITOR statement or the Thread/SERIES table of system defaults (defined within module TTS$TSD). See *Section 10.4 MONITOR statement* of this publication for the detailed syntax and description of these operands.

```
SSID(ssid_list)
```

The SSID operand identifies the DB2 subsystem or list of DB2 subsystems to be associated with this notification list.

```
TSO_IDS(tsoid1,tsoid2,...)
```

You can specify a list of one or more TSO IDs via the TSO_IDS operand. The individual IDs should be coded within parentheses and separated by commas or blanks. Notification messages are sent immediately to TSO administrators if they are currently logged on. Otherwise, the administrative notification is saved in the broadcast data set until that administrator logs on to TSO.

```
EMAIL_IDS(name1@domain1.com,name2@domain2.com,...)
```

You can specify a list of one or more Email addresses via the EMAIL_IDS operand. The entire list must be coded within parentheses. The individual Email addresses should be separated by semi colons, commas or blanks.

```
CICS_IDS((TERMINAL(tname1),CICSJOB(jobname1)),
        (TERMINAL(tname2),CICSJOB(jobname2)),...)
```

You can specify a list of one or more CICS administrative ID descriptor pairs. Each pair consists of the CICS terminal name for the recipient along with the associated (owning) CICS system's MVS jobname or started task name. The TERMINAL parameter defines the administrator's terminal name and the CICSJOB parameter defines the CICS system jobname or started task name. Each administrative ID pair should be coded within parentheses and separated by commas or blanks. The entire list must also be coded within parentheses.

```
IMS_IDS((LTERM(ltname1),IMSID(imsname1)),
       (LTERM(ltname2),IMSID(imsname2)),...)
```

You can specify a list of one or more IMS administrator ID descriptor pairs. Each pair consists of the IMS logical terminal name for the recipient along with the associated (owning) IMS system name. The LTERM parameter defines the logical terminal name and the IMSID parameter defines the IMS system name. Each administrative ID pair should be coded within parentheses and separated by commas or blanks. The entire list must also be coded within parentheses.

Example:

Define a NOTIFY_LIST for a set of Human Resource administrators who should be notified when Thread/SENTRY detects a violation by a thread associated with a Human Resource application. Assign the name HRLIST to the notification list. The IDs of the TSO administrators to be notified when a policy violation takes place are HRADM1 and HRADM2. In addition, notification E-mails should be sent to two administrators In addition, three IMS administrators are to be notified. Logical terminals IMADM1 and IMADM2 are logged onto the IMS system named IMST while IMADM3 is logged onto the

IMS system named IMSP. Finally, one CICS based administrator at terminal CIADM1 on system CICSPROD is to be notified.

```
NOTIFY_LIST
   NAME(HRLIST)
   TSO_IDS(HRADM1,HRADM2)
   EMAIL_IDS(name1@domain.com;name2@domain.com)
   IMS_IDS((LTERM(IMADM1),IMSID(IMST)),
               (LTERM(IMADM2),IMSID(IMST)),
               (LTERM(IMADM3),IMSID(IMSP)))
   CICS_IDS((TERMINAL(CIADM1),(CICSJOB(CICSPROD)))
```

10.9 Thread/SENTRY Audit Trail and Log File

Thread/SENTRY maintains an audit trail of its actions and reasons for action along with many available details on the threads that violate Thread/SENTRY policies. Thread/SENTRY records audit trail information in a table named THREAD_AUDIT that resides within the same DB2 subsystem in which the thread which violated a Thread/SENTRY policy is executing. The Thread_Audit table is described and illustrated in Appendix A. Thread/SENTRY inserts a row into this table when AUDIT(YES) is specified (or defaulted) for the policy which was violated.

NOTE: Ask your Thread/SERIES product administrator for the authorization ID of the owner of the THREAD_AUDIT table used by Thread/SENTRY on a particular DB2 subsystem.

In addition, Thread/SENTRY maintains a discrete output file -- to which it writes all its actions -- for each DB2 subsystem it monitors. *Note however, that this logged information is just a small subset of the data recorded in the Thread_Audit table.* Thread/SENTRY also writes subsystem specific diagnostics, trace data and debugging messages to these files. The names of these output files are formed by concatenating the literal string 'TTSL' with the name of the DB2 subsystem being monitored. These log datasets are typically pre-allocated in the standard Thread/SERIES procedure as described in Chapter 20.10.1.

Chapter 11

Operating Thread/SENTRY

This chapter deals with Thread/SENTRY operation. Section 11.1 describes the JCL to run Thread/SENTRY as a submitted job or started task, while Section 11.2 describes the console commands operators use to control Thread/SENTRY's run-time operation. Section. 11.3 describes debugging facilities to diagnose problems with Thread/SENTRY itself.

11.1 Running Thread/SENTRY

Thread/SENTRY can be submitted as a job or run as a started task. Both methods make use of the catalogued procedure named TTSPROC that is described in Chapter 20.10. Thread/SENTRY records all auditable events (warnings, cancellations, etc.) in the `THREAD_AUDIT` table that resides within each monitored DB2 subsystem.

11.1.1 Edit the JCL to invoke Thread/SENTRY

Member TTSRMON of the TTSCNTL library contains JCL to invoke Thread/SENTRY. Add a valid JOB statement in order to submit Thread/SENTRY as a standard jobstream. Alternatively, place member TTSRMON in one of your installation's catalogued procedure libraries in order to run Thread/SENTRY as a started task. Section 20.10.3 discusses special considerations for running Thread/SENTRY as a started task.

The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow Figure 11.1. Be sure to edit member TTSRMON with CAPS ON since it contains comments in lowercase.

```
//jobname  JOB ......                                                    (1)
//TTSMON   EXEC TTSPROC,    Invoke the general Thread/SERIES procedure
//         PROG=TTSMON,     Run the Thread/SENTRY monitor program
//         PARMS='MODE(CANCEL),WTOR(YES)'  Specify optional parameters  (2)
//TTSIN    DD   DSN=Thread.Monitor.Control.Dataset,DISP=SHR             (3)
```

(1) Define a valid job statement if Thread/SENTRY is to be submitted as a standard job. Alternatively, Thread/SENTRY can run as a STARTed task. In this case, no job statement is needed but this procedure (TTSRMON) and its companion TTSPROC must reside in one of your site's catalogued procedure libraries.

(2) Specify optional parameters for Thread/SENTRY through the PARMS parameter. The **MODE** operand governs whether Thread/SENTRY actually cancels threads for which violations are detected or merely issues a warning. The operational mode specified in the distribution tape is CANCEL. The **WTOR** operand governs whether or not Thread/SENTRY issues a WTOR to which an operator can reply in order to control Thread/SENTRY operation. If WTOR(NO) is specified the operator must issue MVS MODIFY or STOP commands to communicate with Thread/SENTRY. The default is WTOR(YES).

(3) Allocate the dataset which contains Thread/SENTRY control statements to the file named TTSIN.

Figure 11.1 Submitting Thread/SENTRY as a job

11.1.2 Allocating the Thread/SENTRY Control File

Thread/SENTRY reads control statements from the dataset allocated to the DDname TTSIN. Control statements may be specified instream following the TTSIN DD statement as in the following example:

```
TTSIN   DD  *
   MONITOR
      SSID(DB2A,DB2B,DB2C)
      INTERVAL(60)
```

In this example, the TTSIN dataset defines the **MONITOR** statement instream with the JCL. The MONITOR statement identifies DB2A, DB2B and DB2C as the DB2 subsystems to be monitored and specifies a monitoring interval of 60 seconds.

Alternatively, the TTSIN file can reference a dataset which contains Thread/SENTRY control statements as in the following example.

```
//TTSIN DD  DSN=thread.monitor.control(member),DISP=SHR
```

11.1.3 Parameter Precedence

Thread/SENTRY run-time operands are derived from the following sources, in the sequence specified:

- parameters specified through the PARM operand of the JCL EXEC statement
- operands specified through the TTSPARMS file
- operands of the MONITOR statement specified through the TTSIN control file

If this scheme, operands specified through the TTSPARMS file take precedence over values specified through the TTSIN control file, while parameters specified through the PARM operand of the JCL EXEC statement take precedence over both.

11.1.4 An illustrative Thread/SENTRY Run

Figure 11.2 illustrates a Thread/SENTRY run that monitors three DB2 subsystems concurrently. Thread/SENTRY creates what is termed a 'Listener' process (an MVS subtask) for each DB2 subsystem it monitors. In this illustrative run, a STOP command was issued soon after initialization to quiesce Thread/SENTRY.

```
17.12.48 JOB01798  $HASP373 RAI6TINS STARTED - INIT DR - CLASS K - SYS CPUX
17.12.49 JOB01798  IEF403I RAI6TINS - STARTED - TIME=17.12.49
17.12.49 JOB01798  +TTS700 - TTSJMC - Thread / SENTRY is starting
17.12.50 JOB01798  +RAI500I - Report process for TTSR is starting
17.13.07 JOB01798  +TTS701 - Thread / SENTRY start normal completion
17.13.07 JOB01798  +TTS704 - Thread / SENTRY operating in CANCEL mode
17.13.07 JOB01798 @22 TTS705 - THREAD/SENTRY JOB RAI6TINS WAITING FOR WORK
17.13.07 JOB01798  +TTS750 - Listener process for DB2 subsystem DB2A is starting
17.13.07 JOB01798  +TTS750 - Listener process for DB2 subsystem DB2B is starting
17.13.07 JOB01798  +TTS750 - Listener process for DB2 subsystem DB2C is starting
17.13.07 JOB01798  +TTS751 - Listener process start completed for DB2 subsystem DB2A
17.13.08 JOB01798  +TTS751 - Listener process start completed for DB2 subsystem DB2B
17.13.08 JOB01798  +TTS751 - Listener process start completed for DB2 subsystem DB2C
17.14.17 JOB01798  +TTS756 - TTSJCMD received the following command
17.14.17 JOB01798  +STOP
17.14.17 JOB01798  +TTS757 - TTSJCMD STOP command received
17.14.17 JOB01798  +TTS702 - Thread / SENTRY is stopping
17.14.17 JOB01798  +TTS752 - Listener process for DB2 subsystem DB2A is stopping
17.14.17 JOB01798  +TTS752 - Listener process for DB2 subsystem DB2B is stopping
17.14.17 JOB01798  +TTS752 - Listener process for DB2 subsystem DB2C is stopping
17.14.18 JOB01798  +RAI502I - Report process for TTSR is stopping
17.14.18 JOB01798  +TTS703 - Thread / SENTRY stop complete
17.14.19 JOB01798  -                                          --TIMINGS (MINS.)--
17.14.19 JOB01798  -JOBNAME  STEPNAME PROCSTEP   RC  EXCP  CONN   TCB   SRB  CLOCK
```

Figure 11.2 Illustrative Thread/SENTRY Run

11.2 Thread/SENTRY Console Commands

Operators can control Thread/SENTRY while it is active using the console commands described in this section. Thread/SENTRY commands can be issued through an MVS MODIFY command or by replying to an outstanding WTOR. For example, to trigger normal shutdown, you can issue the Thread/SENTRY STOP command through the MVS MODIFY interface as follows:

```
F sentry_jobname,STOP
```

Alternatively, provided WTOR(YES) is specified or defaulted, the operator can reply to Thread/SENTRY's outstanding WTOR as follows:

```
@22 TTS705 - THREAD/SENTRY JOB RAI6TINS WAITING FOR WORK
22,STOP
```

With either method, Thread/SENTRY acknowledges your command by issuing message TTS756 and echoing the text of the command back to the console.

```
17.14.17 JOB01798  +TTS756 - TTSJCMD received the following command
17.14.17 JOB01798  +STOP
```

Thread/SENTRY then executes your command.

11.2.1 ABEND command

Upon receipt of the ABEND command, Thread/SENTRY immediately issues a User 2001 ABEND with Reason code 00DB2000 that abnormally terminates the Thread/SENTRY jobstep. A dump is produced if a SYSDUMP file is allocated.

11.2.2 FORCE command

Upon receipt of the FORCE command, Thread/SENTRY immediately issues a User 2001 ABEND with Reason code 00DB2000 that abnormally terminates the Thread/SENTRY jobstep. No dump is produced.

11.2.3 RULE_REFRESH command

Upon receipt of the RULE_REFRESH command, Thread/SENTRY replaces its list of active policy statements with new ones. RULE_REFRESH lets you completely refresh the policies Thread/SENTRY will enforce. The syntax of the RULE_REFRESH command is:

```
RULE_REFRESH {MEMBER(member_name)}
```

where *member_name* specifies an optional, alternative member of the partitioned dataset (PDS) allocated to the DDname TTSCNTL in the Thread/SENTRY JCL. If member is not specified, then Thread/SENTRY simply re-reads control statements from the original dataset or instream source allocated to file TTSIN.

11.2.4 STOP command

Upon receipt of the STOP command, Thread/SENTRY begins to shutdown normally. The various Thread/SENTRY subtasks are quiesced one by one, after which the Thread/SENTRY main task terminates normally.

You can also signal Thread/SENTRY to stop by issuing the MVS STOP command as follows:

```
STOP Thread_sentry_jobname
```

11.3 Thread/SENTRY Debugging

This section describes the debugging facilities used to diagnose problems with Thread/SENTRY itself. Support staff from Relational Architects will assist you in issuing these commands and interpreting the diagnostic information they produce.

11.3.1 SNAP command

The SNAP command can be entered from the console to produce a formatted listing of Thread/SENTRY internal control blocks. This output is written to the dataset allocated to the DDname TTSTRACE.

11.3.2 TRACE_IFCID command

The TRACE_IFCID command can be used to write DB2 Instrumentation Facility records to a trace dataset. These IFCIDs contain the data that Thread/SENTRY evaluates to identify 'problem' DB2 threads that violate your organization's policies.

The TRACE_IFCID command has no operands. The command acts as a switch that can be toggled on and off. Issue the command once to activate IFCID tracing. Issue it again to deactivate tracing.

Each Thread/SENTRY 'listener' process produces IFCID trace output unique to the DB2 subsystem being monitored. Thread/SENTRY writes this output to a discrete trace file that is owned by the 'listener' subtask. Thread/SENTRY builds the name(s) of these trace files by concatenating the literal string 'TTSL' with the name of each DB2 subsystem being monitored.

To use the TRACE_IFCID command, these trace datasets must be pre-allocated to the jobstep with a record format of FBA , a logical record length of 121 and a block size of 1210. The following example illustrates the JCL used to allocate trace datasets for DB2 subsystems named DB2A, DB2B and DB2C.

```
//TTSLDB2A DD  SYSOUT=*,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//TTSLDB2B DD  SYSOUT=*,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//TTSLDB2C DD  SYSOUT=*,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
```

See the discussion in Chapter 20.10.1 of the catalogued procedure used by Thread/SENTRY for further information.

11.4 Thread/SENTRY Control File Compiler parameters

The operation of the ***Thread/SENTRY Control File Compiler*** is controlled by options specified (or defaulted) when the compiler is invoked. To simplify documentation, the ***Thread/SENTRY Control File Compiler*** will be referred to in the rest of this chapter as simply the ***Compiler***.

Options direct the Compiler's function. For example, certain compiler options specify how input from one or more Thread/SENTRY Control Files should be processed while other options govern what outputs the compiler should produce and how these outputs should be formatted. Explicit compiler options can be specified through a **TTSPARM** DD statement, as illustrated in the following example. Alternatively, you can omit the TTSPARM specification entirely and simply use the vendor supplied defaults.

```
//TTSPARM  DD  DSN=thread.sentry.control.file.compiler.options,DISP=SHR
```

If duplicate or conflicting compiler options are specified, the ***first*** value specified takes effect. The TTSPARM dataset should be defined with DCB attributes of RECFM F or FB, and an LRECL of 80. Alternatively, you can code Compiler options instream -- as in the following example, .

```
//TTSPARM  DD  *
source(yes)
options(yes)
*  this is a comment
//*
```

The Compiler treats any line beginning with an asterisk (*) as a comment. Also, the compiler allows to you to split comments into several lines, as in the following example:

```
source(yes)
options(yes)
/*  this is also a
    comment
 */
```

Compiler options are enumerated below in alphabetical order. A vertical bar (|) separates mutually exclusive values for the various compiler options.

COMPILE(policy_scope_option, (policy list))

The COMPILE operand identifies what types of Thread/SENTRY policy statements should be compiled.

The **Policy_scope_option** denotes one of the following global criteria:

ALL All types of Thread/SENTRY policy statements should be compiled.

NONE None of the Thread/SENTRY policy statement types should be compiled.

ONLY Only the policy statement types whose names appear within the parenthesized ***policy list*** should be compiled.

EXCEPT All policy statement types except those which appear within the parenthesized ***policy list*** should be compiled.

The **Policy_List operand** specifies a parenthesized list of one or more of the following Thread/SENTRY policy statement types: See Chapter 10 for a discussion of the various policy statements supported by Thread/SENTRY.

policy_list **{EXCLUDE}{,LIMIT}{,NOTIFY_LIST}**

COMPILE(ALL) is the default.

DEBUG(YES|NO)

The DEBUG parameter directs the Compiler to print the values to be mapped into Thread/SENTRY internal control blocks as they exist at the time the Compiler parses them from Thread/SENTRY control statements.

The Compiler directs DEBUG output to the file whose name is specified by the value of the SNAPDD operand of the TTS#TSD macro. The default DDname for snap output is TTSTRACE. This SNAP dataset should be defined with DCB attributes of RECFM F or FB, with an LRECL of 133.

DESCRCDE(descriptor_code)

The DESCRCDE operand applies only if Compiler output is directed to the operator via an output specification of WTOMSG(YES). The DESCRCDE value denotes an MCS descriptor code for the WTO messages issued by the Compiler.

DUMP(YES|NO)

The DUMP parameter is used to diagnose problems or errors in the Compiler itself. DUMP(YES) requests that all Compiler variables and their corresponding values should be printed when the Compiler completes its processing. DUMP(NO) is the default.

The Compiler directs DUMP output to the file whose name is specified by the value of the SNAPDD operand of the TTS#TSD macro. The default DDname for snap output is TTSTRACE. This SNAP dataset should be defined with DCB attributes of RECFM F or FB, with an LRECL of 133.

INSOURCE(YES|NO)

The INSOURCE parameter governs whether or not the Compiler should print all Thread/SENTRY source statements exactly as they are coded in one or more Thread/SENTRY Control File(s). INSOURCE(NO) is the default.

MAP(YES|NO)

The MAP parameter governs whether or not the Compiler should print the Thread/SENTRY object maps read from the file named TTSMAPS. The Compiler directs MAP output to the file whose name is specified by the value of the SNAPDD operand of the TTS#TSD macro. The default DDname for snap output is TTSTRACE. This SNAP dataset should be defined with DCB attributes of RECFM F or FB, with an LRECL of 133.

MSG_DISPLAY(TERSE|VERBOSE|DEBUG)

The MSG_DISPLAY operand governs which Thread/SENTRY messages should be printed as well as what messages should be suppressed. A specification of TERSE directs Thread/SENTRY to print just essential messages while VERBOSE requests that all messages related to standard processing (including warnings and informational messages) should appear. The DEBUG specification causes all TERSE and VERBOSE messages to be externalized along with additional messages that may be useful for debugging purposes. MSG_DISPLAY(TERSE) is the default.

OPTIONS(YES|NO)

The OPTIONS parameter directs the Compiler to print a list of both explicit and default compiler options. OPTIONS(YES) is the default.

PAUSE(seconds)

The PAUSE parameter directs the Compiler to wait for the specified number of seconds after compile processing is complete before returning control to the calling routine within Thread/SENTRY. PAUSE(0) is the default. This value should not be changed unless your are directed to do so by vendor support personnel.

RETCODE(return_code_value)

The RETCODE parameter directs the Compiler to return the specified completion code to Thread/SENTRY, regardless of the results of compile processing. RETCODE(0) signals the compiler to not override the compile return code. RETCODE(0) is the default and should not be changed unless you are directed to do so by vendor support personnel.

ROUTECDE(routing_code)

The ROUTECDE operand applies only if Compiler output is directed to the operator via an output specification of WTOMSG(YES). The ROUTECDE value denotes an MCS routing code for the WTO messages issued by the Compiler.

SNAP(YES|NO)

The SNAP operand directs the Compiler to produce a formatted listing of Thread/SENTRY internal control blocks as they exist at the time the Compiler terminates.

The Compiler directs SNAP output to the file whose name is specified by the value of the SNAPDD operand of the TTS#TSD macro. The default DDname for snap output is TTSTRACE. This SNAP dataset should be defined with DCB attributes of RECFM F or FB, with an LRECL of 133.

SOURCE(YES|NO)

The SOURCE parameter directs the Compiler to print a formatted representation of Thread/SENTRY control statements read from one or more Thread/SENTRY control files. SOURCE(YES) is the default.

TRACE(YES|NO)

The TRACE parameter is used to diagnose problems or errors in the Compiler itself. Specify 'YES' if you wish all Compiler procedure labels to be traced before execution. Trace output is written to the file defined by the SYSTSPRT DDname. TRACE(NO) is the default and should not be changed unless you are directed to do so by vendor support personnel.

TTSIN(ddname)

The TTSIN parameter allows you to override the name of the file from which the Compiler reads Thread/SENTRY control statements. The default name of this file is TTSIN.

UPPERCASE(YES|NO)

The UPPERCASE parameter governs whether Thread/SENTRY control statements should be translated automatically to uppercase *before* they are processed. The default is NO.

When UPPERCASE(YES) is specified (or defaulted), Thread/SENTRY control statements are ***case insensitive*** since both keywords and operand values are folded to uppercase.

In contrast, UPPERCASE(NO) supports the specification of ***case sensitive*** values for such keywords as PLAN(name) and CORRELATION(name). Although keyword values can be specified in mixed or lower case while UPPERCASE(NO) is in effect, Thread/SENTRY keywords (such as PLAN and CORRELATION) must be coded in UPPERCASE. Otherwise, they will not be recognized.

WTOMSG(YES|NO)

The WTOMSG parameter governs whether Compiler output is routed to the operator or to the file named TTSOUT. WTOMSG(NO) is the default.

Chapter 20

Installing Thread/SERIES Components

This chapter documents the installation of the DB2 Thread Control Series (Thread/SERIES) and briefly describes the contents of the distribution libraries. References to the Thread/SERIES in this chapter pertain to ***all*** components in the Series (namely the Thread/STOPPER ISPF dialog, Thread/STOPPER Batch Facility, Thread/STOPPER Console Facility, Audit View Facility and Thread/SENTRY) unless a specific component is explicitly identified.

The batch jobs described and illustrated in this chapter will have to be modified in keeping with your installation's conventions regarding dataset names, device names, etc. Installation dependent JCL parameters appear in this chapter as values specified within question marks – such as `?value?` in *lower* or *mixed* case type to make them easier to recognize. You should modify these parameters before the jobstreams are submitted.

> ***See the $TTSREAD member of the restored TTSCNTL library for the values you should substitute for the*** `?variables?` ***in this chapter. The $TTSREAD member will identify which variables should be replaced with literal values <u>exactly</u> as defined in $TTSREAD and which symbolic variables should be replaced with site specific values that <u>you</u> designate.***

Once Thread/SERIES components are installed and verified, they can be made generally available to your installation's users. Deployment issues are discussed in the last section of this chapter.

20.1 Pre-installation Planning

Software	Version / Release
z/OS	Any supported release
DB2 Universal Database for z/OS	Any supported release
ISPF/PDF	Any supported release
TSO/E	Any supported release

The above table enumerates the ***minimum*** release levels of IBM system software required to operate Thread/SERIES components. Subsequent releases may also be used.

> ***Please identify the High Level Qualifier (HLQ) used to name the Thread/SERIES library datasets before you begin the installation process***.

Since most Thread/SERIES load modules, control blocks and buffers reside above the 16MB line, standard TSO/ISPF region sizes should provide adequate virtual storage for most Thread/STOPPER Dialog users. We recommend a minimum region size of 4MB for the catalogued procedure used to run Thread/SERIES components in batch or as started tasks.

Most Thread/SERIES load modules distributed in the TTSLOAD dataset are reenterable and refreshable. The libraries containing the load modules for Thread/SERIES should be defined as APF authorized -- as described in Section 20.7.

20.1.1 DB2 Considerations

The installer requires the following DB2 authorizations on each DB2 subsystem to be monitored and controlled by the various Thread/SERIES components:

- authority to BIND packages and plans
- authority to CREATE the audit table In addition, the installer may need authority to create a storage group, database and tablespace in which the Thread/SERIES audit table(s) will be defined.

20.1.2 Restricting Access to Thread/SERIES Functions

Access to authorized Thread/SERIES functions is controlled through the security product (such as RACF, ACF2, or Top Secret) installed at your site. The Thread/SERIES authorization mechanism makes use of the MVS SAF interface to verify whether a user can perform an authorized function. For example, you can let all users access the Thread/STOPPER displays but restrict the use of the CANCEL command to explicitly authorized users. Appendix R describes the RAI Authorization procedure in detail.

20.2 Pre-installation Preparation

20.2.1 Accounting Trace Classes

DB2 accounting traces should also be active in order to make optimal use of Thread/SERIES components. In addition to the default Class 1 trace, accounting class 2 provides useful TCB, SRB and elapsed time information. Class 3 tracing makes available additional wait times within DB2. If these traces are not active, Thread/SERIES components display the unavailable values as N/P (not present).

Relational Architects strongly recommends that Accounting trace classes 1 and 3 be active continuously in all environments. Trace classes 2 and 7 are also recommended to collect TCB, SRB, and elapsed time information for plans and packages, respectively. Note that Class 2 trace overhead is modest while trace Class 7 has *minimal performance impact* if trace class 2 is active.

You can start these accounting traces automatically during DB2 initialization by enabling SMF ACCOUNTING through the DB2 Tracing Panel (DSNTIPN). Alternatively, accounting traces can be started explicitly through the following DB2 command:

```
START TRACE(ACCTG) CLASS(1,2,3,7)
```

20.2.2 Address Space Priorities

The WLM performance goal appropriate for the Thread/SENTRY address space should be at least as high as that for any DB2 thread they will be expected to monitor and control. The performance goals associated with a CICS address space or IMS dependent region should be suitable for both address spaces.

Performance characteristics for the Thread/SERIES started tasks can be specified through Work Load Manager.

20.2.3 Preparation for CICS DB2 threads

In order to cancel CICS/DB2 threads and send notifications to affected users, the Thread/SERIES components need the LU6.2 Unit-of-Work ID generated by CICS. This is true for both terminal and non terminal driven CICS tasks. To ensure that the necessary accounting information is available, the TYPE=INIT statement of the DSNCRCT macro should specify TOKENI=YES as the default. Alternatively, the TYPE=POOL or TYPE=ENTRY statements associated with particular CICS transactions should specify TOKENE=YES. See the description of the CICS attachment facility macro (DSNCRCT) in the DB2 Installation Guide for further information.

The Thread/SERIES components also need authorization to issue CICS commands through the MVS MODIFY interface. See the CICS/RACF Security Guide and other volumes of the CICS product library for further details.

20.2.4 TSO Command Processor Limiting

Sites that utilize the command limiting feature of ACF2 (or an analogous site developed facility) may need to add TTSD, RLXS and RCXFE to the list of TSO command processors available to individual users and/or the entire site. For ACF2, this entails coding $TSOCMD macro statements like the following in the ACF source module that contains the list of valid TSO commands:

```
$TSOCMD TTSD
$TSOCMD RLXS
$TSOCMD RCXFE
```

See the ACF$CMDS source module in the SYS1.ACFMAC library for a sample list.

> ***NOTE:*** *If ACF2 command limiting is in effect and RCXFE, RLXS or TTSD are not defined as valid, the problem may manifest itself as* `COMMAND NOT FOUND` *or* `INVALID SYNTAX` *even though these command processors are in the standard MVS search order.*

20.2.5 RAI Server Address Space

RAI Server is an authorized address space that is a ***prerequisite*** for the operation of all Thread/SERIES components. RAI Server software at release V4R4M0 or later must be installed and active in order for Thread/SERIES components to run. Refer to the "*RAI Server Installation and Operations Guide" (RAI publication RSV-001)* for details.

RAI Server must be configured with Thread/SERIES passwords (as described in Section 20.6 of this chapter). In addition, the Thread/SENTRY Email notification feature also requires RAI Server configuration as described in Chapter 6 of the "*RAI Server Installation and Operations Guide"*.

Please note, that the default RAI Server name is RAI0. Should you need to run the RAI Server with a different name, be sure to customize and submit JCL in member TTSJRSN of the TTSCNTL library. This job updates the various Thread/SERIES components with the new RAI Server name.

20.2.6 Requirements to Cancel Inactive Threads

NOTE: In order to successfully execute `CANCEL TYPE(INACTIVE)` commands, the following installation requirements must be satisfied: .

- Version 4 Release 4 Modification level 0 or later of the RAI server must be installed. Examine the `PARMS` member of the `RSVCNTL` dataset allocated in the JCL for the RAI Server. started task and ensure the `MCS(YES)`parameter is specified. The RAI Server address space must be restarted if this parameter is changed from `MCS(NO)` to `MCS(YES)`.

- The following RACF definitions must be implemented (via the following RACF commands):

```
SETROPTS CLASSACT(OPERCMDS)
RDEFINE OPERCMDS (MVS.VARY.TCPIP.DROP) UACC(NONE)
PE MVS.VARY.TCPIP.DROP CLASS(OPERCMDS) ID(userid) ACCESS(CONTROL)
SETROPTS RACLIST(OPERCMDS) REFRESH
```

  where: `userid` is the RACF ID of any user who submits a job that invokes the Thread/STOPPER Batch facility. In addition, any ID associated with the Thread/CONSOLE job or started task must be similarly enabled.

20.3 Installation Summary

This section summarizes the steps involved in installing Thread/SERIES components. Detailed descriptions of these steps begin in Section 20.5. In this discussion, ***?ttshlq?*** refers to the *target libraries* created at your site during installation, and ***?ttsvrm?*** refers to the version, release and modification level of the Thread/SERIES software supplied on the distribution tape.

1. Create and restore the Thread/SERIES target libraries on your host system using one of the methods described in Section 20.5.

2. APF authorize the Thread/SERIES load libraries (Section 20.7)

3. The Thread/SERIES exec library is distributed in fixed blocked record format. If necessary, convert this Thread/STOPPER target library to variable blocked format. (Section 20.8)

4. Define the Thread/SERIES table, packages and plans on each DB2 subsystem whose operation you wish to monitor and control. GRANT the DB2 authorizations required by the various Thread/SERIES components. (Section 20.9)

5. Edit the Thread/SERIES catalogued procedure and the various jobstreams that invoke it. (Section 20.10)

6. Define the resource profile named RAI.TTS.TCAN as described in Chapter 4 of the *RAI Server Installation and Operations Guide (publication RSV-001).* This ***required*** profile definition enables authorized users and processes to cancel DB2 threads using Thread/SERIES.

7. Review and implement the configuration requirements for the Thread/SENTRY Email notification feature as discussed in Section 20.11 of this chapter.

8. Define the VTAM application major node used by Thread/SERIES components (Section 20.12).

9. Optionally edit the TTSPAL catalogued procedure (Section 20.13).

10. Optionally update the vendor supplied defaults for Thread/SERIES components (Section 20.14).

11. Prepare the ISPF environment to run the Thread/STOPPER dialogs in a TSO/ISPF environment. (Section 20.15).

12. Optionally, install the Thread/SERIES Audit View Facility (Section 20.16).

13. Verify installation of Thread/SERIES components by deliberately canceling the DB2 thread associated with the Thread/SERIES Installation Verification Procedure. (Section 20.17)

14. Conduct post-installation and deployment procedures that include granting execute plan privileges to those authorized to use Thread/SERIES components. (Section 20.18)

> ***NOTE:*** *Should you experience difficulty installing or using Thread /SERIES components, RAI's support staff stands ready to help. Outside North America call your local RAI representative.*

20.3.1 Thread / SERIES Migration Summary

You may skip this section if you are conducting an initial Thread/SERIES installation. This section summarizes the steps involved in ***migrating*** to the current Thread/SERIES release from a prior release. You may run the current release and a prior release in *parallel* during a test period or immediately *replace* the old release if you prefer. Detailed descriptions of these steps begin in Section 20.5.

1. Create and restore the Thread/SERIES target libraries on your host system using one of the methods described in Section 20.5.

2. APF authorize the Thread/SERIES load libraries (Section 20.7)

3. The Thread/SERIES exec library is distributed in fixed blocked record format. If necessary, convert this Thread/STOPPER target library to variable blocked format. (Section 20.8)

4. Define Thread/SERIES components to each DB2 subsystem whose operation you wish to monitor and control, as described in Section 20.9. The column structure of the THREAD_AUDIT table may change from one Thread/SERIES release to another. To ***replace*** the THREAD_AUDIT table for the current release, simply DROP and recreate the THREAD_AUDIT table and re-bind the Thread/SERIES application plans (TTSP*?ttsvrm?*, TTSM*?ttsvrm?* and TTSI*?ttsvrm?*). GRANT the DB2 authorizations required by the various Thread/SERIES components.

   Alternatively, to run the current Thread/SERIES release ***in parallel with a prior release***, we recommend you create a new THREAD_AUDIT table whose owner is TTS*?ttsvrm?*. Then bind new application plans named TTSP*?ttsvrm?*, TTSM*?ttsvrm?* and TTSI*?ttsvrm?*, respectively and specify a qualifier (for tables, views and aliases) of TTS*?ttsvrm?*.

5. Revise the Thread/SERIES catalogued procedure and the Thread/STOPPER and Thread/SENTRY jobstreams that invoke it. (Section 20.10)

6. Define the resource profile named RAI.TTS.TCAN as described in Chapter 4 of the *RAI Server Installation and Operations Guide (publication RSV-001)* if it is not ***already*** defined. This profile definition is now required to enable authorized users and processes to cancel DB2 threads using Thread/SERIES.

7. There is no need to redefine the VTAM application major node used by Thread/ STOPPER and Thread/SENTRY. (Section 20.12).

8. Optionally, update the vendor supplied defaults for Thread/SERIES components (Section 20.14).

9. Prepare the ISPF environment to run the Thread/STOPPER dialogs as described in Section 20.15.

10. The $TTSREAD member of the TTSCNTL library will state whether reinstallation of the Thread/SERIES Audit View Facility (Section 20.16) is necessary.

11. Verify installation of Thread/SERIES components by deliberately canceling the DB2

thread associated with the Thread/SERIES Installation Verification Procedure. (Section 20.17)

12. Conduct post-installation and deployment procedures that include granting execute plan privileges to those authorized to use Thread/SERIES components. (Section 20.18)

20.4 Thread/SERIES Product Libraries

Figure 20.1 briefly describes the Thread/SERIES distribution files. Figure 20.2 describes the DCB attributes and DASD space requirements for each of the distribution files. Please note that storage estimates are based on the IBM 3390 Direct Access Storage Device. Thread/SERIES target libraries are allotted as many PDS directory blocks as will fit on a full track.

Thread/SERIES distribution datasets can be reblocked to a larger blocksize if necessary. In addition, the panel and message libraries may be copied into libraries with a variable record format.

> ***Please Note:*** *If you want to concatenate datasets having unequal record lengths, their record format* ***must*** *be variable.*

Of particular note are the $PGMDIR and $TTSREAD members of the TTSCNTL library. The $PGMDIR member is a program directory which describes the contents of the TTS product libraries while the $TTSREAD file contains the most current release notes for Thread/SERIES.

```
Tape
File   Dataset         Contents
----   -----------     ------------------------------------------
1      TTSINST         JCL and REXX exec to receive the TTSXMIT file
2      TTSXMIT         Complete Thread/SERIES products in TSO/E XMIT format

TTSXMIT Dataset Contents

Library   Contents
--------  ------------------------------------------
TTSCAF91 Thread/SERIES CAF modules for DB2 Version 9.1
TTSCAF10 Thread/SERIES CAF modules for DB2 Version 10.1
TTSCAF11 Thread/SERIES CAF modules for DB2 Version 11.1
TTSCNTL   JCL, DDL, control statements and README files
TTSDBRM   Thread/SERIES database request module library
TTSEXEC   Thread/SERIES exec library (RECFM=FB)
TTSLOAD   Thread/SERIES load module library
TTSMACS   Thread/SERIES macro library
TTSMAPS   Thread/SERIES compiled control block maps
TTSMLIB   Thread/SERIES ISPF message library
TTSPLIB   Thread/SERIES ISPF panel library
TTSREPOS  RAI Repository in unloaded, sequential format
```

Figure 20.1 Thread/SERIES Distribution Datasets

```
Dataset      Record  Lrecl   Block   Space    3390   PDS Dir
Type         Format           Size    Unit    DASD    Blocks
--------     ------  -----   -----    ----   -----   -------
TTSCNTL        FB       80    6160    Trks   (7,1)        43
TTSCAF91       U         0    6233    Trks   (6,1)         1
TTSCAF10       U         0    6233    Trks   (6,1)         1
TTSCAF11       U         0    6233    Trks   (6,1)         1
TTSDBRM        FB       80    6160    Trks   (9,1)        43
TTSEXEC        FB       80    6160    Trks   (3,5)        43
TTSINST        FB       80    6160    Trks   (2,1)        43
TTSLOAD        U         0    6233    Trks   (123,5)      43
TTSMACS        FB       80    6160    Trks   (15,5)       43
TTSMAPS        FB       80    6160    Trks   (2,5)        43
TTSMLIB        FB       80    6160    Trks   (9,5)        43
TTSPLIB        FB       80    6160    Trks   (17,5)       43
TTSREPOS       VB    32000   32004    Trks   (6,1)         0
```

Figure 20.2 Thread/SERIES distribution datasets

20.5 Restore the TTS Target libraries to your Host system

The first step is to create and restore the Thread/SERIES target libraries on your host z/OS system using one of the following methods:

- Load from tape (Section 20.5.1)
- Download z/OS files from the Relational Architects FTP site (Section 20.5.2)
- Obtain zipped PC files via E-mail attachments (Section 20.5.3) or download them from www.relarc.com.

Once the Thread/SERIES target libraries are restored, continue with the batch installation steps described in Section 20.6.

The *restore* process edits numerous members of the TTSCNTL and TTSEXEC library to aid in product installation. This *edit* process replaces the following symbolic variables (as they appear in TTSCNTL and TTSEXEC library members) with their default or specified values as follows:

Variable Name	Value	Description
?ttshlq?		The high level qualifier for the Thread/SERIES product libraries.
?ttsvrm?	710	Thread/SERIES version, release and modification level.
?volume?		The name of the DASD volume on which the distribution libraries of the Thread/SERIES reside

Table 20.1 Automatically assigned symbolic variables, values and descriptions.

20.5.1 Installation from Tape

Installation from tape entails restoring the TTSINST dataset from distribution tape file 1 to direct access storage. Figure 20.3 illustrates an IEBCOPY job you can use to restore the TTSINST library. The most current instructions are embedded as comments within the JCL.

```
//jobname  JOB
//  SET  RAIHLQ=?ttshlq?,              High level qualifiers of the RAI
//*                                    product target libraries
//       TUNIT=?tape?,                 Unit name for the tape device
//       TVOLUME=?vvvvvv?              Tape/Cartridge Volume Serial #
//*
//* SET  DASDATTR='VOL=SER=?volume?,UNIT=SYSALLDA,' Non-SMS environment
//* SET  DASDATTR='STORCLAS=?storclas?,'           Use SMS explicitly
//* SET  DASDATTR=''                              Use SMS by default
//*
//*--------------------------------------------------------------------
//*
//*    This job performs the following functions:
//*
//*    1 - Restores the Thread/SERIES installation library
//*        (TTS.TTSINST) from a 3480 tape cartridge to DASD.
//*
//*    Status = Version v Release r Modification level m
//*
//*    (c) Copyright Relational Architects Intl - 2001. 2007.
//*    Licensed Material - Program property Relational Architects Intl
//*
//*
//*    Edit this jobstream as follows:
//*    ===============================
//*
//*    1.  Add a valid job card
//*
//*    2.  GLOBALLY change all occurences of the following
//*        parameter strings as described below:
//*
//*        a.  Change ?ttshlq? to the high level qualifiers of the
//*            Thread/SERIES libraries to be created at your site.
//*            The recommended value is TTS.V6R1M0
//*
//*        b.  Change ?tape? to the unit name of the tape device
//*
//*        c.  Change ?vvvvvv? to the Volume Serial Number of the
//*            distribution tape or cartridge from the vendor
//*
//*    3.  Uncomment ONE of the 'SET DASDATTR' JCL statements above:
//*                ===
//*
//*        a.  For non-SMS environments, uncomment the statement:
//*
//*            SET DASDATTR='VOL=SER=?volume?,UNIT=?unit?,'
//*
//*            Change ?volume? to the name of the DASD volume on
//*            which the distribution libraries of the Thread/SERIES
//*            should be allocated.  The default is blank to
//*            receive the distribution libraries to a work pack.  If
//*            you specify a volume parameter, make sure the volume is
//*            eligible, i.e. defined in VATLSTxx as private.  Note:
//*            the VOLUME and UNIT parameters are mutually exclusive
//*            with the STORCLAS parameter.
//*
//*        b.  When you wish to use SMS explicitly, (that is
//*            SMS is active but does not manage all DASD allocations
//*            by default) uncomment the statement:
//*
//*            SET  DASDATTR='STORCLAS=?storclas?,'
```

```
//*
//*            and change ?storclas? to the name of a valid
//*            SMS storage management class defined at your site.
//*            Note: the STORCLAS(storclas) operand is mutually
//*            exclusive with the UNIT and VOLUME parameters.
//*
//*        c.  When SMS, by default, manages ALL dataset allocations
//*            at your site, uncomment the statement:
//*
//*            SET  DASDATTR=
//*
//*    4.  Submit this job.  This job should complete with a return
//*        code Ø.
//*
//*    Once this job completes successfully, the Thread/SERIES
//*    installation library will reside on your host system.
//*    You should then proceed as follows:
//*
//*    Edit and submit the JCL in member TTSJRPT of the TTSINST library
//*    to restore the complete Thread/SERIES product.
//*
//*--------------------------------------------------------------------
//*
//*        Restore the TTSINST library from file 1 of the
//*        Thread/SERIES distribution tape
//*
//*--------------------------------------------------------------------
//RESTORE  EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=TTS.TTSINST,LABEL=(1,SL),
//         UNIT=&TUNIT,VOL=SER=&TVOLUME,DISP=(OLD,PASS)
//SYSUT2   DD  DSN=&RAIHLQ..TTSINST,DISP=(NEW,CATLG,DELETE),
//   &DASDATTR.SPACE=(TRK,(15,1,43),RLSE),DCB=(*.SYSUT1)
//*
```

Figure 20.3 Restore the TTSINST library to disk

Once the TTSINST library is restored, edit the TTSJRPT member shown in Figure 20.4 below. The most current instructions are embedded as comments within the JCL.

> ***NOTE**: Be sure to edit the TTSJRPT job with **CAPS ON** since it contains lower case text.*

```
//jobname  JOB
//  SET  RAIHLQ=?ttshlq?,              High level qualifiers of the RAI
//*                                    product target libraries
//       DUNIT=?tape?,                 Unit name for the tape device
//       DVOLUME=?volume?              Tape/Cartridge Volume Serial #
//*
//* SET  DASDATTR='VOL=SER=?volume?,UNIT=?unit?,'   Non-SMS environment
//* SET  DASDATTR='STORCLAS=?storclas?,'            Use SMS explicitly
//* SET  DASDATTR=''                               Use SMS by default
//*
//*--------------------------------------------------------------------
//*
//*    This job performs the following functions:
//*
//*    1 - Restores the Thread/SERIES product (in TSO/E TRANSMIT
//*        format) to a new dataset allocated on DASD.
//*
//*    2 - Issues TSO/E RECEIVE commands to create and restore the
//*        Thread/SERIES distribution libraries to your z/OS or
//*        OS/390 host system.
//*
//*    Status = Version v Release r Modification level m
//*
//*    (c) Copyright Relational Architects Intl - 1999, 2005.
//*    Licensed Material - Program property Relational Architects Intl
//*
//*
//*    Edit this jobstream as follows:
//*    ===============================
//*
//*    1.  Add a valid job card
//*
//*    2.  GLOBALLY change all occurences of the following
//*        parameter strings as described below:
//*
//*        a.  Change ?ttshlq? to the high level qualifiers of the
//*            Thread/SERIES libraries to be created at your site.
//*            The recommended value is TTS.V6R1M0
//*
//*        b.  Change ?tape? to the unit name of the tape device
//*
//*        c.  Change ?volume? to the Volume Serial Number of the
//*            distribution tape or cartridge from the vendor
//*
//*    3.  Uncomment ONE of the 'SET DASDATTR' JCL statements above:
//*                ===
//*
//*        a.  For non-SMS environments, uncomment the statement:
//*
//*            SET DASDATTR='VOL=SER=?volume?,UNIT=?unit?,'
//*
//*            Change ?volume? to the name of the DASD volume on which
//*            the distribution libraries of the Thread/SERIES should
//*            be allocated.  The default is blank to receive the
//*            distribution libraries to a work pack.  If you specify
//*            a volume parameter, make sure the volume is eligible,
//*            i.e. defined in VATLSTxx as private.  Note:  the VOLUME
//*            and UNIT parameters are mutually exclusive with the
//*            STORCLAS parameter.
//*
//*            Then change ?unit? to the UNIT name for the target
//*            DASD volume on which the Thread/SERIES distribution
//*            libraries will be allocated.  For example specify 3380,
//*            3390 or SYSDA.
//*
//*        b.  When you wish to use SMS explicitly, (that is
//*            SMS is active but does not manage all DASD allocations
//*            by default) uncomment the statement:
//*
//*            SET  DASDATTR='STORCLAS=?storclas?,'
//*
//*            and change ?storclas? to the name of a valid
```

```
//*            SMS storage management class defined at your site.
//*            Note: the STORCLAS(storclas) operand is mutually
//*            exclusive with the UNIT and VOLUME parameters.
//*
//*        c.  When SMS, by default, manages ALL dataset allocations
//*            at your site, uncomment the statement:
//*
//*            SET  DASDATTR=
//*
//*    4.  Submit this job.  All steps should execute with completion
//*        code Ø
//*
//*    Once this job completes successfully, the Thread/SERIES
//*    distribution libraries will reside on your host system.
//*    You should then proceed as follows:
//*
//*    1.  Edit and submit the JCL in member RAIJPSWD of the TTSCNTL
//*        library to define temporary product passwords for the trial
//*        period.  Be sure to use the product names and password
//*        strings EXACTLY as supplied in the vendor's cover letter
//*        or E-mail.
//*
//*--------------------------------------------------------------------
//COMMON   SET DHLQ=TTS.TTS,
//         THLQ=&RAIHLQ..TTS,
//         TDD='DISP=(NEW,CATLG,DELETE),DCB=(*.SYSUT1)'
//*--------------------------------------------------------------------
//*
//*        Restore the file containing the Thread/SERIES product to
//*        DASD.  This dataset is in TSO/E TRANSMIT format.
//*
//*--------------------------------------------------------------------
//XMIT     EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DUMMY
//SYSUT1   DD  DSN=&DHLQ.XMIT,LABEL=(2,SL),
//         UNIT=&DUNIT,VOL=SER=&DVOLUME,DISP=(OLD,KEEP)
//SYSUT2   DD  &TDD,SPACE=(TRK,(5ØØ,1Ø),RLSE),&DASDATTR.DSN=&THLQ.XMIT
//*--------------------------------------------------------------------
//*
//*        RECEIVE (create and restore) the Thread/SERIES
//*        distribution libraries
//*
//*--------------------------------------------------------------------
//RECEIVE  EXEC PGM=IKJEFTØ1,DYNAMNBR=1Ø,REGION=ØM,
//         PARM='TTSERP &RAIHLQ &DASDATTR'
//SYSEXEC  DD  DISP=SHR,DSN=&THLQ.INST
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  DUMMY
```

Figure 20.4 Run the TTSJRPT Job

Submit this job and check that all steps execute with completion code 0. Once this job completes successfully, the Thread/SERIES libraries will reside on your host system. You should then proceed to Section 20.6.

20.5.2 FTP Installation from the RAI website

The Customer Support section of the Relational Architects website (www.relarc.com) contains the most current information about installing the Thread/SERIES products via FTP download. You must obtain a User ID and password from Relational Architects (or your local RAI representative) in order to access this section of the Relational Architects website.

The Relational Architects FTP site contains the following Thread/SERIES files:

FTP File	Format	z/OS Dataset	Member of PDS	Description
tts?TTSVRM?.readme.txt	ASCII	TTSCNTL	$TTSREAD	Release Notes
tts?TTSVRM?.xmi	binary	TTSXMIT	N/A	Thread/SERIES products in TSO/E XMIT format
tts?TTSVRM?.zip	ZIP	N/A	N/A	Contains the following format in ZIP format: tts?TTSVRM?.readme.txt tts?TTSVRM?.tmp tts?TTSVRM?.xmi tts?TTSVRM?at.jcl
tts?TTSVRM?.tmp	binary	TTSINST	TTSERP	Installation REXX EXEC
tts?TTSVRM?at.jcl			TTSJRPA	JCL with which to create and restore the distribution libraries of the Thread/SERIES to your z/OS host system.
tts?TTSVRM?at.txt	ASCII			
tts?TTSVRM?ft.jcl	binary		TTSJRPF	JCL allocates a pair of datasets, runs the FTP to download the Thread/SERIES software from the RAI FTP site, and issues TSO/E RECEIVE commands to create and restore the distribution libraries of the Thread/SERIES to your z/OS host system.
tts?TTSVRM?ft.txt	ASCII			
tts?TTSVRM?ta.txt	ASCII		TTSJRPTI	Restores the Thread/SERIES installation library (TTS.TTSINST) from a 3480 tape cartridge to DASD.

20.5.3 Installation via E-mail

The E-mail you receive from Relational Architects or your local RAI representative will contain both the instructions and attached files with which to install the Thread/SERIES product.

20.6 Define Passwords

Specify product passwords for Thread/STOPPER and/or Thread/SENTRY in the RAI Server password file. Be sure to specify the product name and password strings ***exactly*** as supplied in the vendor cover letter, fax or E-mail. Chapter 3 of the "*RAI Server Installation and Operations Guide*" *(publication RSV-001)* describes this procedure in detail.

> ***NOTE:*** *This method of password definition via the RAI Server is* ***introduced*** *with V6.1 of Thread/SERIES and* ***supersedes*** *the method used in prior versions and releases of Thread/SERIES.*

20.7 APF Authorize the Thread/SERIES Load Libraries

The program libraries for Thread/SERIES must be defined as APF authorized in either an IEAAPFxx or PROGxx member of SYS1.PARMLIB. The following program libraries were restored by one of the methods described in the previous Section.)

```
?ttshlq?.TTSCAF91
?ttshlq?.TTSCAF10
?ttshlq?.TTSCAF11
?ttshlq?.TTSLOAD
```

Alternatively, you can dynamically define these datasets as APF authorized libraries with MVS system commands like the following:

```
SETPROG APF,FORMAT=DYNAMIC
SETPROG APF,ADD,DSNAME=?ttshlq?.TTSCAF91,SMS|VOLUME=volser
SETPROG APF,ADD,DSNAME=?ttshlq?.TTSCAF10,SMS|VOLUME=volser
SETPROG APF,ADD,DSNAME=?ttshlq?.TTSCAF11,SMS|VOLUME=volser
SETPROG APF,ADD,DSNAME=?ttshlq?.TTSLOAD,SMS|VOLUME=volser
```

Any libraries concatenated with the Thread/SERIES program libraries via JCL STEPLIB or JOBLIB statements ***must be APF authorized***. Alternatively, when running the Thread/STOPPER dialog application under ISPF (an unauthorized program), the Thread/SERIES program libraries may be freely concatenated with ***unauthorized libraries***. This is true both for libraries pre-allocated to the ISPLLIB DDname (before ISPF is invoked) as well as dynamic program libraries defined via the ISPF LIBDEF service.

20.8 Convert the record format of the TTS EXEC library to variable blocked format (If necessary)

This step applies *only* if you allocate REXX exec libraries with a variable blocked record format (RECFM=VB). The Thread/SERIES exec library (dataset ?ttshlq?.TTSEXEC) is distributed in fixed blocked record format with a logical record length of 80 and a blocksize of 6160. This dataset was restored to disk in the previous step with these same attributes. If the other libraries with which it will be concatenated have a variable blocked record format, you need to allocate a new REXX exec library whose record format is also variable blocked. This is because datasets must have *like* characteristics in order to be concatenated. Should your site make use of exec libraries in variable blocked format, we recommend you proceed as follows:

- Rename the TTSEXEC dataset just restored from tape to TTSEXECF.
- Allocate a new TTSEXEC dataset with variable blocked attributes via PDF Option 3.2, the TSO ALLOCATE command or via JCL. When choosing a record length and blocksize for this variable blocked library, consider the characteristics of the target DASD volume, as well as the block size of other exec libraries with which it will be concatenated.
- Use PDF option 3.3 to copy all members of the fixed blocked library into the new variable blocked dataset. (Thread/SERIES source execs have no line numbers so they can be copied directly, without source changes.)
- Delete the TTSEXECF library which contains fixed blocked records.

20.9 Prepare each DB2 subsystem for Thread/SERIES Components

This step defines Thread/SERIES components to those DB2 subsystems you wish to monitor and control with Thread/SERIES. Defining Thread/SERIES to DB2 entails the following steps:

- Create the Thread/SERIES DB2 Audit table named ?ttsvrm?.THREAD_AUDIT. Make sure to specify an existing tablespace and database in which the THREAD_AUDIT table will reside, or create new ones.
- BIND the Thread/SERIES packages and plans.
- Installing the run-time only version of RLX for DB2 to power the Thread/SERIES VIEW facility with which to browse the THREAD_AUDIT table.

A single copy of Thread/SERIES can be installed for any or all of the DB2 subsystems defined within a single MVS system. However, a *separate* copy of Thread/SERIES is *required* for each MVS system image.

20.9.1 Edit and submit the Thread/SERIES DB2 definitions jobstream

Edit and submit the job in member TTSJDB2 of the TTSCNTL library to define Thread/SERIES to the target DB2 subsystem. ***A separate job must be edited and submitted for each additional DB2 subsystem into which Thread/SERIES is being installed***. The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow the figure. You should review and update the values that are specified within question marks (such as `?value?`). However, you *need not* specify values for the symbolic variables `?ttshlq?` , `?ttsvrm?`, and `?volume?`. Updating values for these variables were substituted as part of the processing described in Section 20.5.

```
//jobname  JOB . . .
// SET AUTHID=?authid?,                                          (1a)
//     DB2EXIT='?db2hlq?.SDSNEXIT',                              (2a)
//     DB2LOAD='?db2hlq?.SDSNLOAD',                              (2b)
//     DB2SN=?dsn?,                                              (3a)
//     TTSHLQ='?ttshlq?'
//*
//PROCLIB JCLLIB ORDER=&TTSHLQ..TTSCNTL
//*_____________________________________________________________________
//*    Invoke TTSPDB2 procedure to install Thread/SERIES on DB2
//*    SSID=?dsn?
//*_____________________________________________________________________
//?dsn?    EXEC TTSPDB2,
// AUTHID=&AUTHID,
// DB2=&DB2SN,
// DB2EXIT=&DB2EXIT,
// DB2LOAD=&DB2LOAD,
// PLAN=TTS?ttsvrm?C,
// TTSHLQ=&TTSHLQ
//CREATE.SYSIN  DD  *
 create table TTS?ttsvrm?.thread_audit
    (db2_subsystem                 char(4),
     uniqueness_value              char(12),
     thread_token                  integer,
     ace_address                   char(8),
     action_taken                  char(12),
     action_status                 char(12),
     policy_id                     char(8),
     policy_reason                 char(24),
     action_method                 char(40),
     action_date                   date,
     action_timestamp              timestamp,
     action_acee_id                char(8),
     action_db2auth_id             varchar(128),
     action_group_name             char(8),
     action_term_id                char(8),
     action_appl_id                char(8),
     action_surr_id                char(8),
     action_cpu_id                 char(12),
     creation_date                 char(10),
     creation_time                 char(8),
     creation_microsecs            char(7),
     correlation_name              char(12),
     plan_name                     char(8),
     program_name                  varchar(128),
     authorization_id              varchar(128),
     connection_name               char(8),
     original_operator             varchar(128),
     mvs_system                    char(8),
     address_space_id              char(4),
     thread_jobname                char(8),
     where_executing               char(4),
     accounting_token              char(22),
     display_sql_count             smallint,
     status_code                   char(2),
     network_id                    char(8),
     lu_name                       char(8),
```

```
    commit_count                integer,
    sql_dml_count               integer,
    getpages_issued             integer,
    status_literal              char(40),
    connection_code             integer,
    connecting_system           char(12),
    tcb_address                 char(8),
    total_elapsed_time          char(15),
    db2_elapsed_time            char(15),
    class1_tcb_time             char(15),
    class2_tcb_time             char(15),
    home_srb_time               char(15),
    io_wait_time                char(15),
    lock_wait_time              char(15),
    dist_location               varchar(128),
    dist_appc_id                char(8),
    dist_session_id             char(16),
    thread_flag1                char(1),
    thread_flag2                char(1),
    thread_flag3                char(1),
    thread_flag4                char(1),
    thread_flag5                char(1),
    thread_flag6                char(1),
    thread_flag7                char(1),
    thread_flag8                char(1),
    thread_flag9                char(1),
    thread_flaga                char(1),
    thread_flagb                char(1),
    qmda_product                char(3),
    qmda_version                char(2),
    qmda_release                char(2),
    qmda_mod                    char(1),
    qmda_location               char(16),
    qmda_netid                  char(8),
    qmda_luname                 char(8),
    qmda_connect_name           char(8),
    qmda_connect_type           char(8),
    qmda_correlation            char(12),
    qmda_authid                 char(8),
    qmda_plan                   char(8),
    client_platform             char(18),
    client_application          char(20),
    client_authid               char(8),
    account_string#             smallint,
    account_string              varchar(200)
  )
 IN TTSD?ttsvrm?.TTST?ttsvrm?;                                    (4)
/*
//CREATE.SYSTSIN  DD  *
  DSN  SYSTEM(?dsn?)                                              (3b)
    RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA?dv?) -                      (5)
         LIBRARY('?db2hlq?.RUNLIB.LOAD')                          (2c)
    BIND PACKAGE(TTS?ttsvrm?) QUALIFIER(TTS?ttsvrm?) -
         MEMBER(TTSCAUD) -
         LIBRARY('?ttshlq?.TTSDBRM') -
         ISOLATION(CS)  RELEASE(COMMIT)  -
         ACTION(REPLACE)
    BIND PACKAGE(TTS?ttsvrm?) QUALIFIER(?sysibm?) -               (6a)
         MEMBER(TTSLOBJS) -
         LIBRARY('?ttshlq?.TTSDBRM') -
         ISOLATION(CS)  RELEASE(COMMIT)  -
         ACTION(REPLACE)
    BIND PACKAGE(TTS?ttsvrm?) QUALIFIER(?sysibm?) -               (6b)
         MEMBER(TTSIVP) -
         LIBRARY('?ttshlq?.TTSDBRM') -
         ISOLATION(CS)  RELEASE(COMMIT)  -
         ACTION(REPLACE)
    BIND PLAN(TTSM?ttsvrm?) QUALIFIER(TTS?ttsvrm?) -
         PKLIST(TTS?ttsvrm?.*) -
         VALIDATE(BIND) -
         ISOLATION(CS) -
         ACQUIRE(USE)  RELEASE(COMMIT) -
         ACTION(REPLACE) RETAIN
```

```
BIND PLAN(TTSP?ttsvrm?) ACTION(REPLACE) -
          PKLIST(TTS?ttsvrm?.TTSCAUD, TTSRLX?ttsvrm?.*) -
          ISOLATION(CS) QUALIFIER(TTS?ttsvrm?)
     BIND PLAN(TTSI?ttsvrm?) ACTION(REPLACE) -
          PKLIST(TTS?ttsvrm?.TTSIVP) -
          ISOLATION(CS)
  END
/*
//BIND.SYSTSIN  DD  *
TSOLIB ACT DA('?ttshlq?.TTSLOAD')
TTSIBIND DB2(?dsn?) TTSVRM(?ttsvrm?) OWNER(?authid?)            (3c) (1b)
/*
```

Figure 20.7 Defining the Thread/SERIES to a DB2 subsystem

(1a) and (1b) Specify the authorization ID used to GRANT DB2 authorizations for `EXECUTE ON PLAN TTS?ttsvrm?C TO PUBLIC` for use with the Thread/SERIES VIEW Facility.

Also, the DB2 BIND PACKAGE and PLAN OWNER parameter is set to authorization ID.

(2a), (2b) and (2c) Define the names of the system load libraries associated with the target DB2 subsystem. These may include the `SDSNEXIT`, `SDSNLOAD` and `RUNLIB.LOAD` datasets.

(3a), (3b) and (3c) Specify the name of the target DB2 subsystem in which the Thread/SERIES packages and plans will be bound and in which it's THREAD_AUDIT table will be created.

(4) Verify the name of the database and tablespace in which the THREAD_AUDIT table will reside.

(5) Specify the two digits of the version and release level of the target DB2 subsystem. DSNTIA?dv? (4) identifies the plan name assigned for the IBM supplied source module DSNTIAD that was preprocessed, assembled, link edited and bound as part of DB2 subsystem installation.

(6a) and (6b) Specify the owner of the SYSCOLUMNS table that the Thread/SERIES IVP should reference. This can be either the real catalog table owned by AuthID SYSIBM or some shadow copy. ***Executable code for the Thread/SERIES IVP uses unqualified SQL -- so this parameter must be specified.***

20.9.2 GRANT required DB2 Authorizations

The various Thread/SERIES components require the following DB2 authorizations:

- All users of the Thread/STOPPER Dialog as well as users of the Thread/STOPPER Batch and Console Facilities must be GRANTed execute authority on the Thread/STOPPER DB2 application plan named TTSP?ttsvrm?. AuthIDs that will submit the Thread/SENTRY jobstream should possess execute authority on the Thread/SENTRY plan named TTSM?ttsvrm?. Section 20.10.3 discusses special considerations for running the Thread/STOPPER Console Facility and/or Thread/SENTRY as started tasks.

- All AuthIDs that run either Thread/STOPPER or Thread/SENTRY must possess MONITOR2 authority.

- Those AuthIDs which may issue the -TERM UTILITY, -CANCEL DDF THREAD and/or -CANCEL THREAD commands through any Thread/SERIES component require either SYSOPR, SYSCTRL or SYSADM authority.

- AuthIDs authorized to examine the audit trail of canceled threads maintained by Thread/SERIES components must be GRANTed the SELECT privilege on the DB2 table named **owner**.THREAD_AUDIT, where **owner** is the table owner specified in the DDL, as described in Section 20.9.1.

20.10 Customize the Thread/SERIES jobs and catalogued procedure

This step prepares the jobs that invoke Thread/SENTRY, as well as the Thread/STOPPER Batch, Console and Audit Facilities. In addition, you customize a generalized Thread/SERIES catalogued procedure in this step that Thread/SENTRY as well as the various Thread/STOPPER facility jobstreams will invoke to run their respective programs.

Setup for the Thread/SERIES catalogued procedure -- along with the JCL for the Thread/STOPPER Console and Audit Facilities -- is performed once in this step as part of installation. In contrast, JCL for the Thread/STOPPER Batch Facilities is described and illustrated separately in Chapter 4 since each Batch Facility request requires a job. Lastly, the JCL with which to run Thread/SENTRY (as a batch job or started task) is described in Chapter 11.1.1.

20.10.1 Edit the TTSPROC catalogued procedure

The TTSPROC procedure defines common JCL used to run several Thread/SERIES programs. It is distributed as member TTSPROC of the TTSCNTL library (the Thread/SERIES dataset whose low level qualifier is TTSCNTL). Once customized, we recommend that you copy TTSPROC into one of your site's catalogued procedure libraries. Alternatively, TTSPROC can be used as an instream procedure. Lastly, the TTSCNTL dataset can be defined as a JCL procedure library (on a job basis) through a JCLLIB statement or some functionally equivalent dynamic PROCLIB facility.

The various Thread/SERIES programs can maintain connections to multiple DB2 subsystems simultaneously -- ***even DB2 subsystems at different version and release levels.*** To support concurrent connections to multiple DB2 subsystems, each DB2 subsystem requires its own file allocation which must include a special library of CAF load modules that are supplied with Thread/SERIES. These CAF load modules must be concatenated ***ahead*** of their respective DB2 system libraries -- as described and illustrated in Figure 20.8. In addition, each DB2 subsystem being monitored should be allocated a discrete file to which Thread/SENTRY can write trace and diagnostic information pertaining to that subsystem.

The numbers in parentheses to the right of the JCL statements in Figure 20.8 correspond to the numbered, annotating paragraphs which follow the figure. Be sure to edit member TTSPROC with CAPS ON since it contains comments in lowercase.

```
//TTSPROC  PROC TTSHLQ='?ttshlq?',                                  (1)
//         SOUT=*,                      SYSOUT class               (2)
//         ROUT=*,                      Internal reader class      (2)
//         PROG=,                       Thread/SERIES program name (3)
//         PARMS=                       program parameters
//TTSPROC  EXEC PGM=&PROG,REGION=0M,PARM='&PARMS'
//STEPLIB  DD  DISP=SHR,DSN=&TTSHLQ..TTSLOAD
//TTSCNTL  DD  DISP=SHR,DSN=&TTSHLQ..TTSCNTL
//TTSMAPS  DD  DISP=SHR,DSN=&TTSHLQ..TTSMAPS
//TTSPARM  DD  DISP=SHR,DSN=&TTSHLQ..TTSCNTL(TTSPARM)
//TTSRDR   DD  SYSOUT=(&ROUT,INTRDR),DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//TTSOUT   DD  SYSOUT=&SOUT
//TTSTRACE DD  SYSOUT=&SOUT             /* Snaps and traces
//*
//ISPMLIB  DD  DISP=SHR,DSN=&TTSHLQ..TTSMLIB
//ISPPLIB  DD  DISP=SHR,DSN=&TTSHLQ..TTSPLIB
//SYSTSPRT DD  SYSOUT=&SOUT,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSTSIN  DD  DUMMY,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//*--------------------------------------------------------------------
//*
//*        Define a concatenated set of load module libraries for
//*        each DB2 subsystem that is to be monitored and controlled
//*        by a Thread/SERIES program.
//*
//*        In each case, the DDname is derived by appending the suffix
//*        LOAD to the DB2 subsystem name.  For example, the DDname
//*        for a DB2 subystem named DB2T becomes DB2TLOAD.  Be sure
//*        to specify the Thread/SERIES library of CAF load modules
//*        intended for use with a particular version and release of
//*        DB2 as the first dataset in the concatenation sequence.
//*                  =====
//*
//*        For example, if the DB2 subystem named DB2T is at
//*        Version 9.1 of DB2, then specify the Thread/SERIES
//*        library of CAF load modules from the dataset whose
//*        low level qualifier is TTSCAF91.  Similarly, if DB2T
//*        is at V10.1 of DB2, then use the dataset named TTSCAF10.
//*
//*--------------------------------------------------------------------
//db91LOAD DD  DISP=SHR,DSN=&TTSHLQ..TTSCAF91             /* DB2 V9.1 (4)
//         DD  DISP=SHR,DSN=db91.sdsnexit
//         DD  DISP=SHR,DSN=db91.sdsnload
//db10LOAD DD  DISP=SHR,DSN=&TTSHLQ..TTSCAF10             /* DB2 V10  (5)
//         DD  DISP=SHR,DSN=db10.sdsnexit
//         DD  DISP=SHR,DSN=db10.sdsnload
//db11LOAD DD  DISP=SHR,DSN=&TTSHLQ..TTSCAF11             /* DB2 V11  (6)
//         DD  DISP=SHR,DSN=db11.sdsnexit
//         DD  DISP=SHR,DSN=db11.sdsnload
//*--------------------------------------------------------------------
//*        Define discrete log/diagnostic files for each DB2 subsystem
//*        Concatenate 'TTSL' with the DB2 SSID to form the DDname
//*--------------------------------------------------------------------
//TTSLdb91 DD  SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)     (7)
//TTSLdb10 DD  SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)     (8)
//TTSLdb11 DD  SYSOUT=&SOUT,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)     (9)

//*--------------------------------------------------------------------
//*        Uncomment these DD names when instructed by the vendor
//*        to gather diagnostic information
//*--------------------------------------------------------------------
//*RAINPUT  DD  DISP=SHR,DSN=&TTSHLQ..TTSCNTL(TTSTRACE)
//*RFATRACE DD  SYSOUT=&SOUT
//*SYSUDUMP DD  SYSOUT=&SOUT
//*TTSDUMP  DD  SYSOUT=&SOUT
```

Figure 20.8 TTSPROC procedure

(1) Specify the Thread/SERIES libraries installed at your site.

(2) Update the output classes for SYSOUT and Internal Reader files as appropriate.

(3) The PROG keyword governs which Thread/SERIES component is executed through the TTSPROC procedure. The following programs are supported:

TTSMON	The Thread/SENTRY monitor program
TTSAUDIT	The Thread/STOPPER Audit Cancellation Success program
TTSB	The Thread/STOPPER Batch Facility program
TTSCON	The Thread/STOPPER Console Facility program

(4) The TTSPROC procedure must be modified to define a concatenated set of load module libraries for each DB2 subsystem that may be referenced by Thread/SERIES components. Each DB2 subsystem you intend to monitor and control must have its own DD name in the TTSPROC procedure and its own set of load libraries.

In each case, the DD name is derived by appending the suffix LOAD to the DB2 subsystem name. For example, the DD name for a DB2 subsystem named DB2T becomes DB2TLOAD.

Figure 20.8 illustrates how the DD name DB91LOAD defines the set of load module libraries associated with a DB2 Version 9.1 subsystem named DB91. In addition, the Thread/SERIES library of CAF load modules intended for use with DB2 Version 9.1 subsystems (dataset ?ttshlq?.TTSCAF91 in this example) must be the first dataset in the concatenation sequence.

(5) Allocate the load libraries associated with the DB2 Version 10 subsystem named DB10 to the DD name DB10LOAD. The DD name is derived by appending the suffix LOAD to the DB2 subsystem name (DB10). In addition, the Thread/SERIES library of CAF load modules intended for use with DB2 Version 10 subsystems (dataset ?ttshlq?.TTSCAF10 in this example) must be the first dataset in the concatenation sequence.

(6) Allocate the load libraries associated with the DB2 Version 11 subsystem named DB11 to the DD name DB11LOAD. The DD name is derived by appending the suffix LOAD to the DB2 subsystem name (DB11). In addition, the Thread/SERIES library of CAF load modules intended for use with DB2 Version 11 subsystems (dataset ?ttshlq?.TTSCAF11 in this example) must be the first dataset in the concatenation sequence.

(7) Allocate a discrete output file for the DB2 subsystem named DB91 to which Thread/SENTRY can write trace and diagnostic data. The DD name TTSLDB91 is derived by concatenating the literal string 'TTSL' with the name of the DB2 subsystem being monitored (DB91). These trace datasets must be allocated with a record format of FBA, a logical record length of 121 and a block size of 1210.

(8) Allocate a discrete output file for the DB2 subsystem named DB10 to which Thread/SENTRY can write trace and diagnostic data. The DD name TTSLDB10 is derived by concatenating the literal string 'TTSL' with the name of the DB2 subsystem being monitored (DB10). These trace datasets must be allocated with a record format of FBA, a logical record length of 121 and a block size of 1210.

(9) Allocate a discrete output file for the DB2 subsystem named DB11 to which Thread/SENTRY can write trace and diagnostic data. The DD name TTSLDB11 is derived by concatenating the literal string 'TTSL' with the name of the DB2 subsystem being monitored (DB11). These trace datasets must be allocated with a record format of FBA, a logical record length of 121 and a block size of 1210.

20.10.2 Edit the JCL to invoke the Thread/STOPPER Console Facility

Member TTSRCON of the TTSCNTL library contains JCL to invoke the Thread/STOPPER Console Facility. Add a valid JOB statement in order to submit the Thread/STOPPER Console Facility as a standard jobstream. Alternatively, place member TTSRCON in one of your installation's catalogued procedure libraries in order to run the Console Facility as a started task.

The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow Figure 20.9. Be sure to edit member TTSRCON with CAPS ON since it contains comments in lowercase.

```
//jobname  JOB ......                                                  (1)
//TTSCON   EXEC TTSPROC,    Invoke the general Thread/SERIES procedure
//         PROG=TTSCON,     Run the Thread/STOPPER Console Facility pgm
//         PARMS='ORIGIN(WTOR)'     Specify optional parameters         (2)
```

Figure 20.9 TTSRCON procedure

(1) Define a valid job statement if the Thread/STOPPER Console Facility is to be submitted as a batch job. Alternatively, the Console Facility can run as a started task. In this case, no JOB statement is needed but this procedure (TTSRCON) and its companion TTSPROC must reside in one of your site's catalogued procedure libraries.

(2) Specify optional parameters for the Console Facility through the PARMS parameter. Chapter 3 describes the run-time parameters common to both the Thread/STOPPER Batch and Console Facilities. In contrast, Chapter 4 describes several additional parameters unique to the Batch Facility while Chapter 5 describes several additional parameters unique to the Console Facility.

20.10.3 Running Thread/SENTRY and the Thread/STOPPER Console Facility as Started Tasks

The DB2 authorization ID associated with the started tasks for Thread/SENTRY and the Thread/STOPPER Console Facility must have the privilege to execute their respective DB2 application plans. Thread/SENTRY makes use of plan TTSM?ttsvrm? while the various Thread/STOPPER components make use of plan TTSP?ttsvrm?. Since a started task has no JOB statement, you have no opportunity to define the DB2 primary authorization ID for the started task address space through the USER and GROUP parameters of the JOB statement. The DB2 authorization ID for a started task is established through different means.

The DB2 primary authorization ID for the two started tasks ***defaults*** to the unknown AUTHID defined on the DB2 installation panel DSNTIPP. You can GRANT this default user ID the EXECUTE PLAN privilege for the Thread/STOPPER and Thread/SENTRY plans. Alternatively, you can grant EXECUTE PLAN authority for plans TTSP?ttsvrm? and TTSM?ttsvrm? to PUBLIC.

A more granular approach, however, is to explicitly define a RACF User ID and Group Name for the Thread/STOPPER and Thread/SENTRY started task address spaces. To do so, you need to add the Thread/STOPPER and Thread/SENTRY started procedures to the RACF started procedures table ICHRIN03 as illustrated below in Figure 20.10. Then, assemble and link edit the ICHRIN03 load module. Lastly, perform an IPL with the CLPA option (or an MLPA that specifies the ICHRIN03 load module) so these changes take effect.

```
DC  CL8'TTSRCON'       name of the Thread/STOPPER Console
                       Facility procedure
DC  CL8'ttsrcon_id'    This RACF userid becomes the initial
                       DB2 primary authorization ID
DC  CL8'groupname'     RACF Group name
DC  X'00'              No privileged attribute
DC  XL7                Reserved bytes

DC  CL8'TTSRMON'       name of the Thread/SENTRY procedure
DC  CL8'ttsrmon_id'    This RACF userid becomes the initial
                       DB2 primary authorization ID
DC  CL8'groupname'     RACF Group name
DC  X'00'              No privileged attribute
DC  XL7                Reserved bytes
```

Figure 20.10 Adding entries to module ICHRIN03

The RACF user ID you specify above as *ttsrcon_id* for the Thread/STOPPER Console Facility procedure must be granted EXECUTE PLAN authority on application plan TTSP?ttsvrm?. Similarly, the user ID *ttsrmon_id* you define for the Thread/SENTRY started task must be authorized to execute plan TTSM?ttsvrm? as illustrated below:

```
GRANT EXECUTE ON PLAN TTSP?ttsvrm? TO ttsrcon_id
GRANT EXECUTE ON PLAN TTSM?ttsvrm? TO ttsrmon_id
```

The RACF IDs associated with the two started tasks also require the MONITOR and DB2 command authorities described in Section 20.9.3.

20.10.4 Edit the Job that invokes the Thread/STOPPER Audit Facility

Member TTSRACS of the TTSCNTL library contains JCL to invoke the Thread/STOPPER Audit Facility. The Audit Facility ensures that complete details about the cancellation of a particular thread are recorded in the Thread_Audit table, even when cancellation takes an indefinite period of time to complete successfully.

The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow Figure 20.11. Be sure to edit member TTSRACS with CAPS ON since it contains comments in lowercase.

```
//&zuser.T  JOB ......                                              (1)
//TTSRACS   EXEC TTSPROC,    Invoke the general Thread/SERIES procedure
//          PROG=TTSAUDIT    Run the Audit Cancellation Success program
//TTSIN     DD  *            Input stream built dynamically at run-time   (2)
```

Figure 20.11 TTSRACS procedure

(1) Define a valid JOB statement. The Thread/SERIES routines that tailor and submit this job substitute a value from the amper variable &ZUSER. For example, suppose the Thread/STOPPER Dialog is run by a TSO user whose ID is RAI006. In this case, Thread/SERIES converts the jobname &ZUSER.T into the jobname RAI006T.

(2) No input need be specified for the TTSIN DD statement. The various Thread/STOPPER programs that invoke the Audit Facility (i.e. the Thread/STOPPER Dialog as well as the Thread/STOPPER Batch and Console facilities) build this input stream for the Audit Facility dynamically.

20.11 Configuring Thread/SENTRY E-mail Notification

This section describes the steps necessary to enable the Thread/SENTRY e-mail notification feature.

We start by defining some concepts and terminology applicable to e-mail systems. Thread/SENTRY e-mail notification uses the Simple Mail Transport Protocol (SMTP) and Post Office Protocol Version 3 (POP3). Both protocols are implemented in what we term the *Mail Server*. The term *Mail Client* refers to a software component that requests outbound message delivery from the Mail Server (because it cannot send messages itself).

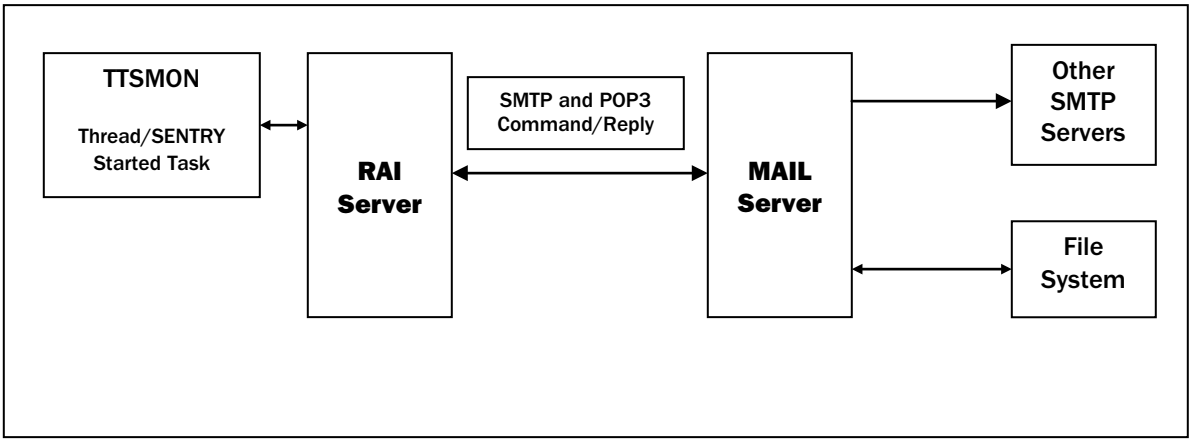


Figure 20.12 Thread/SENTRY e-mail notification environment

In Figure 20.12, TTSMON is the name of the Thread/SENTRY started task and the RAI Server started task (at Version 4.3.3 or later) acts as a Mail Client. The following describes the e-mail delivery sequence:

- TTSMON determines that an e-mail notification must take place
- TTSMON prepares an e-mail message to be sent to the list of recipients specified by the `NOTIFY_LIST` directive and sends the message text and recipient list to the RAI Server.
- The RAI Server is configured with SMTP and POP3 parameters that allow it to establish a TCP/IP connection with the Mail Server. These SMTP and POP3 parameters are described in the RAI Server Installation and Operation Guide (RAI Publication RSV-001). The RAI Server sends message text for ultimate delivery to a target e-mail address. If the specified mailbox resides within the domain of the Mail Server, then the message is delivered directly. Otherwise, the Mail Server further relays the message towards the target domain.

In order to configure Thread/SENTRY e-mail notification, the following requirements must all be satisfied:

- TCP/IP must be active on the system where both Thread/SENTRY and the RAI Server are running

- The RAI Server started task must be active and must have an OMVS segment defined in accordance with your installation's security software standards, as in the following RACF example:

```
AU RAIO DFLTGRP(gggggggg) OWNER(nnnnnnnn) PASSWORD(pppppppp) -
OMVS(UID(0) HOME(/))
```

- Access to an SMTP server must be available. The SMTP Server may reside within a z/OS, Windows or UNIX systems, so long as it conforms to the minimal implementation standard specified in the following documents:

  RFC 2821 - Simple Mail Transfer Protocol
  RFC 1939 - Post Office Protocol - Version 3 (POP3)

- The SMTP Server must support e-mail domain routing if e-mail notifications are to be sent to users that reside in domains other than those adjacent to the SMTP server.

- Presently the RAI Server acting as a Mail Client does not support TCP/IP Secure Sockets Layer (SSL). As such, the designated Mail Server must allow non-SSL connections.

- A good solution is to use the Mail Server that resides within the same private sub-network as the z/OS system. This ensures the privacy of an unsecured SMTP connection established between the RAI Server (Mail Client) and the designated Mail Server.

Some SMTP servers only allow connections to 'trusted' users, such as connections that originate from the same Internet sub-network where SMTP server is installed, or by some other private protocol criteria. Examples of such SMTP servers include: cable modem and DSL providers, yahoo.com, etc. Thread/SENTRY e-mail notification will not operate with such servers.

To complete the RAI server configuration for electronic mail delivery, the following information is required:

SMTP Server IP Address	e.g.	243.150.20.247
SMTP Server name	e.g.	smtp.domain.com
POP3 user ID and password	e.g.	ttsadmin / secret123
SMTP TCP/IP port number	25	(default)
POP3 TCP/IP port number	110	(default)
e-mail return address	an Internet address to which undelivered e-mail notifications will be sent (e.g. ttsadmin@domain.com)	
e-mail CC: address	This is an optional e-mail address to which all Thread/SENTRY notifications will be sent (e.g. john_manager@domain.com)	

20.12 Define the VTAM application major node used by the Thread/SERIES Components

```
         VBUILD TYPE=APPL             APPLICATION MAJOR NODE
*
*        DEFINE VTAM APPLICATIONS AS SECONDARY PROGRAMMABLE OPERATORS
*        TO BE USED BY RAI (RELATIONAL ARCHITECTS INTL) PRODUCTS
*
RAIAPLØ1 APPL   AUTH=(NVPACE,SPO,ACQ),PRTCT=RAI
RAIAPLØ2 APPL   AUTH=(NVPACE,SPO,ACQ),PRTCT=RAI
RAIAPLØ3 APPL   AUTH=(NVPACE,SPO,ACQ),PRTCT=RAI
```

Figure 20.13 VTAM APPL definition

This step defines the application major node through which Thread/SERIES communicates with VTAM. These definitions enable Thread/SERIES components to execute VTAM commands and obtain solicited command responses. Thus, if a DB2 thread is executing (or is suspended) in VTAM, Thread/SERIES programs can cancel the thread's VTAM sessions and permit the thread to terminate.

Figure 20.13 defines a pool of VTAM applications with secondary programmable operator authority and the ability to acquire nodes. A pool of 3 VTAM APPLs guarantees that up to 3 users can use the VTAM facilities provided by Thread/SERIES -- simultaneously. The underlined operands described here should be reviewed and revised as necessary.

RAIAPLØ1 defines an 8 character name for the VTAM APPL. Any name you specify should consist of a pool name of 6 characters (such as RAIAPL) suffixed by two numeric digits (for example 01, 02 and 03). The total length of each name must be 8.

PRTCT defines a password which protects use of the VTAM application. The password value illustrated in Figure 20.13 is RAI.

The definitions illustrated in Figure 20.13 are distributed with Thread/SERIES in member TTSDVTAM of the TTSCNTL library. This member should be copied into one of the datasets allocated to VTAMLST which define network definitions and VTAM start options.

If you change the password or specify a pool name other than RAIAPL, be sure to generate an updated set of Thread/SERIES system defaults as described in Section 20.14.

Lastly, these VTAM application names should be included in the ATCSTRxx member of your VTAMLST library so they will be activated *automatically* when VTAM is started. Alternatively, you can use VTAM commands like the following to manually activate the VTAM applications used by the Thread/SERIES components:

```
V NET,ACT,ID=RAIAPL01
V NET,ACT,ID=RAIAPL02
V NET,ACT,ID=RAIAPL03
```

20.13 Edit the TTSPAL catalogued procedure (Optional)

The TTSPAL procedure described in this section defines a common JCL procedure used to assemble and link edit various Thread/SERIES source modules that include the following:

Member name of the TTSCNTL library		Description
Source module	JCL used to build the source module	
TTS$TGI	TTSJTGI	Thread/SENTRY Table of Group ID definitions (described in AppendixG)
TTS$TNA	TTSJTNA	Thread/SERIES Table of No Action criteria (described in Appendix F)
TTS$TNM	TTSJTNM	Thread/SERIES Table of Notify Messages (described in Appendix C)
TTS$TNS	TTSJTNS	Thread/SERIES Table of Non Standard processing (described in Appendix E)
TTS$TSD	TTSJTSD	Thread/SERIES Table of System Defaults (described in Section 20.14)
TTS$TSM	TTSJTSM	Thread/SERIES Table of site Written Messages (described in Appendix C)
TTS$TXR	TTSJTXR	Thread/SERIES Table of Exit Routines (described in Appendix E)
		Additional, site written exit routines (discussed in Appendix E)

The TTSPAL procedure is distributed as a member of the TTSCNTL library (the Thread/SERIES dataset whose low level qualifier is TTSCNTL). The aforementioned TTSJxxx members of the TTSCNTL library define the TTSCNTL dataset as a JCL procedure library through a JCLLIB statement. The TTSPAL symbolic parameters are specified via JCL in each TTSJxxx member which invokes it.

The TTSPAL procedure, illustrated in Figure 20.14, assembles and link edits the Assembler Language source module identified by the MEMBER parameter to produce a load module with reusable, reenterable and refreshable attributes. (For example, Section 20.14 describes the jobstream to assemble and link edit member TTS$TSD to produce a customized Table of System Defaults).

The numbers in parentheses to the right of the JCL statements correspond to the numbered, annotating paragraphs which follow Figure 20.14. Be sure to edit member TTSPAL with CAPS ON since it contains comments in lowercase.

```
//TTSPAL   PROC HLQ=&HLQ,       <--- 1st level dataset qualifier(s)
//         UNIT=SYSDA,          <--- Unit name for DASD data sets
//         ASMPGM=ASMA9Ø,       <--- Name of the system assembler      (1)
//         PRODUCT=TTS,         <--- Thread/SERIES component code
//         MEMBER=&MEMBER       <--- Source module to assemble
. . .
```

(1) Review, and if necessary revise, the name of the system assembler installed at your site. Typically, this will be IEV90 or ASMA90.

Figure 20.14 Thread/SERIES Assembly Link Edit Procedure

20.14 Update vendor supplied defaults (Optional)

This step lets you update the Thread/SERIES defaults supplied by Relational Architects. This step is ***not required*** unless you changed the VTAM application names and passwords described in Section 20.12. However, if you ***do*** perform this step, then you must first customize the procedures named TTSPAL and RAIAL, as described in Section 20.13.

To assemble and link edit a new Thread/SERIES Table of System Defaults (load module TTS$TSD), first review and revise the values supplied in member TTS$TSD of the TTSCNTL library (the source input to the assembler). Operands of the TTS#TSD macro (which resides in the TTSMACS dataset) define defaults for the various Thread/SERIES components.

Next, edit and submit the JCL in member TTSJTSD of the TTSCNTL library as illustrated in Figure 20.16 Part 1. The numbers in parentheses in the right margin correspond to the numbered, annotating paragraphs which follow the second part of the figure.

```
//JOBNAME  JOB (ACCOUNT),.                                          (1)
// SET TTSHLQ='?ttshlq?'                                            (2)
//*
//PROCLIB JCLLIB ORDER=&TTSHLQ..TTSCNTL
//*
//TTS$TSD  EXEC TTSPAL,         <- Invoke the procedure TTSPAL
//         MEMBER=TTS$TSD,      <- Table of system defaults         (3)
//         HLQ=&TTSHLQ
```

Figure 20.16 - Part I Assemble and linkedit Thread/SERIES Defaults

(1) Provide a valid JOB statement.

(2) the value of ?ttshlq? was replaced during the library restore process with the high level qualifier for the Thread/SERIES product libraries (as described in the Section 20.5).

(3) Member TTS$TSD of the TTSCNTL dataset supplies input to the assembler.

Figure 20.16 - Part II illustrates member TTS$TSD of the TTSCNTL dataset. Operands of the TTS#TSD macro define defaults for the Thread/SERIES components.

```
TTS$TSD  TTS#TSD ABEND=0222,       ABEND CODE - 4 HEX DIGITS          +(1)
              AUDIT_DATA_WHEN=ACTION, AUDIT IFI DATA DURING ACTION    +
              AUTO_FORCE=YES,      FORCE ANY LOCAL THREAD IN APPL.    +
              CCSID=37,            EBCDIC US ENGLISH NO EURO SUPPORT  +
              CONSOLE=NO,          ACTIVATE AN MCS CONSOLE PROCESS    +
              DEBUG_POLICY='__',   THREAD/SENTRY Debug policy ID prefix+
              DESCDE=00,           THREAD/SENTRY DESCRIPTOR CODE      +
              DUMPDD=TTSDUMP,      DUMP DDname for Thread/SENTRY      +
              ESTAE=YES,           TRAP ABENDS                        +
              EXITS=TTS$TXR,       EXIT ROUTINE DEFINITIONS           +
              IDLETHD=NO,          DO NOT MONITOR IDLE THREADS        +
              INTERVAL=00010000,   THREAD/SENTRY WAKEUP INTERVAL      +
              MAX_UNEXPECTED_IFCIDS=20, # UNEXPECTED IFCIDS TO REPORT +
              MESSAGE_MODULE=NONE,  NO TABLE OF SITE WRITTEN MESSAGES +
              MMINT=00020000,      MINIMUM / MAXIMUM INTERVAL         +
              MONPLAN=TTSM?ttsvrm?, THREAD/SENTRY PLAN NAME           +
              MSGDISP=TERSE,       BE TERSE IN ISSUING MESSAGES       +
              NONSTD=TTS$TNS,      NON_STANDARD THREAD DEFINITIONS    +
              NOTENBL=ALL,         NOTIFY_ENABLED(ALL)                +
              NOTIFY_INACTIVE_ADMIN=BOTH, Notify inactive admin and   +
              NOTIFY_INACTIVE_USER=BOTH,  user of WARNING and CANCEL  +
              NOTIFY_ON_WARNING=NO, NO NOTIFY IF WARNING IS PENDING   +
              NOTFREQ=10,          NOTIFY_FREQUENCY VALUE             +
              NOTLIST=,            DEFAULT NOTIFY_LIST ID             +
              NOTMAX=3,            NOTIFY_MAXIMUM VALUE               +
              OPMODE=CANCEL,       THREAD/SENTRY OPERATIONAL MODE     +
              OPW=RAI,             OPERATOR PASSWORD                  +
              OUTDD=TTSOUT,        THREAD/SENTRY OUTPUT DDNAME        +
              PENDING=00010000,    PENDING_INTERVAL(60)               +
              POOL=RAIAPL,         BASE FOR POOL OF VTAM ACB'S        +
              POLWORK=250,         LENGTH OF POLICY WORK AREA         +
              REASON=00DB2000,     REASON CODE - 8 HEX DIGITS         +
              RESET_WARNING=NO,    NO RESET WARNING IF VIOL IS NO MORE +
              ROUTCDE=11,          THREAD/SENTRY ROUTING CODE         +
              RPTDD=TTSRPT,        DDNAME FOR THE REPORTING FUNCTION  +
              SNAPDD=TTSTRACE,     SNAP/TRACE DDNAME FOR THREAD/SENTRY +
              STOPPLAN=TTSP?ttsvrm?, DB2 PLAN NAME FOR THREAD/STOPPER +
              SUM_PARALLEL_ELAPSED=NO,  DON'T COMBINE ELAPSED TIME    +
              SYSTEM_ABEND=YES,    SYSTEM ABEND UPON FORCE            +
              TIME_INTERVAL=NEW,   PERMIT START_TIME > END_TIME       +
              VIOLWORK=50,         LENGTH OF VIOLATION WORK AREA      +
              WAIT=00000500,       WAIT TIME EXPRESSED AS HHMMSSTH    +
              WTOR=YES,            ISSUE WTOR (YES OR NO)             +
              WTO_HRDCPY=NO,       QUEUE WTO FOR HARD COPY ONLY? Y/N  +
              DOC=NO               PRINT NO MACRO DOCUMENTATION
PUNCH    ' IDENTIFY TTS$TSD(''TTS$TSD - TTS?ttsvrm?'')'
         END
```

Figure 20.16 - Part II Assemble and linkedit Thread/SERIES Defaults

(1) The operands of the TTS#TSD macro define the individual Thread/SERIES defaults. These operands, their corresponding vendor supplied defaults, and their meanings are described below:

`ABEND=0222` — 4 Hex digits that define the abend code with which Thread/SERIES components deliberately abend the MVS task associated with a DB2 thread. ***The first digit must be zero***. Please note, that the parameter `SYSTEM_ABEND` defines whether or not to terminate these tasks with a system or user abend. The default is to issue a System 222 abend.

`CCSID=37` — Identifies the Unicode conversion CCSID with which to convert DB2 IFI data into EBCDIC format.

`EXITS=TTS$TXR` — specifies the name of the Thread/SERIES Table of Exit Routine definitions.

`NONSTD=TTS$TNS` — specifies the name of the Thread/SERIES Table of Non_Standard thread definitions.

`OPW=RAI` — Password for the VTAM applications used by Thread /SERIES components.

`POOL=RAIAPL` — Base name for the pool of VTAM ACB's. This value must be six characters long

`REASON=00DB2000` — 8 Hex digits that define the reason code associated with the system abend.

`STOPPLAN=TTSP?ttsvrm?` — The name of the DB2 application plan used by the various Thread/STOPPER components.

`SUM_PARALLEL_ELAPSED=NO` — governs whether or not Thread/SERIES components should summarize elapsed times associated with parallel threads. Section 10.4 of this publication describes this parameter in detail.

`SYSTEM_ABEND=YES` — directs Thread/SERIES components to deliberately abend the MVS task associated with a DB2 thread via a system abend. Specify `SYSTEM_ABEND=NO` to terminate these tasks via a user abend. The actual abend code is defined by the ABEND parameter.

`WAIT=00000500` — defines how long the Thread/STOPPER Dialog should wait after a thread termination request before determining whether the thread is gone or the cancellation method needs to be escalated. Wait time is expressed as HHMMSSTH.

The following operands of the TTS#TSD macro define defaults for Thread/SENTRY. Chapter 10 describes these operands in greater detail and describes how they may be overridden. The set of Thread/SENTRY operands,

their corresponding vendor supplied defaults, and their meanings are described below:

`AUDIT_DATA_WHEN=ACTION` Thread/SENTRY processes site defined policy statements (e.g. LIMIT) in two-phases. The first phase identifies the DB2 threads that violate one or more site policies. Thread/SENTRY acts upon these violations during the second phase. Thread/SENTRY obtains IFI data during both phases independently. The parameter `AUDIT_DATA_WHEN` governs which IFI data should be recorded in the THREAD_AUDIT table. `AUDIT_DATA_WHEN=VIOLATION` directs Thread/SENTRY to record the IFI data collected during the first pass (`VIOLATION`) phase of processing. Alternatively, `AUDIT_DATA_WHEN=ACTION` directs Thread/SENTRY to record the IFI data collected during the second `ACTION` phase of processing into the Thread_AUDIT table. The valid setting for this parameter is `VIOLATION` or `ACTION`.

`AUTO_FORCE=YES` specifies whether or not Thread/SENTRY should use ACTION(FORCE) instead of ACTION(CANCEL) to remove a local thread that is currently executing within the application rather than DB2 (and as such, no SQL statement is being executed). `AUTO_FORCE=NO` directs Thread/SENTRY to CANCEL such threads using the DB2 `-CANCEL THREAD` command. `AUTO_FORCE=YES` directs Thread/SENTRY to FORCE such threads to terminate.

`CONSOLE=NO` governs whether or not Thread/SENTRY should activate an MCS console and listen for selected messages. Specify MCS_CONSOLE(YES) to enable Thread/SENTRY to automatically detect active log full conditions in one or more monitored DB2 subsystem(s). The default is MCS_CONSOLE(NO).

`DEBUG_POLICY='__'` directs Thread/SENTRY to externalize the DB2 IFI and z/OS WLM data for any violated policy whose name starts with the characters '__'.

`DESCDE=ØØ` Thread/STOPPER descriptor CODE

`DUMPDD=TTSDUMP` specifies the destination of an optional Thread/SENTRY formatted dump. For example, add "//TTSDUMP DD SYSOUT=*" to your Thread/SENTRY execution JCL to direct Thread/SENTRY to write a formatted dump of its internal control blocks to the TTSDUMP file.

`ESTAE=YES`	By default -- TRAP abends
`IDLETHD=NO`	governs whether on not Thread/SENTRY monitors inactive DDF threads (i.e. IDLE threads) for policy violations. Thread/SENTRY incurs significant overhead to obtain information about idle threads so the default is IDLE_THREAD(NO).
`INTERVAL=00010000,`	Thread/SENTRY wakeup interval
`MAX_UNEXPECTED_IFCIDS=20,`	Thread/SENTRY issues IFI READS requests for IFCIDs 0148 and 0150 only. However, DB2 occasionally returns IFCIDs records *other than* those requested. The parameter `MAX_UNEXPECTED_IFCIDS` specifies the maximum number of unexpected IFCIDs Thread/SENTRY should report before turning this diagnostic off.
`MESSAGE_MODULE=NONE,`	Specifies that no Thread/SENTRY table of site written messages is defined. Chapter 10 of this publication manual describes the MESSAGE_MODULE operand of the Thread/SENTRY MONITOR statement with which this setting can be overriden.
`MMINT=00020000`	Minimum / maximum interval
`MONPLAN=TTSM?ttsvrm?`	Thread/SENTRY monitor plan name
`MSGDISP=TERSE`	Be terse in issuing messages
`NOTENBL=ALL`	Notifications are enabled for all actions and recipients. This default value should not be changed.
`NOTFREQ=10`	how often logical requests to send notifications and/or insert rows into the Thread_Audit table will be physically honored.
`NOTIFY_INACTIVE_ADMIN=BOTH,`	Any inactive (logged off) administrator is notified of any action. The other valid settings are OFF, WARN and CANCEL. OFF - Notifications are disabled for any inactive administrator; WARN - Notifications are enabled for any inactive administrator for ACTION(WARNING); CANCEL - Notifications are enabled for any inactive administrator for ACTION(CANCEL).
`NOTIFY_INACTIVE_USER=BOTH,`	Any inactive (logged off) user is notified of any action. The other valid settings are OFF, WARN and CANCEL. OFF - Notifications are disabled for any inactive user; WARN - Notifications are enabled for any inactive user for ACTION(WARNING); CANCEL - Notifications are enabled for any inactive user for ACTION(CANCEL).

`NOTIFY_ON_WARNING=NO`	Notifications are suppressed for any pending warning request.
`NOTLIST=`	Default name of NOTIFY_LIST
`NOTMAX=3`	limits the number of notifications that Thread/SENTRY will send to users and administrators for a particular thread. Once the specified maximum is reached, Thread/SENTRY disables further notifications for that thread.
`OPMODE=WARN`	Thread/SENTRY operational mode
`OUTDD=TTSOUT`	Thread/SENTRY output DDname
`PENDING=00010000`	Cancel Pending interval
`POLWORK=250`	default length of policy work area.
`RESET_WARNING=NO`	Maintain a pending warning condition even when a target thread no longer violates the policy that triggered the original warning. Both the `RESET_WARNING` and `NOTIFY_ON_WARNING` defaults must be set to `YES` in order for Thread/SENTRY to examine pending violations and possibly reset them.
`ROUTCDE=11`	Thread/SERIES routing code
`RPTDD=TTSRPT`	DDname for the Thread/SENTRY reporting function
`SNAPDD=TTSTRACE`	Snap/trace DDname for Thread/SENTRY
`TIME_INTERVAL=NEW`	allows the specification of START and END times for a Thread/SENTRY policy that can span a single day Midnight boundary. For example the operands START(11:00) and END(06:00) define an interval which starts at 11 o'clock in the morning of one day and remains in force until 6 o'clock in the morning of the following day.
`VIOLWORK=50`	default length of violation work area
`WTO_HRDCPY=NO`	governs whether or not any WTO's issued by Thread/SENTRY should be queued to hardcopy only.
`WTOR=YES`	governs whether or not Thread/SENTRY should issue a WTOR to which an operator can reply. If the WTOR is suppressed, then the operator must issue MVS MODIFY and STOP commands to communicate with Thread/SENTRY.

Both the assembly and link edit steps of this job should complete with a return code of 0.

20.15 Prepare the ISPF environment for the Thread/STOPPER Dialog

This step prepares the ISPF environment for the Thread/STOPPER Dialog.

The REXX exec TTSRUN manages all Thread/SERIES libraries dynamically *within* ISPF and invokes both the Thread/STOPPER Dialog and Thread Audit View Facility. The TTSRUN exec implements the following functions:

- Allocates the SDSNLOAD and SDSNEXIT libraries associated with each DB2 subsystem to be monitored by the Thread/STOPPER Dialog.
- Dynamically defines the REXX exec and ISPF dialog libraries that the Thread/STOPPER Dialog requires.
- Constructs the Thread/STOPPER parameter list.
- Invokes the Thread/STOPPER Dialog or Audit View Facility.
- Frees the REXX exec and ISPF dialog libraries upon the exit from the Thread/STOPPER dialog or Audit View Facility.

Figure 20.17 illustrates the modifiable section of the TTSRUN exec. You should edit the TTSRUN exec *only* between the lines `=START OF CONFIGURATION=` and `=END OF CONFIGURATION=`. The embedded comments within the modifiable section describe the edits required before it can be executed by Thread/STOPPER Dialog users. The numbers in parentheses in the right margin correspond to the numbered, annotating paragraphs which follow the figure.

```
=========================START OF CONFIGURATION=========================
*  Make the necessary changes within this configuration section as
*  described below.  The configuration section is delineated by the
*  lines which read "START OF CONFIGURATION" and "END OF CONFIGURATION".
*
*  Note:  You can comment out any line within this section be placing
*         an asterisk '*' in column 1.
*
*  Identify the DB2 subsystems to be monitored by Thread/STOPPER
*  using the following pattern:
*
*  DB2 word1 word2 word3 word4
*
*  where:
*    ___________________________________Word1: DB2 subsystem name
*   |     ____________________________Word2: DB2 version and release,
*   |    |                                  two digits, e.g. 11, 10 or 91
*   |    |   _____________________Word3: SDSNEXIT library name used
*   |    |  |                              with the specified subsystem
*   |    |  |                 ____Word4: SDSNLOAD library name used
*   |    |  |                |             with the specified subsystem
*   v    v  v                v
DB2 DB2P 11 DB2P.SDSNEXIT   DSN1110.SDSNLOAD                          (1a)
DB2 DB2T 11 DB2T.SDSNEXIT   DSN1110.SDSNLOAD                          (1b)

*  Use Thread/STOPPER keyword SUBSYSTEM_MODE to specify the
*  Thread/STOPPER subsystem mode:
*
*  S denotes single subsystem mode, in which Thread/STOPPER connects
*    to only one DB2 subsystem at a time.
*  M denotes multiple subsystem mode, in which Thread/STOPPER can
```

```
*    maintain connections to multiple DB2 subsystems simultaneously.
*SUBSYSTEM_MODE S                                                    (2a)
SUBSYSTEM_MODE M                                                     (2b)

*  Use Thread/STOPPER keyword ISPF_SPLIT_SCREEN to specify the
*  Thread/STOPPER ISPF split screen mode:
*
*  E enables ISPF split screen operation
*  D disablesISPF split screen operation
*ISPF_SPLIT_SCREEN E                                                 (2c)
ISPF_SPLIT_SCREEN D                                                  (2d)

*  Use Thread/STOPPER keyword SUBSYSTEM_VIEW to specify
*  the perspective from which the scrollable display of DB2 subsystems
*  should be viewed.
*  C denotes the "classic" subsystem view (from prior releases of
*    Thread/STOPPER) using the TBDISPL panel named TTSTDS (from the
*    TTSPLIB library).
*  D denotes a starting Date and Time view of DB2 subsystems using the
*    TBDISPL panel named TTSTDSD (from the TTSPLIB library).
*  G denotes a Data Sharing Group view of DB2 subsystems using the
*    TBDISPL panel named TTSTDSG (from the TTSPLIB library).
*SUBSYSTEM_VIEW C                                                   (2e)
SUBSYSTEM_VIEW G                                                    (2f)

*  Use Thread/STOPPER keyword TTSHLQ to specify the high level
*  qualifiers of the Thread/SERIES target libraries:
*
TTSHLQ ?ttshlq?                                                     (3)

*  Use Thread/STOPPER keyword TTSQUAL to specify the owner ID of the
*  Thread/SERIES audit DB2 table (named THREAD_AUDIT).  This DB2 table
*  was created by JCL in member TTSJDB2 of the TTSCNTL library.
TTSQUAL TTS?ttsvrm?                                                 (4)

*
*  Use Thread/STOPPER keyword TTSREPO to specify the dataset name of
*  the Thread/SERIES repository KSDS file.
TTSREPO ?ttshlq?.TTSREPO                                            (5)

*  Use Thread/STOPPER keyword RLXEDIT to specify the RLX plan to use
*  with the THREAD_AUDIT VIEW facility.  This DB2 plan was bound by
*  JCL in the TTSJDB2 member of the TTSCNTL library.
RLXEDIT TTS?ttsvrm?E                                                (6)
=========================END OF CONFIGURATION=========================
```

(1a) and (1b) These two sample configuration statements declare two DB2 subsystems (named DB2P and DB2T) to Thread/STOPPER. Replace these two lines with one line for each DB2 subsystem to be monitored by Thread/STOPPER and into which Thread/STOPPER was installed. As such, make sure the job in the TTSJDB2 member of the TTSCNTL library ran successfully for any DB2 subsystem defined in this section.

(2a) through (2f) You can accept the vendor supplied defaults for these optional parameters so there is no need to modify them. The embedded notes describe how to modify these optional parameters. In addition, please male note of the following:

NOTE: *The Thread/STbOPPER Dialog can operate in either of two modes: The first mode* ***(3a)*** *connects to a* ***single*** *DB2 subsystem while the second mode* ***(3b)*** *supports* ***simultaneous*** *connections to* ***multiple***

DB2 subsystems within the local MVS system. When you use Thread/STOPPER in multiple subsystem mode, ***we strongly recommend you avoid ISPF split screen mode (3d).*** *This will prevent the use of other DB2 applications (such as QMF or SPUFI) whose attachment mechanisms do not support concurrent connections to multiple DB2 subsystems. The use of conflicting DB2 attachment mechanisms can cause unpredictable results and system failures.*

The following symbolic variables defined with the Configuration section were automatically *preset* to site specific values you designated during the installation step that was described in the Section 20.5of this manual. The names of these automatically assigned symbolic variables, along with their values and descriptions appear in Table 20.1 (which is also in Section 20.5).

(3) Do *not* alter the `TTSHLQ` parameter.

(4) Do *not* alter the `TTSQUAL` parameter *unless* you changed the default value for `TTSQUAL` within the job in member TTSJDB2 of the TTSCNTL library.

(5) Do *not* alter the `TTSREPO` parameter *unless* you created an alternate KSDS file for the THREAD_AUDIT VIEW repository (via the JCL in member TTSJRDL of the TTSCNTL library or some other method).

(6) Do *not* alter the `RLXEDIT` parameter *unless* you changed the default value `RLXEDIT` within the job in member TTSJDB2 of the TTSCNTL library.

Figure 20.17 TTSRUN EXEC CONFIGURATION section

Once the CONFIGURATION section of the TTSRUN exec is edited, the Thread/STOPPER Dialog can be invoked from within ISPF as illustrated below. Be sure to substitute the high level qualifier of the Thread/SERIES product libraries for the variable `?ttshlq?`:

```
TSO EX '?ttshlq?.TTSEXEC(TTSRUN)' EX
```

20.16 Prepare the Thread/SERIES Audit View Facility (Optional)

You can also invoke the Thread/SERIES Audit View Facility, once the CONFIGURATION section of TTSRUN exec is edited (as described in Section 20.15). The Audit View Facility can be invoked from within ISPF as illustrated below. Be sure to substitute the high level qualifier of the Thread/SERIES product libraries for the variable `?ttshlq?`:

```
TSO EX 'ttshlq.TTSEXEC(TTSRUN)' 'VIEW,DB2P' EX
(1)                              (2)  (3)
```

(1) The `TSO` prefix is not required within ISPF Option 6.

(2) The `VIEW` keyword directs the TTSRUN exec to invoke the Thread/SERIES Audit View Facility, rather than the Thread/STOPPER dialog.

(3) The `DB2P` value following the `VIEW` keyword and comma identifies which DB2 subsystem (and THREAD_AUDIT table) should be accessed by the Audit View Facility. Be sure any DB2 subsystem specified here was defined to the TTSRUN exec as described in ***(1a)*** and ***(1b)*** of Figure **20.17**.

> ***NOTE:*** *The Thread/SERIES Audit View Facility is a dialog application which makes use of the RLX run-time engine. The Audit View Facility also utilizes an application definition which resides in the Relational Architects Repository. This repository dataset is defined and loaded as part of the installation procedure described in Section 20.5 of this chapter. See Appendix Y of this manual if you need to re-create the repository dataset.*

20.17 Thread/SERIES Installation Verification Procedures

This step describes how to verify that Thread/SERIES components are properly installed on one or more DB2 subsystems. The verification procedure involves creating a set of DB2 threads that will be canceled deliberately.

The DB2 threads associated with the Thread/SERIES ***installation verification program*** (IVP) generate a modest level of read-only SQL activity. As such, the IVP serves as a convenient cancellation target for both Thread/STOPPER and Thread/SENTRY. Section 20.17.1 describes how to run the IVP in both batch and foreground modes. Section 20.17.2 describes the installation verification procedure for Thread/SENTRY. Section 20.17.3 discusses the verification procedure for the Thread/STOPPER Dialog.

20.17.1 Prepare to Run the Installation Verification Program

The Installation Verification Program named **TTSIVP** resides in the TTSLOAD library. Its associated plan (**TTSI?ttsvrm?)** was bound as part of job **TTSJDB2** (described in Section 20.9.2). TTSIVP can run in both foreground and background modes.

To run the IVP in batch, the JCL in member TTSRIVP of the TTSCNTL dataset must be updated as illustrated in Figure 20.22.1. As shipped, member TTSRIVP is a single job-step. ***A separate jobstep should be created for each additional DB2 subsystem into which Thread/SERIES components have been installed***. Figure 20.22.1 illustrates a single step job that runs the IVP. Review and update the values that appear in lowercase, underlined type.

```
//JOBNAME JOB (ACCOUNT)
//TTSJIVP  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=?db2hlq?.SDSNLOAD       (1)
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN  SYSTEM(?dsn?)                               (2)
     RUN  PROGRAM(TTSIVP) PLAN(TTSI?ttsvrm?) -     (3)
          LIBRARY('?ttshlq?.TTSLOAD')              (4)
  END
//
```

where

(1) The STEPLIB dataset must specify the names of the DB2 system load libraries associated with the target DB2 subsystem.

(2) The string `?dsn?` must be replaced by the name of the target DB2 subsystem.

(3) Set to the plan name for the Thread/SERIES Installation Verification Procedure (as described in Section 20.5 of this manual).

(4) Set to the name of the TTSLOAD library as described in Section 20.5 of this manual.

Figure 20.22.1

In batch mode, the TTSIVP job externalizes a small SQL query result and then enters a ***wait*** state for 1 minute before it ***resumes*** execution. This precaution avoids unnecessary system activity in case the job is inadvertently left running for an extended period of time.

The TTSIVP program also runs in the TSO foreground which may be convenient when you conduct the verification procedure for the Thread/STOPPER Dialog (as described in Section 20.15.3). From the **DB2I** primary option menu, select the **RUN** option (item 6) to display the DB2I RUN time panel, illustrated in Figure 20.22.2. Specify the three underlined items. The dataset name should specify the fully qualified name of the TTSLOAD library and TTSIVP member. Specify **TTSI?ttsvrm?** as a plan name and request to run the program in the TSO **foreground**.

```
_

Enter the name of the program you want to run:
  1  DATA SET NAME ===> '?ttshlq?.TTSLOAD(TTSIVP)'
  2  PASSWORD .... ===>     (Required if data set is password protected)

Enter the following as desired:
  3  PARAMETERS .. ===>
  4  PLAN NAME ... ===> TTSI?ttsvrm?    (Required if different from program name)
  5  WHERE TO RUN  ===> FOREGROUND (FOREGROUND, BACKGROUND, or EDITJCL)

 NOTE : Information for running command processors is on the HELP panel.
 PRESS: ENTER to process     END to exit     HELP for information
```

Figure 20.22.2

When operating in the foreground, the IVP issues the following prompt:

```
Thread/SERIES Installation Verification Program Starting
Thread/SERIES IVP Foreground Execution
TTSIVP - Enter X to exit or anything else to continue
```

In both batch and foreground modes, the IVP writes out a small answer set in each cycle. When the IVP runs in the foreground, your TSO session receives a thread cancellation message when you use Thread/STOPPER or Thread/SENTRY to cancel its DB2 thread. Your TSO session remains active after thread cancellation.

20.17.2 Verify Thread/SENTRY Installation

To verify installation of Thread/SENTRY, first tailor the statements within member **TTSIN** of the TTSCNTL library. These statements appear in Figure 20.22.3. Member TTSIN defines the policies to be enforced by Thread/SENTRY. The predefined policy is designed to cancel the thread associated with the IVP plan named TTSI?ttsvrm? as soon as it issues more than 10 SQL statements, a limit the IVP application exceeds on every cycle.

The topic of defining Thread/SENTRY policies is discussed at length in Chapter 10. For now, simply replace the underlined items illustrated in Figure 20.21.3 with the names of the DB2 subsystems in which Thread/SENTRY has been installed.

```
MONITOR
  SSID(dsn1,dsn2,dsn3)          (1)
  WAKEUP_INTERVAL(30)
  MINMAX_INTERVAL(60)

LIMIT
  PID(POLICY1)
  SSID(dsn1,dsn2,dsn3)          (2)
  PLAN(TTSI?ttsvrm?)
  MAX_SQL(10)
  ACTION(CANCEL)
//
```

where

(1) Specify the list of DB2 subsystems in which Thread/SENTRY was installed in previous steps. This list identifies the one or more DB2 subsystems that will be concurrently monitored.

(2) Specify the same list of DB2 subsystems defined in ***(1)***.

Figure 20.22.3

Prepare the Thread/SENTRY Job

Next, prepare the Thread/SENTRY jobstream in member TTSRMON of the TTSCNTL dataset, as illustrated in Figure 20.22.4.

```
//jobname  JOB ......                                              (1)
// SET TTSHLQ='?ttshlq?'                                           (2)
//*
//PROCLIB JCLLIB ORDER=&TTSHLQ..TTSCNTL
//*
//TTSMON   EXEC TTSPROC,    Invoke the general Thread/SERIES procedure
//         PROG=TTSMON,     Run the Thread/SENTRY monitor program
//         PARMS='MODE(CANCEL)'     Specify optional parameters
//TTSIN    DD   DISP=SHR,DSN=&TTSHLQ..TTSCNTL(TTSIN)
```

where

(1) Define a valid job statement in order to submit Thread/SENTRY as a standard job. Chapter 11 describes how to run Thread/SENTRY as a started task.

(2) Set to the high level qualifier of the Thread/SERIES libraries as described in Section 20.5 of this manual.

Figure 20.22.4 Submitting Thread/SENTRY as a job

Submit the Thread/SENTRY and IVP Jobs

Next, submit the job in member **TTSRMON** to start Thread/SENTRY**.** Then submit the IVP batch job in member **TTSRIVP.** Thread/SENTRY will enforce the policy defined in member TTSIN. Since that policy limits the IVP thread to 10 SQL statements, Thread/SENTRY should soon cancel the threads associated with the TTSIVP application on each DB2 subsystem in which it executes. Each IVP jobstep that executes TSO in batch (program IKJEFTxx) should terminate with a completion code of 12. See the SYSTSPRT dataset allocated to the jobstep to view the IKJ56641I system abend associated with the canceled thread on a particular DB2 subsystem. Thread cancellation will result from an 04E abend. Once you observe the IVP jobstep(s) are canceled, you can stop Thread/ SENTRY by issuing the following MVS STOP command:

```
STOP thread_sentry_jobname
```

NOTE: *Thread/SENTRY Console Commands are described in detail in Chapter 11.*

You can review the audit trail maintained in the DB2 table whose default name is `TTS?ttsvrm?.Thread_Audit` on each DB2 subsystem in which an IVP thread was created and canceled. The Thread_Audit table maintains detailed statistics relating to the canceled thread and the reason it was canceled. (Appendix A describes the columns comprising the Thread_Audit table). In addition, you can review the log of thread violations and actions that Thread/SENTRY maintains for each concurrently monitored DB2 subsystem. Log files are defined in the TTSPROC procedure with DDnames in the form **TTSLxxxx,** where xxxx is the 4 character DB2 subsystem name.

NOTE: *When viewing the contents of the TTSLxxxx files with IBM's SDSF product, buffered messages may not be visible immediately. Thus, Thread/SENTRY messages may appear to be subject to a time delay when they are, in fact, present in a JES SYSOUT buffer.*

Thread/SENTRY can be considered operational once this Installation Verification Procedure is complete. Problems should be reported immediately to RAI technical support. Once installed and verified, you have the option to run Thread/SENTRY in WARN mode until you are more familiar (and comfortable) with its operation. Thread/SENTRY's warn and cancel modes of operation are described in Section 10.3.1.

20.17.3 Verify Thread/STOPPER Installation

Chapter 2 describes the Thread/STOPPER Dialog in detail. To verify its installation, run the ISPF dialog as described in Chapter 2. Once again, the threads associated with the TTSRIVP application (run in either batch or foreground modes) represent a convenient set of cancellation targets.

20.17.4 Verify the Thread/SERIES Audit View Facility (Optional)

Chapter 7 describes the Thread/SERIES Audit View Facility in detail. To verify its installation, run the ISPF dialog as described in Chapter 7. Once again, the threads associated with the TTSRIVP application (run in either batch or foreground modes) represent a convenient set of threads whose audit trail rows can be viewed.

20.18 Post Installation / Deployment Procedures

Figures 20.23 and 20.24 below illustrate how to set up the Thread/STOPPER Dialog and Thread Audit View Facility as options 'T' and 'V' respectively on the DB2I Primary Option Menu.

```
                          DB2I PRIMARY OPTION MENU
COMMAND ===>

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                 (Process SQL statements)
 2  DCLGEN                (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION   (Prepare a DB2 application program to run)
 4  PRECOMPILE            (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE      (BIND, REBIND, or FREE plans or packages)
 6  RUN                   (RUN an SQL program)
 7  DB2 COMMANDS          (Issue DB2 commands)
 8  UTILITIES             (Invoke DB2 utilities)
 D  DB2I DEFAULTS         (Set global parameters)
 T  Thread/STOPPER        (Monitor and control DB2 threads)
 V  TTSVIEW               (Thread Audit View Facility)
 X  EXIT                  (Leave DB2I)

 PRESS:                    END to exit      HELP for more information
```

Figure 20.23 Adding Thread/SERIES dialogs to the DB2I menu

```
&SEL = TRANS(TRUNC(&OPT,'.')
           0,'CMD(%DSNEAC01)'             /* Service aids (debugging)*/
           1,'CMD(%DSNESC01 FUNC(SPUFI))' /* SPUFI                   */
           2,'CMD(%DSNEDC01)'             /* DCLGEN                  */
           3,'CMD(%DSNEPC01)'             /* Program preparation     */
           4,'CMD(%DSNETC01)'             /* Precompile              */
           5,'PANEL(DSNEBP01)'            /* BIND primary option     */
           6,'CMD(%DSNERC01)'             /* RUN                     */
           7,'CMD(%DSNEKC01)'             /* DB2 commands            */
           8,'CMD(%DSNEUC01)'             /* Utilities               */
           D,'CMD(%DSNEOC01)'             /* Defaults                */
          T,'CMD(%TTSRUN)'              /* Thread/STOPPER Dialog   */
          V,'CMD(%TTSRUN) PARM(VIEW,&SSID)' /* Audit View Facility*/
           X,'EXIT'                       /* Leave DB2I              */
           *,'?')
 &ZTRAIL = .TRAIL                         /* Allow selection chaining*/
)END
```

Figure 20.24 Adding Thread/SERIES dialogs to the DB2I menu

Appendix A

The Thread_Audit Table

A.1 Audit Trail for Actions against Threads

Thread/SERIES maintains an audit trail which describes DB2 threads and Thread/SERIES actions. The audit trail is maintained in a table named THREAD_AUDIT that resides within the same DB2 subsystem in which the audited thread was executing. Section A.2 describes the columns which comprise the Thread_Audit table. Some columns provide statistics and details about the thread while other columns document who took action against a thread, when and for what reason.

Thread/SENTRY (the automated facility that enforces your site's predefined policies) inserts rows into the Thread_Audit table whenever AUDIT(YES) is specified (or defaulted) for a policy which was violated. In contrast, the various Thread/STOPPER Facilities *unconditionally* insert rows into the Thread_Audit table.

The Thread_Audit table can be shared between Thread/SENTRY and the various Thread/STOPPER components. Alternatively, Thread/SENTRY and Thread/STOPPER can access **discrete** THREAD_AUDIT tables as described in Section 20.9.2. The Thread_Audit table referenced by Thread/STOPPER and Thread/SENTRY are governed by the value of the QUALIFIER operand specified when their respective plans are bound. Ask your Thread/SERIES product administrator for the authorization ID of the owner of the THREAD_AUDIT table on a particular DB2 subsystem. Auditors needing access to one or more of these THREAD_AUDIT table(s) through the Thread Audit View Facility, QMF, SPUFI or another SQL application will require the SELECT privilege in order to view them.

A.2 Structure of the Thread_Audit Table

Figure A.1 illustrates the SQL DDL with which the Thread_Audit table is created and shows the columns which comprise the Thread_Audit table. The numbers in parentheses to the right of each Thread_Audit column correspond to the numbered, annotating paragraphs which follow the figure.

```
CREATE TABLE ttsqual.THREAD_AUDIT                          (0)

    The following columns identify the thread against which some action was
    taken

(DB2_SUBSYSTEM              CHAR(4),                       (1)
 UNIQUENESS_VALUE           CHAR(12),                      (2)
 THREAD_TOKEN               INTEGER,                       (3)
 ACE_ADDRESS                CHAR(8),                       (4)

    The following columns describe what action was taken, for what reason,
    and the action's outcome

 ACTION_TAKEN               CHAR(12),                      (5)
 ACTION_STATUS              CHAR(12),                      (6)
 POLICY_ID                  CHAR(8),                       (7)
 POLICY_REASON              CHAR(24),                      (8)
 ACTION_METHOD              CHAR(40),                      (9)

    The following columns describe when, where and by whom a Thread related
    action was taken

 ACTION_DATE                DATE,                          (10)
 ACTION_TIMESTAMP           TIMESTAMP,                     (11)
 ACTION_ACEE_ID             CHAR(8),                       (12)
 ACTION_DB2AUTH_ID          CHAR(8),                       (13)
 ACTION_GROUP_NAME          CHAR(8),                       (14)
 ACTION_TERM_ID             CHAR(8),                       (15)
 ACTION_APPL_ID             CHAR(8),                       (16)
 ACTION_SURR_ID             CHAR(8),                       (17)
 ACTION_CPU_ID              CHAR(12),                      (18)

    The following columns of the Thread_Audit table provide statistics and other
    details about the thread against which Thread/SERIES took action

 CREATION_DATE              CHAR(10),                      (19)
 CREATION_TIME              CHAR(8),                       (20)
 CREATION_MICROSECS         CHAR(7),                       (21)
```

```
CORRELATION_NAME            CHAR(12),          (22)
PLAN_NAME                   CHAR(8),           (23)
PROGRAM_NAME                CHAR(8),           (24)
AUTHORIZATION_ID            CHAR(8),           (25)
CONNECTION_NAME             CHAR(8),           (26)
ORIGINAL_OPERATOR           CHAR(8),           (27)
MVS_SYSTEM                  CHAR(8),           (28)
ADDRESS_SPACE_ID            CHAR(4),           (29)
THREAD_JOBNAME              CHAR(8),           (30)
WHERE_EXECUTING             CHAR(4),           (31)
ACCOUNTING_TOKEN            CHAR(22),          (32)
DISPLAY_SQL_COUNT           SMALLINT,          (33)
STATUS_CODE                 CHAR(2),           (34)
NETWORK_ID                  CHAR(8),           (35)
LU_NAME                     CHAR(8),           (36)
COMMIT_COUNT                INTEGER,           (37)
SQL_DML_COUNT               INTEGER,           (38)
GETPAGES_ISSUED             INTEGER,           (39)
STATUS_LITERAL              CHAR(40),          (40)
CONNECTION_CODE             INTEGER,           (41)
CONNECTING_SYSTEM           CHAR(12),          (42)
TCB_ADDRESS                 CHAR(8),           (43)
TOTAL_ELAPSED_TIME          CHAR(15),          (44)
DB2_ELAPSED_TIME            CHAR(15),          (45)
CLASS1_TCB_TIME             CHAR(15),          (46)
CLASS2_TCB_TIME             CHAR(15),          (47)
HOME_SRB_TIME               CHAR(15),          (48)
IO_WAIT_TIME                CHAR(15),          (49)
LOCK_WAIT_TIME              CHAR(15),          (50)
DIST_LOCATION               CHAR(16),          (51)
DIST_APPC_ID                CHAR(8),           (52)
DIST_SESSION_ID             CHAR(16),          (53)
```

The following columns provide serviceability information about Thread/SERIES internal processing.
These fields are <u>not</u> further described in this appendix.

```
THREAD_FLAG1                CHAR(1),
THREAD_FLAG2                CHAR(1),
THREAD_FLAG3                CHAR(1),
THREAD_FLAG4                CHAR(1),
THREAD_FLAG5                CHAR(1),
THREAD_FLAG6                CHAR(1),
THREAD_FLAG7                CHAR(1),
THREAD_FLAG8                CHAR(1),
THREAD_FLAG9                CHAR(1),
THREAD_FLAGA                CHAR(1),
THREAD_FLAGB                CHAR(1)
```

The following columns of the Thread_Audit table contain MVS and DDF accounting data whose values further describe and identify the thread against which Thread/SERIES took action

```
    QMDA_PRODUCT                CHAR(3),            (54)
    QMDA_VERSION                CHAR(2),            (55)
    QMDA_RELEASE                CHAR(2),            (56)
    QMDA_MOD                    CHAR(1),            (57)
    QMDA_LOCATION               CHAR(16),           (58)
    QMDA_NETID                  CHAR(8),            (59)
    QMDA_LUNAME                 CHAR(8),            (60)
    QMDA_CONNECT_NAME           CHAR(8),            (61)
    QMDA_CONNECT_TYPE           CHAR(8),            (62)
    QMDA_CORRELATION            CHAR(12),           (63)
    QMDA_AUTHID                 CHAR(8),            (64)
    QMDA_PLAN                   CHAR(8),            (65)
    CLIENT_PLATFORM             CHAR(18),           (66)
    CLIENT_APPLICATION          CHAR(20),           (67)
    CLIENT_AUTHID               CHAR(8),            (68)
    ACCOUNT_STRING#             SMALLINT,           (69)
    ACCOUNT_STRING              VARCHAR(200)        (70)
)
```

Figure A.1 Column structure of the Thread_Audit table

A.2.1 Thread Identification Columns

(0) The symbolic variable ‘owner’ denotes the authorization ID of the owner of the THREAD_AUDIT table

(1) The **DB2 subsystem** column identifies the DB2 subsystem name in which the thread was executing.

(2) The column named ***UNIQUENESS_VALUE*** specifies a 12 digit hexadecimal value that uniquely identifies a thread within a particular DB2 subsystem. This uniqueness value or instance number is the third component of a four part DB2 logical-unit-of-work ID which includes:

- the ***NETWORK_ID*** as the first component, described in ***(35)***
- the ***LU_NAME*** qualifier as the second component, described in ***(36)***
- the ***Uniqueness_Value*** (described above) as the third component
- the ***Commit count*** as the fourth and last component described in ***(37)***

(3) The ***THREAD_TOKEN*** column specifies the shorthand number DB2 assigns to each thread. The token is one to five decimal digits. Threads connected to DB2 subsystems at releases ***prior*** to Version 4.1 have no thread token assigned. Their thread token value appears as 0.

(4) The column named ***ACE_ADDRESS*** identifies the Agent Control Element associated with a DB2 thread. The ACE value is an address comprised of 8 hexadecimal digits. Together with the DB2 subsystem name, the ACE uniquely identifies a DB2 thread while it exists. Once the thread is terminated, the Agent Control Element can become associated with a subsequent thread.

A.2.2 Thread/SERIES Actions Columns

(5) The ***ACTION_TAKEN*** column identifies the action (such as cancel or warning) taken against a thread. Action_Taken identifies either the user requested action issued through one of the Thread/STOPPER facilities, or the policy defined action triggered automatically by Thread/SENTRY.

(6) The column named ***ACTION_STATUS*** identifies the current status of processing the action described by ***ACTION_TAKEN***.

(7) ***POLICY_ID*** identifies the site defined policy whose violation led Thread/SENTRY to take action automatically. Alternatively, Policy_ID contains the value 'Manual' to denote an action that was manually initiated by a user.

(8) The column named ***POLICY_REASON*** contains a character string which briefly describes what threshold was violated to trigger a policy defined action. Policy reason strings are comprised of a category prefix followed by the specific threshold that was violated. The currently defined set of policy categories and their corresponding prefixes include:

Life-of-thread thresholds	LOT
Unit-of-work thresholds	UOW
Interval based minimum thresholds	IMIN
Interval based maximum thresholds	IMAX
Inactive thread thresholds	IDLE

(9) The ***ACTION_METHOD*** column describes the means by which a Thread/SERIES component took action. Thread/SERIES cancellation mechanisms are described in detail in Appendix B.

A.2.3 When, where and by whom was an action taken

(10) The ***ACTION_DATE*** column indicates the date when this audited action took place.

(11) The column named ***ACTION_TIMESTAMP*** reflects the current timestamp when this row was inserted into the Thread_Audit table.

(12) The ***ACTION_ACEE_ID*** column identifies what RACF ID took action against the thread.

(13) The column named ***ACTION_DB2AUTH_ID*** identifies the primary authorization ID of the process which issued this action.

(14) The ***ACTION_GROUP_NAME*** column represents the name of the RACF Valid Connect Group.

(15) The column named ***ACTION_TERM_ID*** identifies the ID of the terminal from which the audited action was issued. This field may be blank when for example the Port of Entry (POE) is not specified or the POE class is not 'terminal'.

(16) The ***ACTION_APPL_ID*** column identifies the name of the VTAM application to which the Action requester was connected. ACTION_Appl_ID appears as blanks if no application is specified.

(17) The column named ***ACTION_SURR_ID*** identifies the RACF surrogate userid associated with the audited action.

(18) The ***ACTION_CPU_ID*** identifies the serial number of the CPU from which the audited action was issued.

A.2.4 Thread Statistics and Details

(19) The column named ***Creation_Date*** represents the date on which the thread was created. The Creation_Date value appears in MM/DD/YYYYY format.

(20) The ***Creation_time*** column indicates when the thread was created. The Creation_Time column is recorded in hh.mm.ss format.

(21) The column named ***Creation_microseconds*** contains 6 decimal digits which further qualify (with microsecond precision) when the thread was created.

(22) The ***Correlation_Name*** denotes a 1 to 12 character recovery 'correlation-id' associated with the thread.

(23) The ***Plan_Name*** denotes the 1 to 8 character name of the DB2 application plan associated with the thread.

(24) ***Program_name*** identifies the name of the DB2 package or DBRM that is currently executing within the plan. The value of program_name appears as "N/P" if the current package or DBRM is either not present or cannot be determined.

(25) The column named ***Authorization_ID*** identifies the 8 character DB2 primary authorization associated with the thread.

(26) ***Connection name*** identifies the 8 character DB2 connection type. Possible values include TSO, BATCH, DB2CALL and UTILITY. Threads originating in CICS and IMS address spaces may have additional connection names.

(27) The column named ***Original_Operator ID*** denotes the original value of the primary authorization ID associated with the thread before it could be changed by any authorization exit.

(28) ***The MVS_System column*** denotes the 4 character MVS system identifier.

(29) The column named ***Address_Space_ID*** specifies the four hexadecimal digits which represent the ASID of the thread's allied address space.

(30) ***Thread_Jobname*** denotes the name of the job associated with the allied address space from which the thread originated.

(31) ***Where_Executing*** denotes where the thread was executing when this row was inserted. PGM indicates the thread is currently executing within application program code, while the value 'DB2' denotes a thread that is currently active within DB2. The value 'VTAM' indicates the thread is currently executing or suspended within VTAM. Lastly, 'IDLE' denotes an inactive database access thread.

(32) The column named ***Accounting_Token*** represents an Accounting correlation token. This value can be used when present to correlate DB2 IFC records to CICS records for a CICS transaction.

(33) ***Display_SQL_Count*** denotes the application request count as reported by the DB2 -DISPLAY THREAD command. This is a small integer value that resets to 0 once it reaches its maximum value of 32,767. In contrast, the column named ***SQL_DML_COUNT (38)*** is an integer value accurate beyond 2 billion.

(34) The column named ***Status_code*** contains a 1 or 2 character code which describes the status of the connection. These values correspond to the status codes reported by the -DISPLAY THREAD command and convey the same meanings. The particular status codes (in alphabetic order) are as follows:

D the thread is in the process of termination

DA The database access thread slot is currently not associated with a remote connection and is available to be assigned to a type 2 inactive thread.

DI The thread is disconnected from an execution unit. There is no TCB associated with the DB2 thread. This state is only valid when 'connection name'=RRSAF.

N	the thread is in either IDENTIFY or SIGNON status. No plan is allocated.
ND	The thread is in either IDENTIFY or SIGNON status, and the thread is currently not associated with any TCB.
PT	A parallel task thread was established (plan allocated).
QD	the thread is queued for termination
QT	the CREATE THREAD request is queued
RA	denotes remote access for a distributed thread
R2	A distributed thread is performing a remote access on behalf of a request from another location. The thread is currently an inactive connection (type 2 inactive thread) and is waiting for an agent to become available to process.
RK	A distributed thread is performing remote access on behalf of a request from another location. The thread is performing an operation that invoked Kerberos services. This status is displayed until Kerberos services returns control to DB2.
RN	denotes a remote distributed thread that is suspended
RQ	denotes a distributed thread which is remotely queued
RX	The distributed thread is executing an XA transaction on behalf of a request from another location.
SA	denotes an active stored procedure
SP	A thread is executing within a stored procedure. This status is displayed until the stored procedure terminates and returns control to DB2.
SW	denotes a stored procedure waiting to be scheduled
T	denotes an allied, non-distributed thread for which a DB2 application plan is allocated
TD	An allied thread was established (plan allocated), and the thread is currently not associated with any TCB.
TN	denotes an allied, distributed thread which is suspended
TR	indicates an allied thread was distributed to access data at another location

A.2.5 Columns which identify the LU 6.2 Logical Unit of Work ID

The DB2 logical-unit-of-work ID uniquely identifies a logical unit of work associated with a thread within the network. It consists of four concatenated components:

- the ***NETWORK_ID*** is the first component and is described in ***(35)***
- the ***LU_NAME*** is the second component and is described in ***(36)***
- the ***Uniqueness_Value*** is the third component and is described in ***(2)***
- the ***Commit_Count*** is the fourth and last component and is described in ***(37)***

(35) The ***NETWORK_ID*** qualifier is the first component of the logical-unit-of-work ID. It specifies the SNA Network ID associated with a thread. Network_ID can be between 1 and 8 characters long.

(36) The ***LU_NAME*** qualifier represents the second component of the logical-unit-of-work ID. It specifies an SNA Logical Unit Name between 1 and 8 characters long.

(37) The column named ***Commit_Count*** is the fourth component of the logical-unit-of-work ID.

A.2.6 Other statistics and details about the Thread

(38) The column named ***SQL_DML_COUNT*** represents an 'accurate' count of the number of SQL data manipulation language (DML) statements issued by the thread. This value is in contrast to the 'wrap around' half word counter appearing in the output of the -DISPLAY THREAD command (whose maximum value is 32767). Anytime the -DISPLAY THREAD counter exceeds this value, it 'wraps around' or resets to 0. In contrast, the SQL_DML_COUNT is accurate beyond 2 billion.

(39) ***Getpages_issued*** denotes the number of GETPAGE requests issued on behalf of the thread. This includes conditional and unconditional requests as well as successful and unsuccessful requests. This value is an excellent measure of thread activity in that the getpage count is reliably updated by DB2 during thread execution.

(40) The column named ***Status_Literal*** provides a more meaningful description of the 1 or 2 character Status_Code described in ***(34)***.

(41) The ***Connection_Code*** column contains a hexadecimal code denoting the type of system connecting to DB2.

Hex Code	Connection Type
1	TSO foreground and background
2	DB2 call attach
3	DL/I batch
4	CICS attach
5	IMS attach BMP
6	IMS attach MPP
7	System-directed access
8	Application-directed access
9	IMS control region
A	IMS transaction bmp

The ***Connection_Code*** column can also have a null value. For example, utilities do not have a connecting system type.

(42) The column named ***Connecting_System*** contains a literal value which corresponds to the Connection_Code in (41). The hexadecimal Connection_Codes and their corresponding Connecting_System literal strings are as follows:

Hex Code	Connecting_System Literal
1	TSO
2	CALL ATTACH
3	DL/1 BATCH
4	CICS ATTACH
5	IMS BMP
6	IMS MPP
7	DISTRIBUTED UOW
8	REMOTE UOW
9	IMS CONTROL RGN
A	IMS TRX BMP

(43) ***TCB_Address*** represents a task control block address when a thread is canceled by abnormally terminating its associated task or possibly its parent task. The ***TCB_Address*** column contains zeros when mechanisms other than abending the task are used to cancel the thread.

(44) The column named ***Total_Elapsed_time*** represents the elapsed 'wall clock' time since the thread was created. This value is reported in hours, minutes and seconds.

(45) ***DB2_Elapsed_Time*** represents the total elapsed time spent by the thread within DB2. This value is reported in hours, minutes and seconds.

(46) The column named ***Class1_TCB_Time*** represents the amount of TCB CPU time expended so far by the thread, both within the application and within DB2. For database access agents this value represents TCB time for the agent. A value of 'N/P' means no timing is available.

(47) ***Class2_TCB_Time*** represents accumulated home CPU time expended by the thread while processing within DB2. A value of 'N/P' means no timing is available.

(48) The column named ***Home_SRB_Time*** represents accumulated SRB time charged to the allied address space associated with a thread. Note that SRB times reflect the SRB time for the *entire* address space. For example, a single CICS address space can support multiple threads. As such, the SRB times for CICS threads reflect the SRB time of the entire CICS address space rather than just a specific thread.

(49) ***IO_wait_time*** represents the amount of time the thread spent waiting to perform I/O.

(50) The column named ***Lock_wait_time*** represents the amount of time the thread spent waiting to acquire locks.

(51) The ***Dist_Location*** column identifies the name of the location from which a distributed thread originates. Location name may be up to 16 characters long.

(52) The column named ***Dist_APPC_ID*** represents the VTAM APPC conversation ID. This 32-bit APPC conversation ID is used to identify a particular APPC conversation.

(53) ***Dist_Session_ID*** identifies the VTAM session which underlies the APPC conversation associated with the distributed DB2 thread.

A.2.7 MVS and DDF Accounting Data Associated with the Thread

(54) ***QMDA_Product*** identifies the product that generated the accounting string. The product identifier may assume one the following values:

DSN	denotes DB2 for OS/390 or DB2 for MVS/ESA
ARI	denotes SQL/DS or DB2 for VM
SQL	denotes DB2 client/server
QSQ	denotes DB2/400

(55) ***QMDA_Version*** identifies the version of the product that generated the accounting string.

(56) ***QMDA_Release*** identifies the release level of the product that generated the accounting string.

(57) ***QMDA_Mod*** identifies the modification level of the product that generated the accounting string.

The following columns of the Thread_Audit table apply only to threads whose accounting strings were generated by either DB2 for OS/390 or DB2 for MVS/ESA, i.e. threads whose QMDA_Product value is 'DSN'.

(58) ***QMDA_Location*** identifies the DB2 location name of the DB2 system that created the accounting string.

(59) ***QMDA_NetID*** identifies the SNA NETID of the DB2 system that created the accounting string.

(60) ***QMDA_LUName*** identifies the SNA LU name of the DB2 system that created the accounting string.

(61) ***QMDA_Connect_Name*** identifies the DB2 Connection Name at the DB2 system where the SQL application is running.

(62) ***QMDA_Connect_Type*** identifies the DB2 Connection Type at the DB2 system where the SQL application is running.

(63) ***QMDA_Correlation*** identifies the DB2 Correlation ID at the DB2 system where the SQL application is running.

(64) ***QMDA_AuthID*** identifies the DB2 authorization ID that the SQL application used, prior to name translation and prior to driving the connection exit at the DB2 site where the SQL application is running.

(65) ***QMDA_Plan*** identifies the DB2 PLAN that the SQL application used at the DB2 site running the SQL application.

The following columns of the Thread_Audit table apply only to threads whose accounting strings were created by DB2 client server products such as DB2 for Windows NT, DB2 for OS/2 and DB2 for various Unix implementations

(66) ***Client_Platform*** identifies the Client Platform where the SQL application is running. This is a blank padded value.

(67) ***Client_Application*** identifies the name of the Client Application. This is a blank padded value.

(68) ***Client_AuthID*** identifies the authorization ID of the client application process.

The following columns of the Thread_Audit table apply to all threads for which accounting strings are available

(69) ***Account_String#*** specifies the length (i.e. number of characters) in the accounting string.

(70) ***Account_String*** contains the significant data present in the accounting string. The accounting string value may be up to 200 bytes in length.

A.3 The Values of Columns in Thread_Audit table Rows

Thread/SERIES components perform two types of inserts to the Thread_Audit table. A *full insert of all columns* takes place when most (or all) details about the thread are available. In contrast, a *partial row insert* provides values for just a few columns. The remaining, omitted columns contain nulls (typically denoted by the hyphen '-' character.)

For example, a full insert takes place when a Thread/SERIES component initially takes action (and marks the Action_Status as pending.) Later, when the thread no longer exists, Thread/SERIES inserts just a subset of columns in the row to indicate the action is complete. The omitted columns assume null values.

Sometimes column values are simply not available to be recorded in the Thread_Audit table. In these cases, Thread/SERIES makes use of the following codes to denote missing values:

Code	Meaning
N/P	not present
N/C	cannot be calculated
N/A	not applicable

Appendix B

Thread Eligibility
Cancel Mechanisms

Descriptions of Cancel Responses
Notifications

This Appendix documents several topics that are applicable to all Thread/SERIES components. Exceptions and information unique to a specific Thread/SERIES component are noted as appropriate.

B.1 Cancellation Eligibility Rules

When any Thread/SERIES component receives a CANCEL command, it performs a series of checks to determine whether a particular thread is eligible to be canceled, and if eligible, the best means to cancel it. The thread cancellation mechanisms employed by Thread/SERIES are described in Section B.2. The rules that govern a thread's eligibility for CANCEL and FORCE processing are as follows:

1. A thread that is already being deleted (denoted by a status code of D) or a thread that is queued for deletion by DB2 (denoted by a status code of QD) are not eligible for conventional cancellation. We recommend you be patient with such threads since they are in the process of being gracefully terminated by DB2. If absolutely essential, you can issue the FORCE command against such threads.

2. Threads in IDENTIFY or SIGNON status (i.e. threads without a plan name) may not be canceled. Although not recommended, you may issue the FORCE command against such threads.

3. If the ASID associated with a thread at the time a cancel request is issued is that of the DB2 System Services address space, then Thread/SERIES rejects all CANCEL and FORCE requests. To do otherwise might jeopardize the availability of DB2 itself. In such cases, requesters should simply wait until the ASID associated with a thread again references the client address space. At that time, the thread may be canceled through one of the standard mechanisms supported by Thread/SERIES.

4. The threads associated with the various Thread/SERIES DB2 applications cannot be canceled. Instead, use the documented means to stop the Thread/SERIES component as described in this manual. If essential, you can issue the FORCE command against the DB2 thread(s) associated with a Thread/SERIES component.

As a precaution, Thread/SERIES rejects any CANCEL command which would cancel more than a single thread. The only exception is when the Thread/STOPPER Batch Facility is invoked with CANCEL_MULTIPLE as an execution parameter (as described in Chapter 4). Be careful to use the CANCEL_MULTIPLE option judiciously since you may inadvertently cancel many (even all) active DB2 threads.

B.2 Thread Cancellation Mechanisms

Once Thread/SERIES determines that a particular thread is eligible to be canceled, it chooses the best means to cancel it. This section describes the various mechanisms used by Thread/SERIES to cancel DB2 threads,

Thread/SERIES employs a principle of minimum force to cancel DB2 threads in the least disruptive way possible. Thread/SERIES uses native DB2 thread cancellation commands whenever circumstances permit. Thread/SERIES utilizes other thread cancellation mechanisms only when DB2 facilities will not work. These include situations such as the following:

- a thread is looping within application code
- a distributed thread is currently executing or suspended in the communications network (TCP/IP and/or SNA/VTAM)
- a thread is idle (an inactive Database Access Thread)

B.2.1 The DB2 CANCEL (DDF) THREAD command

The -CANCEL THREAD command was introduced with DB2 V4.1 to cancel both local and distributed threads. The -CANCEL DDF THREAD command has been available since Version 2.3 of DB2 to cancel distributed threads. These native DB2 facilities provide the most graceful, well behaved means to cancel a thread provided the thread passes control to DB2 at some point.

B.2.2 Abending a Thread Task

Thread/SERIES provides a means to abend the MVS task associated with a DB2 thread. Thread/SERIES causes the thread task within the allied address space (such as a batch jobstep, QMF user or CICS thread subtask) to be abnormally terminated. DB2 'listens' for such abnormal end of task events and proceeds to delete the thread and backout any uncommitted changes to DB2 resources. This Thread/SERIES cancel mechanism also lets you obtain a diagnostic dump if an appropriate SYSUDUMP dataset is allocated to the jobstep of the abended task.

B.2.3 The FORCE command

The FORCE command is designed for persistent threads that are particularly difficult to remove. The FORCE command can also be used to terminate threads that are not eligible for conventional cancellation. These include database access threads within a DDF address space and threads which are being deleted by DB2. Basically, FORCE bypasses a number of checks the CANCEL command normally makes to determine whether a thread is eligible to be canceled.

The FORCE command should be used judiciously since its effects may be broader than just abending the thread you wish to cancel. For example, the CICS/DB2 Attachment may need to be recycled through the DSNC STRT command or the DDF address space may need to be restarted.

However, the Thread/SERIES FORCE command ***should not be confused*** with the MVS FORCE command. While the Thread/SERIES FORCE command *unconditionally* abends the MVS task associated with a DB2 thread, it remains far less drastic than the MVS FORCE command with removes an entire address space and can make it necessary to reIPL MVS.

B.2.4 Canceling an ISPF Logical Screen

Thread/SERIES can cause the ISPF logical screen associated with a DB2 thread to abend. This allows threads associated with DB2 editors such as File-AID for DB2™ to be canceled gracefully -- without placing the user's TSO session in a wait state. The TSO/ISPF user can continue with other work once Thread/SERIES cancels their DB2 editor session.

If the target thread is not executing within a TSO/ISPF environment, then a request to cancel an ISPF logical screen is converted internally to a more appropriate Cancel command.

B.2.5 Communications Network Cancellation

When a distributed thread is currently active or suspended within the communications network (TCP/IP or SNA/VTAM), Thread/SERIES can issue TCP/IP or SNA/VTAM commands to terminate the communication connections that underlie the distributed thread. These communication canceling mechanisms do not apply to local or utility threads.

B.2.6 Canceling CICS Threads

Thread/SERIES issues the CICS command **`SET TASK(taskid) PURGE`** when the CICS/DB2 thread cannot be canceled via the DB2 -CANCEL command. The **`SET TASK(taskid) PURGE`** command gracefully cancels the CICS transaction -- without effecting the CICS/DB2 Attachment. Should it be necessary to abend the CICS/DB2 thread subtask, the CICS/DB2 Attachment may terminate. In this case you will have to restart the CICS/DB2 Attachment through the DSNC STRT command.

B.2.7 Canceling DB2 Utility Threads

Thread/SERIES issues the DB2 -TERM UTIL command to cancel DB2 utility threads. Thread/SERIES can also abend the DB2 utility jobstep but in this case the DB2 utility cannot be restarted.

B.2.8 Canceling Threads via Exit Routines

Thread/SERIES can invoke vendor supplied or site written exit routines that get control when installation specified criteria are met. These exit routines receive all available information about the selected thread and can perform non standard and/or site specific processing as appropriate (For example, an exit routine might be designed to issue commands to a DB2 gateway product.) In addition, site written exit routines can request standard cancellation, auditing and notification services from Thread/SERIES.

The RAI supplied module named TTS3CLS is an example of an exit routine which cancels the ISPF logical screen associated with a DB2 thread. Appendix E discusses how to develop exit routines to perform non standard processing as well as how to specify the criteria that govern when these routines are invoked.

B.3 Description of Cancel Responses

B.3.1 Canceling an ISPF Logical Screen

User's whose ISPF logical screens are canceled can observe two different behaviors, depending on the way in which ISPF is invoked.

A normally invoked ISPF session exhibits the most graceful behavior when a Thread/SERIES component cancels an ISPF logical screen. The user sees an ISPF abend panel which displays register contents and ISPF Version / Release information. After pressing ENTER, the ISPF Primary Option Menu is displayed from which the user can continue with other work.

In contrast, when ISPF is invoked with the TEST option, cancellation of an ISPF logical screen proceeds less gracefully. Instead of displaying an abend panel, ISPF issues line mode messages that indicate abends within the ISPF subtask as well as the ISPF Main task. The user may have to press ENTER several times to clear the screen of mini dump information. Eventually, the TSO READY prompt should appear at which point the user can reenter ISPF.

B.3.2 Canceling Distributed Threads

Cancellation of distributed threads proceeds more gracefully, safely and surely if the allied distributed thread (the client requester) is canceled rather than the Database Access Thread active at the server. Moreover, the Database Access Thread at the server will terminate automatically when the allied requester thread is canceled.

Should it be necessary to cancel a database access thread directly, the conversation between the local Database Access Thread and the remote requester will be canceled. However, both threads may continue to exist on their respective subsystems until the allied distributed thread issues another SQL request. Although not recommended, you can issue a FORCE command to terminate the Database Access Thread immediately.

B.4 Thread/SERIES Notification Messages

Whenever possible, the various Thread/SERIES components send notification messages to users whose threads are effected. In addition, it sends notifications to a list of designated administrators when Thread/SENTRY detects a policy violation.

B.4.1 User Notifications

When a DB2 thread originating in a TSO foreground address space is canceled through one of the Thread/STOPPER Facilities, the following multi-line message is sent to the terminal of the effected TSO user:

```
TTS11Ø - Your DB2 thread (Plan: PPPPPPPP, DB2 auth ID: AAAAAAAA)
is being deliberately terminated.  Your DB2 application will abend
with a system abend code CCC and an abend reason code of RRRRRRRR.
***
```

where

PPPPPPPP denotes the name of the DB2 application plan whose thread was terminated

AAAAAAAA identifies the thread's DB2 authorization ID

CCC denotes the system abend code with which the MVS task (associated with the canceled thread) terminates abnormally. By default, Thread/STOPPER uses system code '222' to abend a task while tasks terminated by DB2 typically abend with an 04E system completion code.

RRRRRRRR identifies the abend reason code with which the thread's MVS task terminates abnormally. By default, Thread/STOPPER uses abend reason code 00DB2000 to denote tasks it cancels directly. In contrast, threads canceled through such DB2 mechanisms as CANCEL THREAD and CANCEL DDF THREAD often terminate with a reason code of 00E50013 to indicate the DB2 execution unit has abended.

When Thread/SENTRY cancels a thread due to a policy violation, it sends a similar message to the effected TSO, CICS or IMS terminal -- as follows:

```
TTS57Ø- SSSS thread (Plan: PPPPPPPP  Auth ID: AAAAAAAA  Jobname: JJJJJJJJ)
violated policy XXXXXXX.  It is being canceled at TTTTTTTT on DDDDDDDD
with system abend code CCC and an abend reason code of RRRRRRRR
```

where

SSSS denotes the name of the DB2 subsystem in which the canceled thread was executing.

PPPPPPPP denotes the name of the DB2 application plan whose thread was terminated

AAAAAAAA identifies the thread's DB2 authorization ID

JJJJJJJJ identifies the name of the job associated with the allied address space from which the thread originated.

XXXXXXXX identifies the name of the site defined policy whose violation led Thread/SENTRY to cancel the thread.

TTTTTTTT indicates the time when the thread was canceled. This value is displayed in hh.mm.ss format.

DDDDDDDD identifies the date on which the thread was canceled. This value appears in MM/DD/YY format.

CCC denotes the system abend code with which the MVS task (associated with the canceled thread) terminates abnormally. By default, Thread/SENTRY uses system code '222' to abend a task while tasks terminated by DB2 typically abend with an 04E system completion code.

RRRRRRRR identifies the abend reason code with which the thread's MVS task terminates abnormally. By default, Thread/SENTRY uses abend reason code 00DB2000 to denote tasks it cancels directly. In contrast, threads canceled through such DB2 mechanisms as CANCEL THREAD and CANCEL DDF THREAD often terminate with a reason code of 00E50013 to indicate the DB2 execution unit has abended.

In contrast, when Thread/SENTRY issues a warning, it sends the following message to the effected TSO, CICS or IMS terminal.

```
TTS570- SSSS thread (Plan: PPPPPPPP  Auth ID: AAAAAAAA  Jobname: JJJJJJJJ)
has violated policy XXXXXXXX and receives this warning.
Your DB2 application will NOT be abended.
```

where

SSSS denotes the name of the DB2 subsystem in which the thread was executing.

PPPPPPPP denotes the name of the DB2 application plan whose thread is receiving this warning

AAAAAAAA identifies the thread's DB2 authorization ID

JJJJJJJJ identifies the name of the job associated with the allied address space from which the thread originated.

XXXXXXXX identifies the name of the site defined policy whose violation led Thread/SENTRY to issue a warning.

B.4.2 Notifications sent to Administrators

Thread/SENTRY can send notifications to a list of designated administrators whenever it detects a policy violation. (Chapter 10 describes the Thread/SENTRY NOTIFY_LIST statement in detail). Administrator's receive a cancel notification like the following:

```
TTS575- SSSS thread (Plan: PPPPPPPP  Auth ID: AAAAAAAA  Jobname: JJJJJJJJ)
violated policy XXXXXXXX and was terminated on DDDDDDDD at TTTTTTTT
with system abend code CCC and an abend reason code of RRRRRRRR
```

where

SSSS denotes the name of the DB2 subsystem in which the canceled thread was executing.

PPPPPPPP denotes the name of the DB2 application plan whose thread was terminated

AAAAAAAA identifies the thread's DB2 authorization ID

JJJJJJJJ identifies the name of the job associated with the allied address space from which the thread originated.

XXXXXXXX identifies the name of the site defined policy whose violation led Thread/SENTRY to cancel the thread.

DDDDDDDD identifies the date on which the thread was canceled. This value appears in MM/DD/YY format.

TTTTTTTT indicates the time when the thread was canceled. This value is displayed in hh.mm.ss format.

CCC denotes the system abend code with which the MVS task (associated with the canceled thread) terminates abnormally. By default, Thread/SENTRY uses system code '222' to abend a task while tasks terminated by DB2 typically abend with an 04E system completion code.

RRRRRRRR identifies the abend reason code with which the thread's MVS task terminates abnormally. By default, Thread/SENTRY uses abend reason code 00DB2000 to denote tasks it cancels directly. In contrast, threads canceled through such DB2 mechanisms as CANCEL THREAD and CANCEL DDF THREAD often terminate with a reason code of 00E50013 to indicate the DB2 execution unit has abended.

The warning notification that Thread/SENTRY sends to administrators appears as follows:

```
TTS575- SSSS thread (Plan: PPPPPPPP  Auth ID: AAAAAAAA  Jobname: JJJJJJJJ)
has violated policy XXXXXXXX and receives this warning.
```

where

SSSS denotes the name of the DB2 subsystem in which the thread was executing.

PPPPPPPP denotes the name of the DB2 application plan whose thread is receiving this warning

AAAAAAAA identifies the thread's DB2 authorization ID

JJJJJJJJ identifies the name of the job associated with the allied address space from which the thread originated.

XXXXXXXX identifies the name of the site defined policy whose violation led Thread/SENTRY to issue a warning.

Appendix C

Composing Site Written Messages and Customizing Default Notification Message Text

This Appendix describes how to create site written messages that will be issued when Thread/SENTRY policies are violated. This Appendix also describes how to customize the text of the *default* notification messages Thread/SERIES components send both to users and administrators.

C.1 Composing Site Written Messages

Member TTS$TSM of the TTSCNTL library provides an example of a site written messages module. Message IDs and message text can be composed subject to the guidelines and restrictions appearing in the #MSG macro within the TTSMACS library. Please read the documentation within the #MSG macro before composing any messages. Your site written messages may contain references to the Z amper variables described in the next Section.

C.1.1 Z Amper Variables

Z amper variables all pertain to the *current thread* for which Thread/SERIES is issuing a message. At execution time, Thread/SERIES performs variable substitution in which each reference to a Z amper variable in the prototype message text is replaced with its current value. The Z amper variables currently supported by Thread/SERIES are as follows:

ZAUTH	Authorization ID
ZCAUSE	The character string which briefly describes what threshold was violated to trigger a policy defined action. Policy reason strings are comprised of a category prefix followed by the specific threshold that was violated. The currently defined set of policy categories and their corresponding prefixes include:

Life-of-thread thresholds	LOT
Unit-of-work threshold	UOW
Interval based minimum thresholds	IMIN
Interval based maximum thresholds	IMAX
Inactive thread thresholds	IDLE

ZCODE	The user or system code with which the target thread abends
ZDATE	Current date
ZDSN	Current DB2 Subsystem ID
ZJOBNAME	Jobname associated with the current thread
ZMSGID	Message ID appearing in the text of the message labeled TTS570
ZPLAN	DB2 application plan name
ZPOLICY	Thread/SENTRY Policy ID
ZREASON	The reason code associated with the user or system abend code identified by ZCODE
ZTIME	Current time

NOTE: *When composing a prototype message,* ***always*** *follow a Z amper variable with a blank. For example, '&&ZAUTH ' is allowed but variable substitution will* ***not*** *occur for &&ZAUTH.'*

C.1.2 Assembly / Link Edit Instructions for Site Written Message Modules

The job in member TTSJTSM of the TTSCNTL library may be used to assemble and link edit your site written messages module. Before using job TTSJTSM, you must first edit the TTSPAL and RAIAL procedures (also in the TTSCNTL library) as described in Section 20.13 of the Thread/SERIES Guide and Reference.

Alternatively, you can use any assembly/link edit jobstream that includes the TTSMACS library in the SYSLIB concatenation for the Assembler.

We recommend your site written message module be link edited with the following attributes:

```
AMODE(31),RMODE(ANY),RENT,REUS,REFR
```

C.2 Customizing the text of the Default Notification Messages

Member TTS$TNM of the TTSCNTL library is an Assembler language source module which contains the text of the default notification messages Thread/SERIES sends to both users and administrators.

Your site can customize the text of these default notification messages -- subject to the following guidelines and restrictions.

- Both the CSECT and the link edited load module must be named TTS$TNM.
- Do not change the message numbers, either in the label field in column 1 or within the message text.
- Do not change the names of the Z amper variables (such as &&ZAUTH or &&ZPLAN) which appear embedded in the message text. The Z amper variables are described in Section C.1.1.

C.2.1 Sets of NOTIFY Messages issued by Thread/SENTRY

The following are the ID's of messages sent to users whose thread is cancelled. For this set of messages, Thread/SENTRY assigns a value 'TTS570' to the ZMSGID variable in the message labeled TTS570.

```
TTS570
TTS571
TTS572
```

The following are the ID's of messages sent to users when a thread warning is issued. For this set of messages, Thread/SENTRY assigns a value 'TTS570' to the ZMSGID variable in the message labeled TTS570.

```
TTS570
TTS577
TTS578
```

The following are the ID's of messages sent to USERs when an action on a thread is 'pending'. For this set of messages, Thread/SENTRY assigns a value 'TTS580' to the ZMSGID variable in the message labeled TTS570.

```
TTS570
TTS581
TTS582
```

The following are the ID's of messages sent to ADMINISTRATORs when a thread is cancelled. For this set of messages, Thread/SENTRY assigns a value 'TTS575' to the ZMSGID variable in the message labeled TTS570.

```
TTS575
TTS570
TTS576
TTS572
```

The following are the ID's of messages sent to ADMINISTRATORs when a warning is issued for a thread. For this set of messages, Thread/SENTRY assigns a value 'TTS575' to the ZMSGID variable in the message labeled TTS570.

```
TTS575
TTS570
TTS579
```

The following are the ID's of messages sent to ADMINISTRATORs when an action on a thread is 'pending'. For this set of messages, Thread/SENTRY assigns a value 'TTS583' to the ZMSGID variable in the message labeled TTS570.

```
TTS583
TTS570
TTS584
TTS585
```

C.2.2 Assembly / Link Edit Instructions for Module TTS$TNM

The job in member TTSJTNM of the TTSCNTL library may be used to assemble and link edit your customized version of the Thread/SERIES Notification Messages module (TTS$TNM). Before using job TTSJTNM, you must first edit the TTSPAL and RAIAL procedures (also in the TTSCNTL library) as described in Section 20.13 of the Thread/SERIES Guide and Reference.

Alternatively, you can use any assembly/link edit jobstream, provided you link edit load module TTS$TNM with the following attributes:

```
AMODE(31),RMODE(ANY),RENT,REUS,REFR
```

Appendix E

Thread/SERIES

Exit Routines

E.1 Concepts and Facilities

Thread/SERIES can be customized and extended through vendor supplied and/or site written *exit routines* that get control when installation specified criteria are met. Exit routines receive all available information about the selected thread and can perform non-standard and/or site specific processing as appropriate. For example, one exit routine supplied by Relational Architects cancels the ISPF logical screen associated with a DB2 thread while a site written routine might be designed to issue commands to a DB2 gateway product.

This Appendix describes:

- How to develop Thread/SERIES exit routines to handle non-standard processing requirements. Sections E.2 to E.5 describe how Thread/SERIES exit routines are designed and coded. They also discuss how site written exit routines can request standard cancellation, auditing and notification services from Thread/SERIES.

- How to define exit routines (both site written and vendor supplied) to Thread/SERIES. (Section E.6)

- How to define the criteria and circumstances under which Thread/SERIES will implicitly invoke an exit routine to perform non-standard processing. (Section E.7).

E.2 Exit environment

Thread/SERIES exit routines are invoked via standard CALL statements under the following circumstances:

- An exit routine is called *explicitly* when a Thread/SENTRY policy statement specifies an action of PGM. (See Chapter 10 for details). For example:

```
LIMIT
   ACTION(PGM,exitrtn)
```

- An exit routine is called *implicitly* when a thread selected by any Thread/SERIES component matches the identification pattern specified by an entry in the currently active non-standard processing table. (Section E.7).

Any exit routines you develop must conform to these rules:

- It must be written in assembler.
- It must be written to be reentrant and must restore registers before return.
- It must be link-edited with the REENTRANT parameter.
- It must not issue DB2 commands, SQL statements and/or IFI requests.
- It must reside in an authorized program library.
- The names of site written exit routines should not start with the letters ‘TTS’ so as to not conflict with the names of Thread/SERIES modules.

An exit routine runs as an extension of Thread/SERIES and has all its privileges. It can therefore impact your DB2 subsystems as well as the operating system. Exit routines should not be changed nor modified while Thread/SERIES is running. Although Thread /SERIES has its own recovery facilities, your exit routines can establish supplemental recovery routines of their own.

The execution environment for Thread/SERIES exit routines is as follows:

- problem state
- Authorized
- 24 or 31 bit addressing mode
- Enabled for interrupts
- PSW key 8
- No MVS locks held
- Non Cross-memory mode

E.3 Parameter list on entry

When Thread/SERIES passes control to an exit routine, the registers are set as follows:

Register	Contents
R1	Address of the parameter list passed to the exit routine
R13	Address of the register save area
R14	Return address
R15	Entry point address of the exit routine

Register 1 points to a three word parameter list as follows:

```
Register 1-> +--------------------------------------------------+
             | Address of Instrumentation Facility Record.      |
             | The QWIW structure is mapped by macro DSNDQWIW   |
             |--------------------------------------------------|
             | Address of QWT0, the IFC Self Defining Section.  |
             | The QWT0 structure is mapped by macro DSNDQWT0   |
             |--------------------------------------------------|
             | Address of XPS, the eXit Parameter Structure.    |
             | The XPS structure is mapped by macro TTS@XPS     |
             +--------------------------------------------------+
```

Figure E.1: Parameter List passed to Thread/SERIES Exit Routines

> ***NOTE:*** *The annotated example in Section E.5 illustrates the coding necessary to address the discrete parameters passed in the parameter list.*

Thread/SERIES exit routines receive the three parameters described above for active threads for which IFC 0148 records are available. Exit routines receive the ***same*** parameters for inactive database access threads -- even though no IFC 0148 records are available. Thread/SERIES maps the limited information available for an inactive database access thread into the same IFC 0148 record format applicable to active threads.

E.4 The Thread/SERIES eXit Parameter Structure (XPS)

Thread/SERIES passes the Exit Parameter Structure (XPS) illustrated in Figure E.2 to both site written and vendor supplied exit routines. The XPS is mapped by macro TTS@XPS in the TTSMACS library. The XPS provides the exit routine with input values and a work area for its own use. Additional fields in the XPS allow the exit routine to communicate, in turn, with Thread/SERIES.

Figure E.2 illustrates the field structure of the XPS. Individual fields are described in the subsections which follow:

```
_____________________________________________________________________

Field    Hex offset  Data Type     Description
-------- ----------  ------------  ------------------------------------

e(X)it (P)arameter (S)tructure
*
Values passed from Thread/SERIES to the Exit routine
*
XPSPVS   Ø           Structure     Passed values structure
XPSID    Ø           Character 4   Eye catcher = 'XPS'
XPSTAC   4           Hex       2   Thread/SERIES abend code (12 bits)
XPSTRC   6           Hex       4   Thread/SERIES reason code (32 bits)
XPSWORK@ C           Pointer   2   -> Exit's persistent work area
XPSWORK# 1Ø          Integer   4   Length of exit's work area
XPSTCB@  14          Pointer   4   TCB address of the target thread
*                                  (QW0148MT is the jobstep TCB addr)
*                                  Ø for an inactive DDF thread
XPSJOBNM 18          Character 8   Jobname of the target thread
XPSFLAG1 2Ø          Bit Flag  1   Invocation circumstances
XPS1ACT  EQU         B'10000000'   Bit Ø on:  An active Thread
*                                  Bit Ø off: An inactive DDF thread
XPS1MAN  EQU         B'01000000'   Bit 1 on:  Exception triggered
*                                              manually
*                                  Bit 1 off: Violation detected
*                                              automatically
         21           Character 3  Reserved
*
*
Values returned by the Exit routine
*
XPSRVS   24          Structure     Returned values structure
XPSRC1   24          Integer   4   Exit return code
XPSRC2   28          Integer   4   Exit reason code
         2C          Character 8   reserved
*
*
The exit routine can set (or change) bits in the field named XPSFLAG2 to
indicate what further processing Thread/SERIES should perform -- after the
exit routine returns control.

Thread/SERIES  sets these bits ON before calling the exit.
*
XPSFLAG2 34          Bit Flag  1   Actions after exit completes
XPS2AUD  EQU         B'10000000'   Bit Ø on:   Audit this action
*                                  Bit Ø off:  Do not audit action
```

```
XPS2POP  EQU          B'01000000'  Bit 1 on:   Display POPUP window
*                                 (applies to Thread/STOPPER dialog)
*                                 Bit 1 off:  No POPUP window
XPS2UNOT EQU          B'00100000'  Bit 2 on:   Notify user
*                                 Bit 2 off:  No user notification
XPS2ANOT EQU          B'00010000'  Bit 3 on:   Notify administrators
*                                 Bit 3 off:  No administrator notify
*
*
```

The following values are inserted as columns of a row in the Thread_Audit table when the XPS2AUD bit is set to 1. The line following each field name identifies its corresponding column within the Thread_Audit table. Thread/SERIES initializes these fields with appropriate values before calling the exit.

```
*
XPSACT   35           Char     12  Action taken  (e.g. Cancel)
*                                     Thread_Audit.Action_Taken
XPSSTAT  41           Char     12  Action status (e.g. Pending)
*                                     Thread_Audit.Action_Status
XPSMETH  4D           Char     40  Method used to take action
*                                     Thread_Audit.Cancel_Method
XPSPID   75           Char      8  ID of violated policy
*                                     Thread_Audit.Policy_ID
XPSREAS  7D           Char     24  Reason action was taken
*                                     Thread_Audit.Policy_Reason
```

Figure E.2: Fields comprising the eXit Parameter Structure

E.4.1 Values passed from Thread/SERIES to the Exit routine

Thread/SERIES passes values to your exit routine via the following fields whose names begin with the letters "XPS".

XPSID contains the eyecatcher 'XPS '.

XPSTAC is a binary halfword whose low order 12 bits contains the system abend code defined in the Thread/SERIES Table of System Defaults.

XPSTRC is a binary fullword which contains the 32 bit abend reason code defined in the Thread/SERIES Table of System Defaults..

XPSWORK@ points to a persistent work area provided for the exclusive use of this exit routine. Thread/SERIES initializes this work area to binary zeros before invoking the exit routine for the first time. Thereafter, Thread/SERIES does not alter the contents of this area.

XPSWORK# is a fullword binary integer containing the length of the work area provided to the exit routine.

XPSTCB@ contains the address of the TCB associated with the target thread. This value is set to 0 for an inactive database access thread. Note that field QW0148MT contains the address of the jobstep TCB, not the TCB of the task which actually established the thread.

E.4.2 Values returned by the Exit routine

XPSRC1 is a fullword binary integer which contains the return code set by the exit routine. The conventions for return codes set by Thread/SERIES exit routines are as follows:

- 0 means the exit routine processed the violation itself. Thread/SERIES should perform any actions identified by flag bits set within the XPSFLAG2 field once control returns from the exit routine.
- A non zero return code signals Thread/SERIES to process the violation in standard fashion, as if no exit routine were invoked.

XPSRC2 is a fullword binary integer which contains the reason code returned by the exit routine.

XPSFLAG2 defines a set of flag bits. Your exit routine can set (or change) bits in the XPSFLAG2 field to indicate what further processing Thread/SERIES should perform -- after the exit routine returns control. Thread/SERIES sets these bits ON before calling the EXIT.

When the ***XPS2AUD*** flag bit is on (value 1), Thread/SERIES should insert a row into the Thread_Audit table to audit this action.
If the bit is off (value 0), this action need not be audited.

When the ***XPS2POP*** flag bit is on (value 1), the Thread/STOPPER Dialog should display a POPUP window for a period of time to permit any action taken by the exit routine to complete. If the bit is off (value 0), no POPUP window is displayed. (Note: The POPUP duration is defined in the Table of System Defaults.)

Your exit routine can set the ***XPS2UNOT*** flag bit on (value 1) to direct Thread/SERIES to send a notification to the user of the thread in violation. If the bit is off (value 0), no user notification is sent.

Your exit routine can set the ***XPS2ANOT*** flag bit on (value 1) to direct Thread/SERIES to send notifications to the list of IDs defined as Thread/SERIES administrators. If the bit is off (value 0), no administrative notifications are sent.

E.4.3 Values Inserted into Thread_Audit Columns

When the XPS2AUD bit is set to 1, the following fields supply column values for the row inserted into the Thread_Audit table. Thread/SERIES initializes these fields with appropriate values before calling the exit routine.

XPSACT is a twelve byte character field which identifies the action taken by the exit routine (e.g. Cancel). Field XPSACT corresponds to the column within the Thread_Audit table named Action_Taken.

XPSSTAT is a twelve byte character field which describes the status of the action taken by the exit routine (e.g. Pending). XPSSTAT corresponds to the column within the Thread_Audit table named Action_Status.

XPSMETH is a forty byte character field which describes the method used by the exit routine to take action. (e.g. 'Cancel ISPF Screen'). XPSMETH corresponds to the column within the Thread_Audit table named Cancel_Method.

XPSPID is an eight byte character field which identifies the policy whose violation caused the exit routine to be invoked. XPSPID corresponds to the column within the Thread_Audit table named Policy_ID. XPSPID contains the literal 'Manual' when an action and exit routine are triggered *manually*.

XPSREAS contains a 24 character description of the reason an action was taken against a thread. XPSREAS corresponds to the column within the Thread_Audit table named Policy_Reason.

E.5 An Annotated Thread/SERIES Exit Routine

A sample Thread/SERIES exit routine is provided in source form as member TTS3SXR of the TTSCNTL library. TTS3SXR (shown in Figure E.3) illustrates coding conventions for Thread/SERIES exit routines. TTS3SXR simply locates its parameters and then returns to its Thread/SERIES caller with a return code of 4.

```
        TITLE 'TTS3SXR - THREAD/SERIES SAMPLE EXIT ROUTINE'
*--------------------------------------------------------------------
*
*       Entry Assumptions
*       =================
*
*       R15 ---> TTS3SXR entry point address
*       R14 ---> caller's return address
*       R13 ---> caller's save area
*       R1  ---> 3 word parameter list
*                Word Ø ---> QWIW: a structure mapped by DSNDQWIW
*                Word 1 ---> QWTØ: a structure mapped by DSNDQWTØ
*                Word 2 ---> XPS:  a structure mapped by TTS@XPS
*
*       Exit Conditions
*       ===============
*
*       R15 = Return code
*
*       R15 = Ø means the exit processed the
*             violation.  Thread/SERIES should
*             perform the post exit actions (if
*             any) identified by flag bits set
*             within the XPSFLAG2 field.
*
*       R15 > Ø.  A non zero exit return code
*             signals Thread/SERIES to process
*             the violation in standard fashion,
*             as if no exit routine were invoked.
*
*       Register Usage
*       ==============
*
*       R12 ---> program base
*       R11 ---> XPS
*       R1Ø ---> QWTØ
*       R9  ---> QWIW:IFC record base
*
*       Mapping Macros
*       ==============
*
*       TTS@XPS  TTS exit parameter structure
*       DSNDQWIW IFC record header
*       DSNDQWTØ self defining section
*       DSNDQW02 includes the IFC Ø148 DSECT
*
*--------------------------------------------------------------------
TTS3SXR  CSECT ,
         LM    R9,R11,Ø(R1)        Locate parameters
         USING XPS,R11             map e(X)it (P)arameter (S)tructure
         USING QWTØ,R1Ø            map IFC self defining section
         USING QWIW,R9             map IFC record header
         LA    R15,4               Thread/SERIES should process thread
         BR    R14                 Return to Thread/SERIES caller
*--------------------------------------------------------------------
*
*        Data Declarations
*
*--------------------------------------------------------------------
         #REGEQU                   Set register equates
         TTS@XPS  ,                TTS exit parameter structure
         DSNDQWIW ,                IFC record header
         DSNDQWTØ ,                Self defining section
         DSNDQW02 ,                Includes IFC Ø148 DSECT
         END   TTS3SXR
```

Figure E.3 Thread/SERIES Sample Exit Routine TTS3SXR

E.5.1 Assembly and Link Edit of the Thread/SERIES Exit Routine

Member TTSJSXR of the TTSCNTL library contains the jobstream illustrated in Figure E.4. This job invokes the TTSPAL procedure to assemble and link edit the Thread/ SERIES sample exit routine named TTS3SXR. Be sure to edit JCL member TTSJSXR with CAPS ON since it contains comments in lowercase.

```
//jobname JOB (account)                                          (1)
//* <optional JCLLIB statement>                                   (2)
//TTS3SXR  EXEC TTSPAL,      <- Invoke the procedure TTSPAL
//         MEMBER=TTS3SXR    <- Thread/SERIES Sample Exit routine
```

where

(1) Provide a valid JOB statement.

(2) Add a JCLLIB statement to identify the TTSCNTL dataset if the procedures TTSPAL and RAIAL do not reside within one of your site's catalogued procedure libraries.

Figure E.4 Assemble and Link Edit the Thread/SERIES sample exit routine

E.6 Defining Exit Routines to Thread/SERIES

Both vendor-supplied and site-written exit routines must be defined to Thread/SERIES *before* they can be used. This is true both for routines that are *implicitly* invoked by Thread/SERIES as well as exit routines that are *explicitly* invoked through the ACTION operand of a Thread/SENTRY policy.

Thread/SERIES provides three macros for the purpose of defining your site written exit routines (to add to those provided by Relational Architects). All three macros reside in the macro library whose low level qualifier is TTSMACS. Section E.6.1 provides an annotated example of coding a Thread/SERIES Table of Exit Routines using these macros. Section E.6.2 documents the Thread/SERIES exit definition macros themselves and Section E.6.3 illustrates a job with which to assemble and link edit a Thread/ SERIES Table of Exit Routines.

E.6.1 The Sample Table of Thread/SERIES Exit Routines - TTS$TXR

The source module within member TTS$TXR of the TTSCNTL library illustrates how to define exit routines to Thread/SERIES. The numbers in parentheses to the right of the Assembler language source statements correspond to the numbered, annotating paragraphs which follow.

```
        TITLE 'TTS$TXR - THREAD/SERIES TABLE OF EXIT ROUTINES'
TTS$TXR TTS#TXRI                                    (1)
        TTS#TXR EXIT=TTS3SXR,WORKLEN=1000           (2)
        TTS#TXR EXIT=TTS3CLS,WORKLEN=100            (3)
        TTS#TXRF                                    (4)
        END                                         (5)
```

where

(1) The label field of the TTS#TXRI macro provides the name of the CSECT. The TTS#TXRI macro, the initial macro of the TTS#TXRx macro set, is described in Section E.6.2.1.

(2) The TTS#TXR macro (described in Section E.6.2) defines a single exit routine to Thread/SERIES. This instance of the TTS#TXR macro defines the sample exit routine named TTS3SXR.

(3) This instance of the TTS#TXR macro defines load module TTS3CLS as a Thread/SERIES exit routine. Relational Architects supplies the TTS3CLS exit routine to cancel the ISPF logical screen associated with a DB2 thread

(4) The TTS#TXRF macro (described in Section E.6.2) must be coded last to generate the Thread/SERIES exit routine table.

(5) The END instruction, as the last source statement, terminates the assembly of the program.

Figure E.6 The Sample TTS$TXR Source Module

E.6.2 The TTS#TXRx Macro Set

E.6.2.1 TTS#TXRI

TTS#TXR is the ***initial*** macro of the TTS#TXRx macro set. The TTS$TXRI macro must be specified first, before any other TTS#TXRx macros are coded. TTS#TXRI generates the CSECT header for a Table of Thread/SERIES Exit Routine definitions. The label field of the TTS#TXRI macro provides the name of the CSECT. The default CSECT name (if a label is omitted) is TTS$TXR. This macro has no other operands.

E.6.2.2 TTS#TXR

Each instance of the TTS#TXR macro defines a single exit routine to Thread/SERIES. These exit routines are referenced:

- ***explicitly*** by Thread/SENTRY policies whose ACTION operands name the exit routine (See Chapter 10).

- ***implicitly*** when a thread selected by any Thread/SERIES component matches the identification pattern specified by an entry in the currently active non-standard processing table. (See Section E.7).

TTS#TXR macros must be specified *after* the TTS#TXRI macro. Figure E.7 illustrates the TTS#TXR macro and its operands.

```
      TTS#TXR &EXIT=,            exit name                          +
              &WORKLEN=,         length of persistent work area     +
              &DOC=NO
```

where

`EXIT` specifies the name of the exit routine load module. This name may be up to 8 characters long.

`WORKLEN` specifies the length of a persistent work area acquired by Thread/SERIES for the exclusive use of this exit routine. Thread/SERIES initializes this work area to binary zeros. Thereafter, data saved in this work area is available to the exit routine across multiple invocations.

Figure E.7 The TTS$TXR Macro and its Operands

E.6.2.3 TTS#TXRF

TTS#TXRF is the ***final*** macro of the TTS#TXRx macro set. The TTS#TXRF macro must be coded last to generate the Thread/SERIES exit routine table. The TTS#TXRF macro has no operands.

E.6.3 Building a Thread/SERIES Table of Exit Routines

Member TTSJTXR of the TTSCNTL library contains the jobstream illustrated in Figure E.8. This job invokes the TTSPAL procedure to assemble and link edit the Thread/SERIES Table of Exit routines (Load module TTS$TXR). Be sure to edit JCL member TTSJTXR with CAPS ON since it contains comments in lowercase.

```
//jobname JOB (account)                                    (1)
//* <optional JCLLIB statement>                            (2)
//TTS$TXR  EXEC TTSPAL,      <- Invoke the procedure TTSPAL
//         MEMBER=TTS$TXR    <- Thread/SERIES Table of Exit Routines
```

where

(1) Provide a valid JOB statement.

(2) Add a JCLLIB statement to identify the TTSCNTL dataset if the procedures TTSPAL and RAIAL do not reside within one of your site's catalogued procedure libraries.

Figure E.8 Assemble and Link Edit the Sample Table of Exit Routines

One or more load modules can be created, each of which defines a discrete Thread/SERIES Table of Exit Routines. The default name is TTS$TXR. This default may be overridden by rebuilding the Thread/SERIES Table of System Defaults (load module TTS$TSD) as described in Section 20.14. Alternatively, you can specify use of another Exit Routine Table through the Thread/SENTRY MONITOR statement (described in Chapter 10).

E.7 Defining Criteria for Non-Standard Processing

Thread/SERIES allows you to define criteria under which a Thread/SERIES exit routine should be *implicitly* invoked. An exit routine is called *implicitly* when a thread selected by any Thread/SERIES component matches the identification pattern specified by an entry in the currently active non-standard processing table described in this section.

Thread/SERIES provides three macros for the purpose of defining threads for which non-standard processing is appropriate. All three macros reside in the macro library whose low level qualifier is TTSMACS. Section E.7.1 provides an annotated example of coding a Thread/SERIES Table of Non-Standard Processing definitions using these macros. Section E.7.2 provides detailed documentation of the Thread/SERIES macro set with which to define non-standard processing criteria, while Section E.7.3 illustrates a job to assemble and link edit a Table of Non-Standard Processing as a Thread/SERIES load module.

E.7.1 TTS$TNS - The Sample Table of Non-Standard Processing

The source module within member TTS$TNS of the TTSCNTL library illustrates how to define threads for which non-standard processing is required. It further illustrates how to specify the exit routine to be invoked when Thread/SERIES identifies a thread which meets the non-standard processing criteria.

```
        TITLE 'TTS$TNS - THREAD/SERIES NON-STANDARD PROCESSING TABLE'
TTS$TNS TTS#TNSI                                              (1)
        TTS#TNS PLAN=TTSIVPS,EXIT=TTS3SXR                     (2)
        TTS#TNS PLAN=TTSIVPI,EXIT=TTS3CLS                     (3)
        TTS#TNS PLAN=F2PLN361,EXIT=TTS3CLS                    (4)
        TTS#TNS DSN=DSN,          DB2 subsystem name          (5)       +
              PLAN=PPPP_PPPP,     Plan name for thread                  +
              PGM=YYY%YYY,        program name (package or dbrm)       +
              AUTHID=AAAA_A_A,    DB2 authorization id                 +
              JOBNAME=JJJJ%JJJ,   jobname                              +
              CORR=12346789Ø12,   correlation id                       +
              CONNAME=CICS_XXX,   connection name                      +
              CONTYPE=6,          connection type                      +
              EXIT=XXXXXXXXX      exit name
        TTS#TNSF                                              (6)
        END TTS$TNS                                           (7)
```

where

(1) The label field of the TTS#TNSI macro provides the name of the CSECT. TTS#TNSI, the initial macro of the TTS#TNSx macro set, is described in Section E.7.2.1.

(2) The TTS#TNS macro (described in Section E.7.2.2) provides a single non-standard processing specification. This instance of the TTS#TNS macro specifies that when a violation occurs with a thread whose plan name is TTSIVPS, Thread/SERIES should invoke the sample exit routine named TTS3SXR.

(3) This instance of the TTS#TNS macro specifies that when a violation occurs with a thread whose plan name is TTSIVPI, Thread/SERIES should invoke the exit routine named TTS3CLS. This RAI supplied exit routine cancels the ISPF logical screen associated with a DB2 thread.

(4) This instance of the TTS#TNS macro directs Thread/SERIES to invoke the exit routine named TTS3CLS when any thread whose plan name is F2PLN361 incurs a violation.

(5) This instance of the TTS#TNS macro illustrates TTS#TNS coding using the SQL wild card characters _ and %. The operands of the TTS#TNS macro are described in detail in Section E.7.2.2.

(6) The TTS#TNSF macro (described in Section E.7.2.3) must be coded last to generate the Thread/SERIES Table of Non-Standard Processing.

(7) The END instruction, as the last source statement, terminates the assembly of the program.

Figure E.9 The Sample Table of Non-Standard Processing

E.7.2 The TTS#TNSx Macro Set

E.7.2.1 The TTS#TNSI Macro

TTS#TNSI is the ***initial*** macro of the TTS#TNSx macro set. The TTS$TNSI macro must be specified first, before any other TTS#TNSx macros are coded. TTS#TNSI generates the CSECT header for a Thread/SERIES Table of Non-Standard Processing definitions. The label field of the TTS#TNSI macro provides the name of the CSECT. The default CSECT name (if a label is omitted) is TTS$TNS. The TTS#TNSI macro has no other operands.

E.7.2.2 The TTS#TNS Macro

The TTS#TNS macro defines a thread or set of threads for which Thread/SERIES should perform non-standard processing. The TTS#TNS macro also identifies the exit routine to be called when such a thread is encountered. TTS#TNS macros must be specified *after* the TTS#TNSI macro. Figure E.10 illustrates the TTS#TNS macro and its operands.

```
TTS#TNS &DSN=,              DB2 Subsystem Name                 +
        &PLAN=,             plan name (may specify pattern)    +
        &PGM=,              program name (package or dbrm)     +
        &AUTHID=,           DB2 authorization id               +
        &JOBNAME=,          jobname                            +
        &CORR=,             correlation id                     +
        &CONNAME=,          connection name                    +
        &CONTYPE=,          connection type                    +
        &EXIT=,             exit name                          +
        &DOC=NO
```

Figure E.10 The TTS#TNS Macro and its Operands

Except for the EXIT operand which is required, all the other thread selection operands of the TTS#TNS macro are optional.

DSN specifies the name of the DB2 subsystem in which the thread may run. If the DSN operand is blank or omitted, then a thread running on *any* DB2 subsystem is eligible for selection on the basis of the remaining criteria. If DSN is specified, then only threads executing on that particular DB2 subsystem are eligible for selection.

PLAN specifies the name of the DB2 application plan associated with the thread. Plan name may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

PGM specifies the name of the program (corresponding to a DB2 package or DBRM) that is currently associated with the thread. PGM may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

AUTHID specifies the DB2 authorization ID associated with the thread. AUTHID may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

JOBNAME specifies the name of the job associated with the thread. JOBNAME may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

CORR specifies the DB2 Correlation ID associated with the thread. CORR may be up to 12 characters long and may specify a pattern using the SQL wild card characters _ and %.

CONNAME specifies the DB2 connection name associated with the thread. CONN may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

CONTYPE specifies the DB2 connection type associated with the thread. CONTYPE, if specified, must correspond to one of the allowable values for the field labeled QWHCATYP in mapping macro DSNDQWHC in the SDSNMACS library.

For example, a value of `X'1'` denotes TSO attach while `X'2'` indicates DB2 call attach. No pattern may be specified for CONTYPE.

EXIT is the only *required* operand. It identifies the exit routine to be called when a thread meeting the selection criteria specified by the other operands is encountered. The named EXIT must in turn be defined in a Thread/SERIES Exit routine table.

E.7.2.3 The TTS#TNSF Macro

TTS#TNSF is the ***final*** macro of the TTS#TNSx macro set. The TTS#TNSF macro must be coded last to generate the Thread/SERIES Table of Non-Standard Processing definitions. The TTS#TNSF macro has no operands.

E.7.3 Building the TTS$TNSF Load Module

Member TTSJTNS of the TTSCNTL library contains the jobstream illustrated in Figure E.11. This job invokes the TTSPAL procedure to assemble and link edit the Thread/SERIES Table of Non-Standard Processing definitions (Load module TTS$TNS). Be sure to edit JCL member TTSJTNS with CAPS ON since it contains comments in lowercase.

```
//jobname JOB (account)                                          (1)
//* <optional JCLLIB statement>                                   (2)
//TTS$TNS  EXEC TTSPAL,      <- Invoke the procedure TTSPAL
//         MEMBER=TTS$TNS    <- Thread/SERIES tbl of non std processing
```

where

(1) Provide a valid JOB statement.

(2) Add a JCLLIB statement to identify the TTSCNTL dataset if the procedures TTSPAL and RAIAL do not reside within one of your site's catalogued procedure libraries.

Figure E.11 Assemble and Link Edit the Sample Table of Non-Standard Processing

One or more load modules can be created, each of which defines a discrete Thread/SERIES Table of Non-Standard Processing. The default name is TTS$TNS. This default may be overridden by rebuilding the Thread/SERIES Table of System Defaults (load module TTS$TSD) as described in Section 20.14. Alternatively, you can specify use of another Non-Standard Processing table through the Thread/SENTRY MONITOR statement (described in Chapter 10).

Appendix F

Defining Thread/SERIES

No Action Criteria

Thread/SERIES allows you to predefine criteria for which Thread/SERIES actions should be implicitly suppressed. An action is suppressed when a thread selected by Thread/SENTRY or the Thread/STOPPER ISPF dialog *matches* an entry in the currently active No Action table described below.

Thread/SERIES provides three macros for the purpose of defining threads for which one or more actions are inappropriate. All macros reside in the macro library whose low level qualifier is TTSMACS. Section F.1 provides an annotated example of coding a Thread/SERIES Table of No Action definitions using these macros. Section F.2 provides detailed documentation of the Thread/SERIES macro set with which to define No Action processing criteria, while Section F.3 illustrates a job to assemble and link edit a Table of No Action definitions as a Thread/SERIES load module.

F.1 TTS$TNA - The Sample Table of No Action definitions

A sample Thread/SERIES Table of No Action definitions appears in Figure F.1.

```
         TITLE 'TTS$TNA - Thread/SERIES No Action definitions'
TTS$TNA  TTS#TNAI                                                    (1)
         TTS#TNA JOBNAME=BACKUP,   No actions are allowed against    (2)+
               ACTION_SUPPRESSED=ANY  job BACKUP
         TTS#TNA JOBNAME=RTS%,     Never Cancel, Force or Quiesce    (3)+
               ACTION_SUPPRESSED=(CANCEL,FORCE,QUIESCE) any RTS jobs
         TTS#TNA PLAN=SRS%,        Never quiesce address spaces      (4)+
               JOBNAME=RJS%,       when job name is like 'RJS%' and   +
               ACTION_SUPPRESSED=QUIESCE   plan name is like 'SRS%'
*
*        The following illustrates TTS#TNA coding using
*        the SQL wild card characters _ and %.
*                                                                    (5)
         TTS#TNA DSN=DSN,          DB2 subsystem name                 +
               PLAN=PPPP_PPP,      Plan name for thread               +
               PGM=YYY%YYY,        program name (package or DBRM)     +
               AUTHID=AAAA_A_A,    DB2 authorization id               +
               JOBNAME=JJJJ%JJJ,   jobname                            +
               CORR=12346789012,   correlation id                     +
               CONNAME=CICS_XXX,   connection name                    +
               CONTYPE=6,          connection type                    +
               ACTION_SUPPRESSED=KILL  Never KILL such CICS thread
*
         TTS#TNAF                                                    (6)
         END TTS$TNA                                                 (7)
```

where

(1) The label field of the TTS#TNAI macro provides the name of the CSECT. TTS#TNAI, the initial macro of the TTS#TNAx macro set, is described in Section F.2.1.

(2) The TTS#TNA macro (described in Section F.2.2) provides a single No Action specification. This instance of the TTS#TNA macro invalidates any action against a thread whose job name is BACKUP.

(3) This instance of the TTS#TNA macro suppresses the multiple actions CANCEL, FORCE and QUIESCE against any thread whose job name matches the pattern 'RTS%'.

(4) This instance of the TTS#TNA macro suppresses the QUIESCE action against any thread whose job name is like 'RJS%' and whose plan name is like 'SRS%'.

(5) This instance of the TTS#TNA macro illustrates TTS#TNA coding using the SQL wild characters _ and %. The operands of the TTS#TNA macro are described in detail in Section F.2.2.

(6) The TTS#TNAF macro (described in Section F.2.3) must be coded ***last*** to generate the Thread/SERIES Table of No Action definitions.

(7) The END instruction, as the last source statement, terminates the assembly of the program.

Figure F.1 The Sample Table of No Action definitions

F.2 The TTS#TNAx Macro Set

F.2.1 The TTS#TNAI Macro

TTS#TNAI is the initial macro of the TTS#TNAx macro set. The TTS#TNAI macro must be specified first, before any other TTS#TNAx macros are coded. TTS#TNAI generates the CSECT header for a Thread/SERIES Table of No Action definitions. The label field of the TTS#TNAI macro provides the name of the CSECT. The default CSECT name (if a label is omitted) is TTS$TNA. The TTS#TNAI macro has no other operands.

F.2.2 The TTS#TNA Macro

The TTS#TNA macro defines a thread or set of threads for which Thread/SENTRY and the Thread/STOPPER ISPF dialog should suppress one or more predefined actions. TTS#TNA macros must be specified after the TTS#TNAI macro. The following figure illustrates the TTS#TNA macro and its operands.

```
TTS#TNA &DSN=,            DB2 Subsystem Name                 +
      &PLAN=,             Plan name                          +
      &PGM=,              Program name (package or DBRM)     +
      &AUTHID=,           DB2 authorization id               +
      &JOBNAME=,          Jobname                            +
      &CORR=,             Correlation ID                     +
      &CONNAME=,          Connection name                    +
      &CONTYPE=,          Connection type                    +
      &ACTION_SUPPRESSED=,  One or more actions to suppress  +
      &DOC=NO
```

Figure F.2 The TTS#TNA Macro and its Operands

All the thread selection operands of the TTS#TNA macro are optional but the ACTION_SUPPRESSED operand is required.

DSN specifies the name of the DB2 subsystem in which the thread may run. If the DSN operand is blank or omitted, then a thread running on *any* DB2 subsystem is eligible for selection on the basis of the remaining criteria. If DSN is specified, then only threads executing on that particular DB2 subsystem are eligible for selection.

PLAN specifies the name of the DB2 application plan associated with the thread. Plan name may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

PGM specifies the name of the program (corresponding to a DB2 package or DBRM) that is currently associated with the thread. PGM may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

AUTHID specifies the DB2 authorization ID associated with the thread. AUTHID may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

JOBNAME specifies the name of the job associated with the thread. JOBNAME may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

CORR specifies the DB2 Correlation ID associated with the thread. CORR may be up to 12 characters long and may specify a pattern using the SQL wild card characters _ and %.

CONNAME specifies the DB2 connection name associated with the thread. CONN may be up to 8 characters long and may specify a pattern using the SQL wild card characters _ and %.

CONTYPE specifies the DB2 connection type associated with the thread. CONTYPE, if specified, must correspond to one of the allowable values for the field labeled QWHCATYP in mapping macro DSNDQWHC in the SDSNMACS library. For example, a value of X'1' denotes TSO attach while X'2' indicates DB2 call attach. Additionally, you may use the following RAI extensions to distinguish between TSO batch and foreground threads:

CONTYPE=254	TSO foreground
CONTYPE=255	TSO batch

No pattern may be specified for CONTYPE.

ACITION_SUPPRESSED
is the only required operand. It identifies one or more actions that are inappropriate and will be suppressed when a thread meeting the selection criteria specified by the other operands is encountered.

Valid settings are

CANCEL	Cancel thread
DUMP	Cancel with Dump
FORCE	Force (abend thread)
KILL	Kill (Force Purge) CICS transaction
PGM	Call site written program
QUIESCE	***Quiesce thread's address space. This action suppresses action RESUME as well.***
VTAM	Remove thread's VTAM or TCP/IP connection
WARN	Issue a warning message
ANY	All of the above

F.2.3 The TTS#TNAF Macro

TTS#TNAF is the ***final*** macro of the TTS#TNAx macro set. The TTS#TNAF macro must be coded last to generate the Thread/SERIES Table of No Action definitions. The TTS#TNAF macro has no operands.

F.3 Building the TTS$TNA Load Module

Member TTSJTNA of the TTSCNTL library contains the jobstream illustrated in the following figure. This job invokes the TTSPAL procedure to assemble and link edit the Thread/SERIES Table of No Action definitions (Load module TTS$TNA). Be sure to edit JCL member TTSJTNA with CAPS ON since it contains comments in lowercase.

```
//jobname JOB (account)                                          (1)
//* <optional JCLLIB statement>                                   (2)
//TTS$TNA  EXEC TTSPAL,      <- Invoke the procedure TTSPAL
//         MEMBER=TTS$TNA    <- Thread/SERIES table of No Actions
```

where

(1) Provide a valid JOB statement.

(2) Add a JCLLIB statement to identify the TTSCNTL dataset if the procedures TTSPAL and RAIAL do not reside within one of your site's catalogued procedure libraries.

Figure F.3 Assemble and Link Edit the Sample Table of No Action definitions

Appendix G

Defining Thread/SENTRY Table of Group ID Definitions

The Thread/SENTRY Table of Group ID definitions enable you to target multiple DB2 authorization IDs with a single LIMIT or EXCLUDE policy. Thread/SENTRY provides three macros to aggregate numerous authorization IDs into groups. All macros reside in the macro library whose low level qualifier is TTSMACS. Section G.1 provides an annotated example of coding a Thread/SENTRY Table of Group ID definitions using these macros. Section G.2 provides detailed documentation of the Thread/SENTRY macro set with which to define said groups, while Section G.3 illustrates a job to assemble and link edit a Table of Group ID definitions as a Thread/SENTRY load module. Finally, Section G.4 provides an example of how to reference the defined groups in the LIMIT policy.

G.1 TTS$TGI - The Sample Table of Group ID definitions

A sample Thread/SENTRY Table of Group ID definitions appears in Figure G.1.

```
      TITLE 'TTS$TGI - Thread/SENTRY Table of Group ID definitions'

TTS$TGI  TTS#TGIH                                                    (1)
*
RAIG1    TTS#TGI AUTHID=(RAI016,RAI021,RAI039,RAI042)                (2)
*
RAIG2    TTS#TGI AUTHID=(RAI01%,RAI05%,R_I99%)                       (3)
*
TEST     TTS#TGI AUTHID=(IBMUSER,                                    (4)+
               'This is a quite long primary authorization string ID')
*
         TTS#TGIF                                                    (5)
         END TTS$TGI                                                 (6)
```

where

(1) The label field of the TTS#TGIH macro provides the name of the CSECT. TTS#TGIH, the initial macro of the TTS#TGIx macro set, is described in Section G.2.1.

(2) The TTS#TGI macro (described in Section G.2.2) provides a single group specification named RAIG1. This instance of the TTS#TGI macro combines four distinct authorization IDs into one group whose name is RAIG1.

(3) This instance of the TTS#TGI macro illustrates TTS#TGI coding using the SQL wild characters _ and %, and targets any DB2 authorization ID like ‘RAI01%’, ‘RAI05%’ or ‘R_I99%’. The group name is RAIG2.

(4) This instance of the TTS#TGI macro illustrates TTS#TGI coding when a DB2 authorization ID contains one or more blanks.

(5) The TTS#TGIF macro (described in Section G.2.3) must be coded ***last*** to generate the Thread/SENTRY Table of Group ID definitions.

(6) The END instruction, as the last source statement, terminates the assembly of the source module.

Figure G.1 The Sample Table of Group ID definitions

G.2 The TTS#TGIx Macro Set

G.2.1 The TTS#TGIH Macro

TTS#TGIH is the initial macro of the TTS#TGIx macro set. The TTS#TGIH macro must be specified *first*, before any other TTS#TGIx macros are coded. TTS#TGIH generates the CSECT header for a Thread/SENTRY Table of Group ID definitions. The label field of the TTS#TGIH macro provides the name of the CSECT. The required CSECT name is TTS$TGI. The TTS#TGIH macro has no other operands.

G.2.2 The TTS#TGI Macro

The TTS#TGI macro combines one or more DB2 authorization IDs into a single group and must follow the header macro TTS#TGIH. (see figure G.1). The label field of the TTS#TGI macro must be specified. It identifies the group name and may be up to 32 characters long. The AUTHID operand specifies one or more DB2 authorization IDs separated by comas. AUTHID may be up to 128 characters long and may specify a pattern using the SQL wild card characters _ and %. The TTS#TGI macro has no other operands.

G.2.3 The TTS#TGIF Macro

TTS#TGIF is the ***final*** macro of the TTS#TGIx macro set. The TTS#TGIF macro must be coded last to generate the Thread/SENTRY Table of Group ID definitions. The TTS#TGIF macro has no operands.

G.3 Building the TTS$TGI Load Module

Member TTSJTGI of the TTSCNTL library contains the jobstream illustrated in the following figure. This job invokes the TTSPAL procedure to assemble and link edit the Thread/SENTRY Table of Group ID definitions (Load module TTS$TGI). Be sure to edit JCL member TTSJTGI with CAPS ON since it contains comments in lowercase.

```
//jobname JOB (account)                                    (1)
// SET TTSHLQ='?ttshlq?'                                   (2)
//PROCLIB JCLLIB ORDER=&TTSHLQ..TTSCNTL
//TTS$TGI  EXEC TTSPAL,      <- Invoke the procedure TTSPAL
//         MEMBER=TTS$TGI,   <- Thread/SENTRY Table of Group IDs
//         HLQ=&TTSHLQ
```

where

(1) Provide a valid JOB statement.

(2) Replace ?ttshlq? with the high level qualifier of Thread/SERIES product libraries restored onto DASD at your site.

Figure G.2 Assemble and Link Edit the Sample Table of Group ID definitions

G.4 Using TTS$TGI groups

In the following example, the LIMIT policy named POLICY1 will cancel a thread whose primary authorization ID is defined within the sample RAIG1 group (see figure G.1) and issues more than 10,000 GETPAGE requests.

```
LIMIT
  PID(POLICY1)
  GROUPID(RAIG1)
  MAX_GETPAGES(10000)
  ACTION(CANCEL)
```

The sample RAIG1 group includes the following DB2 authorization IDs: RAI016, RAI021, RAI039 and RAI042.

GROUPID may specify a pattern using the SQL wild card characters _ and %. E.g. specifying GROUPID(RAIG%) would target the authorization IDs associated with both the RAIG1 and RAIG2 groups defined in the sample Table of Group ID definitions appearing above in figure G.1.

The GROUPID operand can be specified in both LIMIT and EXCLUDE policies.

Appendix M

Thread/STOPPER and

Thread/SENTRY Messages

Messages appear in alphabetical order.

Call Attach Facility related Messages

```
CAF010 - Required parameter <ssid> or :HV is missing. Identify DB2 subsystem.
CAF011 - Open failed - DSN=&THRDSN Plan=&THRPLAN RC=&ALIRC CD=&ALIRRC
CAF012 - Open failed - DSN=&THRDSN Plan=&THRPLAN RC=&ALIRC CD=&ALIRRC
CAF013 - Failed to identify DB2 application thread - CAF stmt syntax error
CAF201 - CAF CLOSE issued when there was no successful Multi/CAF OPEN
CAF202 - CAF DISCONNECT issued when there was no successful Multi/CAF CONNECT
CAF203 - Multi/CAF control blocks reset successful - ready to re-CONNECT'
CAF204 - DB2 subsystem &THRDSN is not active
CAF205 - DB2 subsystem &THRDSN is not defined
CAF206 - DB2 connection type parameter is in error
CAF207 - User ID is not authorized to use DB2
CAF208 - Task Control Block not connected to DB2
CAF209 - Connection failed - DB2=&&THRDSN is terminating
CAF210 - Application plan=&&THRPLAN is not valid
CAF212 - Maximum connections reached, attempt DB2 connection later
CAF213 - Release level mismatch between CAF modules and DB2 subsystem &THRDSN
CAF214 - User ID not authorized to use DB2 application plan &THRPLAN
CAF215 - Resource unavailable
CAF216 - CONNECT to DB2 subsystem failed - critical error - no retries will be
          attempted
CAF217 - Failed to CONNECT to DB2 subsystem, holt connection retries
CAF218 - Multi/CAF DISCONNECT command failed
CAF219 - Multi/CAF OPEN plan failed - TRANSLATE was issued
CAF220 - Multi/CAF CLOSE application plan failed
CAF221 - Multi/CAF OPEN application plan failed - TRANSLATE was not issued
CAF901 - CAF error: RC = &ALIRC REASON Code = &ALIRRC
CAF902 - Thread = &THRUID DB2 Subsystem &THRDSN Plan &THRPLAN
```

MVS related Messages

```
PMV000 - Allocation error.  r15 = &PMVRC, error code = &PMVEC, reason code = &PMVIC
PMV001 - &ZUSER lacks authority to access &LMDFQDSN
PMV002 - Abend &PMVAC Reason &PMVARC occurred during open processing
PMV003 - File is not open after return from Open SVC
PMV004 - CAMLST/LOCATE system service failed to retrieve DS volume serial #
PMV005 - Dataset not cataloged
PMV006 - Failed to acquire a buffer for the open file
PMV007 - Camlst/obtain system service failed to retrieve dataset label
PMV009 - Try LISTA STatus command
PMV011 - Invalid project - Reenter Project name
PMV012 - Invalid library - Reenter library name
PMV013 - Enter Type Qualifier
PMV014 - Enter closing parenthesis after member name
PMV015 - Member name must be between 1 and 8 characters long
PMV016 - No System Diagnostic Work Area available
PMV017 - Abend profile formatted from MVS System Diagnostic Work Area

PMV050 - Link to module &PMVMOD failed.  Abend = &PMVAC  Reason = &PMVARC
PMV051 - Attach of module &PMVMOD failed.  Abend = &PMVAC Reason = &PMVARC
PMV052 - Load of module &PMVMOD failed.  Abend = &PMVAC Reason = &PMVARC
PMV053 - A severe error occurred in attempting to invoke exec &PMVEXEC
PMV054 - Recursive abend detected - PMVRTM1 will NOT attempt retry
```

RFARMD Messages

```
RAI000 - Report process unable to match issuing macro name &RFAMACRO
```

RAI Password Verification Messages

```
RAI002E - First specify &RAIPROD / &RAIVRM CPU passwords
RAI003E - &RAIPROD / &RAIVRM usage has expired on CPU &RAICPU using passwords
          RAIPSWD1 / &RAIPSWD2 - contact vendor
RAI004W - &RAIPROD / &RAIVRM usage has &RAIDAYS days remaining on CPU &RAICPU using
          passwords &RAIPSWD1 / &RAIPSWD2 - contact vendor
RAI005E - &RAIPROD Version &RAIVRM not authorized on CPU &RAICPU using passwords
          &RAIPSWD1 / &RAIPSWD2 - contact vendor
RAI006E - CPU serial &RAICPU not authorized to run &RAIPROD Release &RAIVRM using
          passwords &RAIPSWD1 / &RAIPSWD2 - contact vendor
```

Report Process Messages

```
RAI500I - Report process for &&PVTNAME is starting
RAI501I - Report process start completed for &&PVTNAME
RAI502I - Report process for &&PVTNAME is stopping
RAI503E - RFARMX diagnostic process ended prematurely or was never attached at all
RAI504E - jobstep will abend with U2001 and reason code 00DB2000
```

RAI DB2 related Messages

RDB000 - Failed to acquire storage for DB2 related control block
RDB001 - Process failed to acquire a large IFI return area

RFA Messages

RFA010E - process &&PVTNAME did not complete after being signaled to stop
RFA011E - process &&PVTNAME is being forcibly detached

Messages pertaining to the Call Attach Facility Interface component

RSQ010 - Call attach facility error: Return code &CAFRC1 Reason code &CAFRC2
RSQ011 - CAF CLOSE issued when there was no active OPEN
RSQ012 - CAF DISCONNECT issued when there was no active CONNECT
RSQ013 - CAF control blocks reset successfully - Ready to reconnect. This is NOT an error!
RSQ014 - The referenced DB2 subsystem named &CAFDSN is not active
RSQ015 - The referenced DB2 subsystem named &CAFDSN does not exist
RSQ016 - DB2 connection type parameter is in error
RSQ017 - ID &ZUSER is not authorized to use DB2
RSQ018 - Task control block not connected to DB2
RSQ020 - Application plan named &CAFPLAN is not valid
RSQ021 - Either your DB2 connection ID is not authorized to use the plan named &CAFPLAN or this plan does not exist.
RSQ022 - Resources are unavailable for plan &CAFPLAN as described by message DSNT500I. The plan is probably not bound on subsystem &CAFDSN
RSQ023 - Maximum connections reached. Attempt DB2 connection later.
RSQ024 - Release level mismatch between CAF routines and the DB2 subsystem named &CAFDSN

Main prolog messages

TSN010 - Thread/SENTRY requires a password load module named TSN$TPV in order to operate but this module was not found. Please ensure this module is in the standard MVS search order. If module TSN$TPV does not exist, call Relational Architects or your local representative for passwords.
TSN011 - Thread/SENTRY password load module TSN$TPV is invalid. Refer to the Write to Programmer message(s) in the JES listing that describe WHY the passwords are invalid.

TTS010 - Thread/SENTRY failed to acquire a large block of non lifo storage
TTS011 - Thread/SENTRY main prolog failed to #GMAIN a control block
TTS012 - Thread/STOPPER failed to acquire a large IFI return area
TTS013 - Thread/STOPPER failed to create an ISPF table of DB2 subsystems
TTS014 - Thread/STOPPER failed to create an ISPF table of DB2 threads
TTS015 - Thread/STOPPER failed to start the RAI Call Attach Facility
TTS016 - As a precaution, ISPF split screen mode has been DISABLED so the Thread/STOPPER may be connect to multiple DB2 subsystems simultaneously. This prevents the use of a different DB2 attachment mechanism in another

ISPF logical screen whose use in such a multiple connection context can cause severe problems.

TTS017 - Thread/STOPPER requires a password load module named TTS$TPV in order to operate but this module was not found. Please ensure this module is in the standard MVS search order. If module TTS$TPV does not exist, call Relational Architects or your local representative for passwords.

TTS018 - Thread/STOPPER password load module is invalid. The previously issued line mode (Write to Programmer) message(s) describe WHY the passwords are invalid.

TTS019 - Thread/STOPPER failed to acquire a token scan table for its use

Subsystem selection messages

TTS020 - No DB2 subsystems are currently active

TTS021 - No DB2 subsystems are defined (none are active or even inactive)

TTS022 - Select active DB2 subsystem(s) with "S" or press END key to exit

TTS023 - A DB2 subsystem name must be specified in single subsystem mode

TTS024 - DB2 subsystem &wqaldsn is not currently active in MVS system &zsysid

TTS025 - DB2 subsystem &wqaldsn is not defined within MVS system &zsysid

TTS026 - Select one or more active DB2 subsystems with the "S" row command. Enter the END command to process DB2 subsystem selections. Alternatively, enter the END command without selecting any subsystems to exit immediately.

TTS027 - #LISTADD error within TTSCMDSN - contact vendor

TTS028 - ALESERV request failed within TTSCMDSN - contact vendor

Thread qualification messages

TTS030 - Qualify which DB2 threads are of interest and press ENTER. To see all threads, press ENTER without supplying any qualifications. Press the END key to return to the DB2 subsystem selection display.

TTS031 - Thread/STOPPER terminated since no DB2 subsystems were selected

TTS032 - Load module libraries for DB2 subsystem &TDSDSN must be allocated to the DDname &TDSDSN.LOAD

TTS033 - Internal error - CR_THREAD request for DB2 subsystem &tdsdsn failed

TTS034 - Internal error - Failed to connect to DB2 subsystem &tdsdsn

TTS035 - IFI error: Return code = &IFCARC1, Reason code = &IFCARC2

TTS036 - No threads match the qualifications specified. Please respecify or simply provide NO qualifications to view all current threads.

TTS037 - TTS internal error - failed to remove thread &tdsthr@

TTS038 - TTS internal error - what should this text say?

TTS039 - Internal error - failed to close MONITOR plan for subsystem &tdsdsn

DB2 subsystem selection messages

TTS040 - DB2 subsystem(s) selected. Continue to scroll or ENTER to process

TTS041 - DB2 subsystem &tdsdsn is not active

TTS042 - The only valid row command is S

TTS043 - Qualify which DB2 threads are of interest and press ENTER. To see all threads, press ENTER without supplying any qualifications. Press the END key to exit Thread/STOPPER altogether.

DB2 thread selection messages

TTS050 - Press ENTER to update statistics, enter CMD letter or hit PF3 to exit
TTS051 - Local thread (DB2 SSID: &tdtdsn, Plan: &tdtplan, Correlation ID: &tdtcorr, DB2 auth ID: &tdtauth) terminated successfully
TTS052 - Thread &tdtcorr &tdtplan &tdtconn already terminated
TTS053 - (S)elect a thread for inspection and possible cancellation
TTS054 - Job &tdtjobnm / Address Space &tdtasid is no longer active
TTS055 - DB2 thread for Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm, Address space: &tdtasid, TCB address &tdttcb@ is no longer active
TTS056 - Internal error on #CALLRTM service for TCB &tdttcb@ / address space &tdtasid / jobname &tdtjobnm
TTS057 - DB2 thread for Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm, Address space: &tdtasid, TCB address &tdttcb@ was abended with system code &tmvac# and abend reason code &tmvarc#'
TTS058 - The combination of thread type and thread status make the thread ineligible for cancellation
TTS059 - No action taken for the DB2 thread associated with Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm and Address space: &tdtasid

DB2 thread mapping messages

TTS060 - TTS internal error - not processing IFCID 0148'
TTS061 - TTS internal error - not processing an IFI Correlation Header
TTS062 - The two megabyte buffer supplied by Thread/STOPPER on a IFI READS request exceeds the maximum length supported by this maintenance level of DB2 V2.3. Ask your DB2 systems programmer to apply fixes for DB2 APARs PN28197 and PN34186 that will enable Thread/STOPPER to operate.
TTS063 - IFI READS error: Return code = &IFCARC1, Reason code = &IFCARC2
TTS064 - The only valid selection command is "S"
TTS065 - The only valid selection command is "S"
TTS066 - A TTS IFI record SNAP request has failed
TTS067 - READS request for 0148 record qualified by WQALUNIQ not found in IFI return area. Notify vendor.

DB2 thread termination messages

TTS070 - The selected thread is not eligible to be canceled because it is already being deleted by DB2. If essential, use the FORCE command.
TTS071 - TTS internal error - failed to find matching DB2 subsystem row
TTS072 - TTS internal error - cross memory move failed
TTS073 - Failed to switch to the thread for DB2 subsystem &TDSDSN
TTS074 - DB2 subsystem &tdsdsn has accepted the request to terminate IBM DB2 utility &tduutil, ID &tduuid, User &tduuser at its next cleanup point
TTS075 - Thread/STOPPER does not support BMC DB2 utilities at this time. Use the BMCDSN command processor instead
TTS076 - IBM DB2 utility &tduutil, ID &tduuid, User &tduuser was successfully terminated at a cleanup point
TTS077 - IBM DB2 utility &tduutil, ID &tduuid, User &tduuser is NOT yet canceled. Press ENTER to track cancellation progress or issue a (F)orce command.

```
TTS078 - The selected thread is not eligible to be canceled or forced right now
         because it is running under a task within the DB2 system services address
         space for &tdsdsn (ASID &tdsasid). If the jobname changes back from
         &tdsproc, you can try the (C)ancel command again.  Otherwise, DB2 may be
         recognizing End-of-Task or End-of-Memory for the thread.
TTS079 - Thread/STOPPER should not be canceled.  Simply exit the dialog.
```

IBM DB2 utility threads, messages

```
TTS080 - Internal -DISPLAY UTILITY error: Return code = &IFCARC1  Reason code =
         &IFCARC2
TTS081 - The DB2 utility thread you selected to terminate Jobname &tdtjobnm,
         Correlation ID &tdtcorr and DB2 Auth ID &tdtauth is not presently known to
         DB2 subsystem &tdsdsn. If the thread 'exists, wait a few moments and try the
         (C)ancel command again.  By then the thread may be registered as an active
         UTILITY.
TTS082 - Confirm DB2 utility termination or enter the END command to bypass
TTS083 - Select which DB2 utility you wish to terminate by keying an "S" in its row
         selection field.  Enter the END command to bypass
TTS084 - TERM UTILITY command error: Return code = &IFCARC1 Reason code = &IFCARC2
TTS085 - Specify Y to cancel DB2 utility thread or N to bypass
TTS090 - Failed to create an ISPF table for DB2 utility threads
TTS091 - &tmvtext
```

DB2 utilities, messages

```
TTS100 - DB2 utility thread selected.  Press END key to continue
TTS101 - Select just one DB2 utility job from among the candidates displayed below
         that corresponds to DB2 thread &tdtcorr, &tdtjobnm, &tdtauth
TTS102 - The only valid row command is "S"
```

DB2 thread termination messages

```
TTS110 - Your DB2 thread (Plan: &tdtplan, DB2 auth ID: &tdtauth)
TTS111 - is being deliberately terminated.  Your DB2 application will abend
TTS112 - with system abend code &tmvac# and an abend reason code of &tmvarc#.
TTS113 - The selected thread no longer exists.  No action taken.
TTS114 - The TSO user failed to receive &tmvlml# (of 3) lines of abend notification
         text (because no terminal buffers were available).
TTS115 - No change in inactive DDF thread statistics
TTS116 - Inactive (IDLE) threads list could not be obtained
TTS117 - Failed to acquire pseudo QWIW and QWT0 - notify vendor
TTS118 - Error occured during non standard processing initialization
```

Thread Summary Primary Commands, messages

```
TTS120 - No primary commands are supported from this display at this time
TTS121 - Threads sorted in plan name order
```

DB2 TRACE Commands, messages

TTS130 - You lack the authority to issue a -START TRACE command (through the Thread/STOPPER) on DB2 subsystem &tdsdsn..
Ask your DB2 security administrator to grant you at least MONITOR1 and preferably MONITOR2 authority. Otherwise, you cannot utilize the Thread/STOPPER.

TTS131 - STOP TRACE request on DB2 subsystem &tdsdsn failed

TTS133 - START TRACE request on DB2 subsystem &tdsdsn failed

TTS134 - DB2 command on subsystem &tdsdsn failed

Thread/STOPPER initialization messages

TTS140 - The Relational Architects extended SVC is not properly installed. Thread/STOPPER requires this SVC to operate. Check with your Thread/STOPPER installer and/or MVS systems programmer to ensure the RAI extended SVC is link edited into SYS1.LPALIB or one of its extensions. Futher, the SVC must become part of the live LPA through an IPL with the CLPA option. See the Thread/STOPPER installation chapter for details.

Canceling discrete threads

TTS150 - The selected thread (ACE token &tdtace@, plan &tdtplan, auth ID &tdtauth) no longer exists on subsystem &tdtdsn

TTS151 - START TRACE request on DB2 subsystem &tdsdsn failed

TTS152 - STOP TRACE request on DB2 subsystem &tdsdsn failed

TTS153 - SQL INSERT to Thread/STOPPER audit table failed with SQLCODE = &sqlcode

TTS154 - DB2 has queued thread (plan &tdtplan, auth ID &tdtauth) for deletion on subsystem &tdtdsn as a consequence of cancellation method: "&tcpmeth"

TTS155 - Cancellation method: "&tcpmeth" did not work for (plan &tdtplan, auth ID &tdtauth) on subsystem &tdtdsn

TTS156 - The thread you selected to cancel (Plan &tdtplan, Auth ID &tdtauth Subsystem &tdtdsn) terminated BEFORE Thread/STOPPER took action to halt it

TTS157 - You requested VTAM session cancellation for a thread that is not distributed

TTS158 - Thread for plan &tdtplan and auth ID &tdtauth is being deleted on subsystem &tdtdsn as a consequence of cancellation method: "&tcpmeth"

Initialization messages

TTS160 - The load module containing Thread/STOPPER systems defaults (module name TTS$TSD) could not be found in the standard MVS search order. Contact your Thread/STOPPER product administrator

DB2 thread cancellation messages

TTS170 - Cancellation in progress for the DB2 thread associated with Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm, Address space: &tdtasid and TCB address &tdttcb@. Thread/STOPPER waiting &tmvsec# seconds before checking thread status

TTS171 - DB2 thread for Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm, Address space: &tdtasid and TCB address &tdttcb@ was cancelled successfully. The thread task was abended with system code &tmvac# and abend reason code &tmvarc#

TTS172 - The thread for Plan: &tdtplan, DB2 Auth ID: &tdtauth, Job: &tdtjobnm, Address space: &tdtasid and TCB address &tdttcb@ is currently being deleted by DB2. Thread/STOPPER will recheck status in &tmvsec# seconds.

TTS173 - Thread still exists. Press ENTER to check status or try another CMD

Terminating distributed DB2 threads

TTS180 - CANCEL successful for the distributed threads associated with the Logical Unit of Work ID

TTS181 - The distributed threads associated with the Logical Unit of Work ID' (LUWID) &tdtnetid..&tdtlun..&tdtuv were terminated by issuing ' -CANCEL DDF THREAD in combination with causing the allied agent TCB to abend

TTS182 - The distributed threads associated with the Logical Unit of Work ID (LUWID) &tdtnetid..&tdtlun..&tdtuv were terminated by cancelling its VTAM sessions, abending its application tasks and issuing the -CANCEL DDF THREAD command

TTS183 - The conversation between the local Database Acces thread and the remote requestor is canceled. However, both threads may still exist on their respective subsystems. The next SQL request issued by the allied distributed thread will fail with one of several possible error conditions. If necessary, you can cancel the distributed allied thread to remove both threads. As a last resort, you can FORCE this Database Access Thread.

TTS184 - There are no active conversations associated with the distributed thread defined by Logical Unit of Work ID (LUWID) &tdtnetid..&tdtlun..&tdtuv. Thus, cancelling the VTAM session on which the conversation was executing serves no purpose.

TTS185 - Thread/STOPPER tried to terminate the VTAM session(s) underlying the distributed DB2 thread defined by LUW ID &tdtnetid..&tdtlun..&tdtuv. This attempt failed for the following reason: &tmvtext

TTS186 - The distributed thread defined by Logical Unit of Work ID (LUWID) &tdtnetid..&tdtlun..&tdtuv still exists. However, the following VTAM commands issued by Thread/STOPPER were successful: &tmvtext

Thread Qualification Messages

TTS190 - There is no active job right now named &wqaljobn

TTS191 - Threads in IDENTIFY or SIGNON status should not be cancelled. Use the FORCE command if it is essential to terminate this thread.

TTS192 - Invalid TDTFLAG7 value. Report this internal error to RAI

TTS193 - Unknown TDTFLAG2 value. Report this internal error to RAI

TTS194 - Unknown TDTFLAG2 value for a thread queued for delete. Please report this internal error to RAI.

TTS195 - Unknown TDTFLAG2 value for a thread being deleted. Please report this internal error to RAI.

TTS196 - Thread in IDENTIFY or SIGNON status (not at the PLAN level) is being deleted on subsystem &tdtdsn as a consequence of cancellation method: "&tcpmeth"

TTS197 - Cancel successful for the following DB2 thread that was in IDENTIFY or SIGNON status: AuthID: &tdtauth Job: &tdtjobnm, Address space: &tdtasid and

```
         TCB address &tdttcb@.  The thread task was abended with system code &tmvac#
         and abend reason code &tmvarc#..  &tmvmta
TTS198 - Unknown TDTFLAG2 value for a thread that was cancelled.  Please report this
         internal error to RAI.
```

DB2 VTAM session management messages

```
TTS201 - Required parameter &p1 was omitted
TTS202 - Error received from TTSVTAM. RC= &p1
TTS203 - DB subsystem &p1 has no active sessions
TTS204 - Specified SID = &p1 is not valid for DB2 subsystem &p2
TTS205 - Session ID &p1 between &p2 and &p3 was terminated successfully
TTS206 - The Password &p2 supplied for VTAM application &p1 is invalid
TTS207 - The VTAM application (APPLID &p1 / Password &p2) used by Thread/STOPPER to
         issue VTAM commands is either not installed, not installed properly or needs
         to be activated
```

Thread/STOPPER Snap Messages

```
TTS210 - An RFASNAP file must be allocated to use the SNAP facility
TTS211 - Thread/STOPPER failed to acquire DCB and TFD for RFASNAP file
TTS212 - SNAP failed to obtain ACE control block through cross memory move
TTS213 - SNAP failed to obtain currently active execution block
TTS214 - SNAP failed to obtain DB2 address space control element
```

Audit Cancellation Success (TTSACS) Messages

```
TTS220 -  Failed to acquire a thread audit entry
TTS221 -  A thread audit jobstream was submitted in the background
TTS222 -  Failed to create an ISPF table for audit cancellation entries

TTS300I -  Thread/STOPPER Cancellation Auditor started on DATE() 'at' TIME()
TTS301E -  A DB2 ssid and Thread argument pair must be supplied
TTS302E -  Failed to define an IFI host command environment.  Halting.
TTS303I -  Thread' p1 'on DB2 subsystem' p2 'was cancelled successfully
TTS304E -  Thread/STOPPER cancellation audit processing halted due to severe error.
           RC = 12
TTS305I -  Thread/STOPPER confirmed that all auditable thread(s) were cancelled
           successfully:
TTS306I -  Thread' p1 'on DB2 subsystem' p2 is currently being deleted by DB2
TTS307I -  Thread' p1 'on DB2 subsystem' p2 is currently queued for deletion by DB2
TTS308I -  Thread/STOPPER audit processing complete on DATE() 'at' TIME()||'. '
           'Completion code =' p1
TTS309E -  Thread/STOPPER unable to load defaults module TTS$TSD.  Terminating.
TTS310E -  Invalid parameter syntax beginning ='p1
TTS311E -  Parameter='kword 'invalid or value is missing
TTS312E -  Invalid / Unknown keyword:' p1
TTS313I -  Thread/STOPPER Audit Options
TTS314I -  Processing will confirm the following thread(s) were cancelled
           successfully:
TTS315I -  Thread/STOPPER will record these confirmation(s) in the THREAD_AUDIT table
```

```
TTS316I -  maintained on the same DB2 subsystem(s) in which the cancelled thread(s)
           existed
TTS317I -  Unable to load Thread/STOPPER system defaults module
TTS$TSD.-  Thread Audit cancellation job must abort.
TTS319E -  Invalid message number:' msgid
```

Tailor Audit Jobstream (TTSTAJ) Messages

```
TTS320E -  Read failed from TTSCNTL file
TTS321E -  A DB2 ssid and Thread argument pair must be supplied
TTS322E -  Failed to define an IFI host command environment. Halting.
TTS324E -  Thread/STOPPER cancellation audit processing halted due to severe error.
           RC = 12
TTS325I -  Batch job' p1 'submitted to audit thread cancellation outcome(s)
TTS329E -  Invalid message number:' msgid
```

Thread/SERIES Messages set by Exit routines

```
TTS400 - ISPF logical screen canceled successfully
TTS401 - ISPF logical screen already terminated
TTS402 - ISPF logical screen scheduled for termination
```

Automated Monitor Messages

```
TTS500 - Failed to attach Monitor process DB2 ssid &tvldsn
TTS501 - File TTSIN (which defines DB2 subsystems to be monitored) is not allocated.
         Execution impossible.
TTS502 - Failed to open file TTSIN
TTS503 - Failed to parse input record from file TTSIN
TTS504 - Failed to acquire TVL control block through #LISTADD
TTS505 - DB2 subsystem &wqaldsn is not currently active in MVS system &zsysid..
         Monitoring of &wqaldsn will be deferred until the subsystem starts up.
TTS506 - Internal error - #LISTNEW for TVL chain failed
TTS507 - Failed to create TTS Report process
TTS508 - No TVLs (DB2 subystems) selected to monitor
TTS509 - Internal error - #LISTNEW for TDS chain failed
```

Monitor Listener Process Messages

```
TTS510 - Failed to START a MONITOR trace on DB2 subsystem &cafdsn
TTS511 - Failed to acquire a global chain for DB2 subsystem &cafdsn
TTS512 - Failed to acquire storage for a list resource name for DB2 subsystem &cafdsn
TTS517 - #AREA failed for the monitor process for DB2 subsystem &cafdsn
TTS518 - DB2 subsystem &cafdsn is not currently active in MVS system &cafmvs
TTS519 - DB2 subsystem &cafdsn is not defined within MVS system &cafmvs
TTS521 - #LISTNEW for an ETD chain failed within DB2 subsystem &TVLDSN
TTS522 - Load module libraries for DB2 subsystem &cafdsn must be allocated to the
         DDname &cafdsn.LOAD
TTS523 - #LISTADD for a new EAP failed within DB2 subsystem &cafdsn
TTS524 - DB2 subsystem &cafdsn failed to acquire storage for an EAP structure map
```

```
TTS525 - #LISTNEW for an EAQ chain failed within DB2 subsystem &TVLDSN
TTS526 - #RVMAP failed for Thread/SENTRY system defaults
TTS527 - Invalid action flag calculation in calcindx routine.
```

Automated Monitor Messages

```
TTS530 - Failed to open the load module library for the DB2 subsystem named &CAFDSN
TTS531 - Load module &CAFLMOD not found in the set of libraries concatenated to file
         &CAFDSN.LOAD for DB2 subsystem &CAFDSN
TTS532 - Failed to load the module named &caflmod from the set of libraries
         concatenated to file &cafdsn.LOAD for DB2 subsystem &cafdsn
TTS533 - Thread/SENTRY failed to create a REXX language processor environment
TTS534 - Thread/SENTRY failed to dynamically create a REXX host command environment
         named TTS
```

Listener Process Messages

```
TTS540 - Thread/SENTRY failed to acquire a non lifo storage area
TTS541 - Thread/SENTRY main prolog failed to #GMAIN a control block
TTS542 - Thread/SENTRY main prolog failed to #GETMAIN a control block
TTS543 - Thread/SENTRY failed to acquire a large IFI return area
TTS544 - #LISTNEW for an ETD chain failed within DB2 subsystem &tvdowner
TTS545 - #LISTNEW for an EAQ chain failed within DB2 subsystem &tvdowner
TTS546 - #LISTNEW for an EIT chain failed within DB2 subsystem &tvdowner
```

Monitor Cancellation Messages

```
TTS550 - Failed to lock the EAC chain for DB2 subsystem &cafdsn
TTS551 - Failed to acquire an EAC for DB2 subsystem &cafdsn
TTS552 - Failed to unlock the EAC chain for DB2 subsystem &cafdsn
TTS553 - Failed to delete an EAC for DB2 subsystem &cafdsn
TTS560 - Thread/SENTRY module TTSJMAR failed to attach the common REPORT process
TTS561 - Thread/SENTRY failed to load its table of system defaults (TTS$TSD)
TTS570 - &&ZMSGID - &&ZDSN thread (Plan: &&EACPLAN Auth ID: &&EACAUTH Jobname:
         &&EACJOBNM )
TTS571 - violated policy &&POLID   It is being cancelled at &&ZTIME on &&ZDATE
TTS572 - with system abend code &&TMVAC# and an abend reason code of &&TMVARC#'
TTS574 - The TSO user failed to receive &&TMVLML# (of 3) lines of abend notification
         text (because noterminal buffers were available).
TTS575 - The DB2 thread associated with
TTS576 - Authorization ID &&TDTAUTH
TTS577 - was deliberately terminated by Thread/SENTRY
TTS578 - Your DB2 application will NOT be abended.
TTS579 - has violated Policy &&POLID and receives this warning.
```

Thread/SENTRY Messages

```
TTS580 - -CANCEL (DDF) THREAD command failed on DB2 ssid &cafdsn
TTS581 - violated the &&POLID policy.  It was cancelled but remains active.
         Monitoring
TTS582 - will continue until the thread terminates.
TTS583 - The DB2 thread associated with Authorization ID &&EACAUTH
TTS584 - violated the &&POLID policy.   Thread cancellation
TTS585 - is still in progress.

TTS590 - DB2 utility thread selected.  Press END key to continue

TTS600 - Failure during Unit of Work limit checking of thread for Plan &&PLANID using
         Policy &&POLID
TTS601E -  Failure during Interval based limit checking of thread for Plan &&PLANID
           using Policy &&POLID
TTS602I -  Thread for Plan &&PLANID has violated limits imposed through Policy
           &&POLID
TTS603W -  TSO user &&TSOID is on cancellation notification list but is not logged
           on.
TTS604I -  Thread for Plan &&PLANID has violated limits imposed through Policy
           &&POLID but will continue.
TTS605E -  Deletion of ETD failed during action processing for policy &&POLID
TTS606I -  Thread for Plan &&PLANID within job &&JOBNAME scheduled for action --
           policy &&POLID - AC = &&ACCDE
TTS607I -  Thread action was taken due to an &&ACSTRING limit violation within policy
           &&POLID
TTS608W -  IFCID record trace has been disabled due to processing failure.
TTS609E -  Unable to establish required level of APF authorization.
TTS610E -  Unable to reset APF authority.
TTS611E -  Error sensing current APF authoriation level.
TTS612E -  Invalid CICS jobname encountered during TTSNCICS notification processing.
TTS613E -  Unsuccessful attempt to notify a CICS based Thread/SENTRY administrator.
TTS614E -  Invalid CICS terminal encountered during TTSNCICS notification processing.
TTS615I -  Thread violated policy &&POLID but will continue due to WARN setting.
TTS616I -  An attempt to notify terminal &&LTERM on IMS system &&IMSID has failed.
TTS617W -  CICS job &&CICSID is on the notification list but is not active.
TTS618E -  #LOCCTRM error attempting to locate CICS Teminal Name &&LUNAME
TTS619W -  IMS ID &&IMSID is on the notification but is not active.
TTS620W -  VTAM Terminal &&LUNAME may not be logged on
TTS621I -  Thread agent was a batch TSO address space.  No notify was issued.
TTS622E -  Error during TSO address space analysis
TTS623E -  TSO NOTIFY processing has been bypassed
TTS624I -  Notifies disabled for Plan &&PLANID / Jobname &&JOBNAME / Policy &&POLID
           Reason &&ACSTRING
TTS625W -  ASID for TSO user &&TSOID not located. Notify will be bypassed.
TTS630I -  Job &&JOBNAME being deliberately swapped out due to shortage of active log
           space

TTS700 - TTSJMC - Thread/SENTRY is starting
TTS701 - Thread/SENTRY start normal completion
TTS702 - Thread/SENTRY is stopping
TTS703 - Thread/SENTRY stop complete
TTS704 - Thread/SENTRY operating in &&TTSMODE mode
```

```
TTS705 - Thread/SENTRY job &&TTSJOB waiting for work
TTS706 - removing completed TTSL process for DB2 subsystem &&CAFDSN
TTS707 - all monitor processes completed
TTS708 - &&JOBNAME waiting for work
TTS709 - &&JOBNAME waiting for work
TTS710 - MVS START command received
TTS711 - MVS STOP command received
TTS712 - MVS START command received

TTS711 - null LU name supplied for CICS DB2 thread
TTS712 - VTAM LUname for CICS DB2 thread not found
TTS713 - multiple instances of same Transaction ID
TTS714 - target ASID not a CICS address space
TTS715 - target ASID not active
TTS716 - Processing error during DB2 / CICS Task identification
TTS717 - CICS / DB2 thread canceled via CICS SET TASK(taskid) PURGE.  Thread no
         longer has a plan associated with it.  It is in SIGNON status.
```

Listener Process Messages

```
TTS750 - Listener process for DB2 subsystem &&CAFDSN is starting
TTS751 - Listener process start completed for DB2 subsystem &&CAFDSN
TTS752 - Listener process for DB2 subsystem &&CAFDSN is stopping
TTS756 - TTSJCMD received the following command
TTS757 - TTSJCMD STOP command received
TTS758 - TTSJCMD ABEND command received
TTS759 - TTSJCMD FORCE command received
TTS760 - File &&PVTNAME not available to receive TRACE_IFCID output
TTS761 - Thread/SENTRY deliberately abending
TTS762 - Issuing user Abend 2001 reason code 00DB2000
TTS763 - Thread/SENTRY jobstep will abend
TTS764 - TTSJCMD received no command text within MVS START or MODIFY.
TTS770I - TTSJRR RULE_REFRESH process starting
TTS771E - Purging of &&MSGVAR LAB failed during RULE_REFRESH processing
TTS772I - TTSJRR completed RULE_REFRESH process
TTS773E - Thread/SENTRY Compiler failed to insert new rules
TTS774I - TTSJRR inserted new Thread/SENTRY rules
TTS775I - TTSJRR removed current Thread/SENTRY rules
TTS776W - User &&TSOID had no TSO logon buffer available.  Notify via MVS SEND.
TTS777W - TPUT for notify of user &&TSOID failed.  Notify suppressed.
TTS800 -  Thread/SENTRY BETA 1 does not support DB2 utilities at this time
TTS801 -  Escalated cancel not supported at this time - PLAN &&PLANID, Unique Value
          &&THRUV is still active
TTS802I - thread for Plan &&PLANID has violated limits defined by Policy &&POLID
TTS803I - Thread for Plan &&PLANID within job &&JOBNAME scheduled for action - policy
          &&POLID - AC = &&ACCDE
TTS804I - Thread action taken due to &&ACSTRING limit violation defined by policy
          &&POLID '
TTS805I - Thread for Plan &&PLANID has violated limits defined by multiple policies.
TTS806I - Invalid QWHC correlation header - cannot determine PLAN name.
TTS807E - Invalid pseudo QWXX structures - cannot map EIT.
```

Batch Facility and Console Facility Messages

```
TTSE001 - Invalid parameter keyword
TTSE002 - Output lines limit of  exceeded.  Use the LIMIT(nnnn) parameter to increase
          the limit or restrict the qualifications of your request.
TTSE003 - SSID(...) is a required parameter
TTSE004 - The DSN command allows only the SSID(...) parameter
TTSE005 - Failed to connect to DB2 subsystem= Plan= RC= ALIRC= Reason=
TTSE006 - Error on READS request: RC= IFCARC1= IFCARC2=
TTSE007 - Invalid command= -- Commands are: DB2S, DISPLAY, CANCEL, STATUS, LOCK,
          ELOCK and STOP
TTSE008 - Unable to define REXX IFI Host command environment -- RC=
TTSE009 - The requested DB2 subsystem  is not defined
TTSE010 - The requested DB2 subsystem  is not active
TTSE011 - Specified thread: SSID= ACE= is not active
TTSE012 - Parameter keyword  is not allowed on the CANCEL command
TTSE013 - You cannot cancel the Thread/STOPPER thread and plan
TTSE014 - Thread: SSID= ACE= is in the process of termination
TTSE015 - Thread: SSID= ACE= is scheduled to terminate
TTSE016 - Unable to load Thread/STOPPER parameters.  RC=
TTSE017 - Command:  -- IFCARC1= IFCARC2=
TTSE018 - The CANCEL command requires both SSID and ACE parameters
TTSE019 - The Syntax of the DSN command is: DSN SSID(ssn1,ssn2,...) <db2 cmd>
TTSE020 - The CANCEL command failed since the thread is not at the plan level.
          The FORCE command modifier must be specified
TTSE022 - Thread: SSID= ACE= was canceled via  command, RC=, Reason=.
TTSE023 - Specified thread: SSID= ACE= already was FORCED
TTSE024 - The LIMIT parameter must have a numeric value.  9999 is assumed
TTSE025 - Parameter  can only have YES or NO values.
TTSE026 - The ORIGIN parameter supports only WTOR, QEDIT or TSO values
TTSE027 - Parameter  allows no value.  Found
TTSE028 - The thread with plan name DSNUTIL was not shown on -DIS UTIL(*)
TTSE029 - Failed to insert a row in the Thread_Audit table, SQLCODE=
TTSE030 - Multiple Threads were selected for CANCEL processing but
          CANCEL_MULTIPLE(NO) is either specified or defaulted
TTSE031 - More than one utility job is presently active. Issue the -TERM UTIL command
          manually.
TTSE032 - No threads were found that satisfy your qualification criteria
TTSE033 - Failed to read input parameters from file TTSIN, RC=
TTSE034 - Failed to read JCL  from the TTSCNTL DD name, RC=
TTSE035 - Failed to submit JCL  to the internal reader TTSRDR, RC=
TTSE036 - Selected  threads.  You must uniquely identify a single thread (through its
          ACE address).
TTSE037 - ROUTECDE and DESCRCDE parameter values must be within the range 1-16.  You
          specified a value of
TTSE038 - Specify a list of codes delimited by commas.  E.g. 1,2,...
TTSE039 - Invalid JOBNAME parameter  - job not found
TTSE040 - CANCEL request for Thread SSID='ssid ACE='ace',PLAN='plan ',AUTH='auth will
          be terminated asynchronously
TTSE041 - Thread/STOPPER Audit Facility job submitted
TTSE042 - Thread/STOPPER is waiting for work
TTSE043 - The VTAM LUNAME was not found in the target CICS
TTSE044 - Target Address Space is not CICS
TTSE045 - Target Address Space is not active
```

TTSE046 - Functional error in CICS cancel interface
TTSE047 - Invalid character found in string
TTSE048 - Instead of LUWID parameter, use any or combination of NETID(), LUNAME(), UNIQ() and CCNT()
TTSE049 - Commit count (CCNT) must be a decimal number, value ignored
TTSE050 - Error on -START TRACE(MON) command: RC= IFCARC1= IFCARC2=
TTSE051 - Thread/STOPPER: Version : .
TTSE052 - The CANCEL LOCKS command requires a DBNAME, TBNAME or TSNAME qualifier
TTSE053 - The Thread/STOPPER task is not authorized for profile - Contact the Thread/SERIES security administrator
TTSE054 - The TBNAME and TSNAME qualifiers are mutually exclusive. Specify either TBNAME or TSNAME
TTSE055 - The DBNAME, TBNAME and TSNAME qualifiers require the LOCKS operand
TTSE056 - TTSIN DD name must be specified and must contain valid TTSB commands
TTSE057 - A NULL LUNAME was passed to the CICS cancel interface

Appendix Z

Problem Determination

Additional Information Checklist

Although every effort has been made to develop defect-free software, you may still encounter problems. You can assist in the diagnosis and correction of problems by furnishing us with the proper documentation. The following list enumerates the information we typically require. Your Relational Architects support representative will assist you in collecting these item(s).

1. A brief description of the application and operating environment. For example:

   ```
   "The application is a REXX dialog using RLX/REXX to
   access a DB2 table.  It executes within ISPF and is
   invoked from PDF option 6.  The EXEC usually runs
   successfully but sporadically abends with a System 0C4.
   RLX/REXX displays a panel which shows the abend occurred
   within module PMVRVA at offset X'150'.  Listings for the
   RFASNAP and RFATRACE datasets were obtained.
   ```

2. A listing of the application in which the problem occurred.

3. JCL of the job or TSO session.

4. If running under TSO, print the output from the TSO command: `"LISTALOC ST H"` with a brief explanation of the allocated user libraries.

5. Refer to the appropriate appendices in your product's Installation and Customization Guide for guidance in allocating the RFASNAP, RFATRACE, and SYSUDUMP data sets.

6. When running in an ISPF environment and abnormal termination occurs, the "Abnormal Termination Diagnostic Information" panel is displayed. Print this panel. Also, change the selection in the "Produce a dump?" field to "Y". Specify "N" in the "Attempt to retry?" field and type "DUMP" in the panel's COMMAND field. Lastly, press the ENTER key.

   Several dumps will be obtained. This may take some time so please be patient. After the dumps are written, the MVS message "IEA995I SYMPTOM DUMP OUTPUT" will appear on the screen signifying abnormal termination. Please print this screen and save the trace, snap and dump dataset(s) obtained thus far.

7. In extreme cases, RAI may also request a GTF trace, SLIP TRAP or system dump.

Reader Comment Form

Manual : **DB2 Thread Control Series -- Installation and Customization Guide**
Edition: Version 7.1 -- May 2014

Please assist us in improving this manual and in developing future enhancements to this product. We would appreciate your comments about the content and organization of this manual and the product it documents. Please refer to specific page numbers when applicable.

Please forward your comments to us on this Reader's Comment Form with the understanding that all comments become the property of Relational Architects Intl, Inc. We may use or distribute whatever information you supply in any way we believe appropriate without incurring any obligation to you.

Outside the United States, your local Relational Architects product representative will be glad to forward this form on to us.

Name ______________________ Company ______________________

Title ______________________ Address ______________________

Department ______________________ City ______________________

Telephone ______________________ State __________ Zip __________

- -

Comments: