Adabas

Triggers and Stored Procedures

Version 8.5.4

October 2025

ADABAS & NATURAL

This document applies to Adabas Version 8.5.4 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ADAMF-TRIGGERS-854-20251029

Table of Contents

Triggers and Stored Procedures

Organization

This documentation provides all the information necessary to install and use the Adabas facility for implementing and maintaining triggers and stored procedures.

The documentation is organized in the following parts:

Introduction	Introduces procedures and the distinctions that are relevant for programming and processing them. It tells you how procedures are processed.
Installation and Configuration	Provides installation and configuration information.
Processing and Performance	Explains the run-time processing of procedures in detail, providing performance information throughout the processing sequence.
Programming and Performance	Discusses issues related to writing procedures, including performance issues.
Calling Stored Procedures	Tells you how to use the PC command in conjunction with the stored procedure link routine STPLNKnn to invoke a stored procedure.
Trigger Maintenance	Tells you how to use the online Trigger Maintenance facility that is accessible from the main menu of Adabas Online System (AOS). Includes both user and administrator information.
TRGMAIN	Presents the TRGMAIN API for maintaining triggers from a user program.
TRGUNLD and TRGLOAD Utilities	Presents the TRGUNLD and TRGLOAD utilities for unloading trigger definitions to a work file and subsequently loading them into the trigger file.
Programming Examples	Provides sample, annotated program listings.

Messages and Codes

For information about interpreting messages and codes related to triggers and stored procedures and about resolving problems that they identify, see the *Adabas Messages and Codes* documentation. SYSTRG message explanations are available using the Natural SYSERR utility.

1 Conventions

Data set names starting with DD are referred to throughout the Adabas documentation with a slash separating the DD from the remainder of the data set name to accommodate z/VSE data set names that do not contain the DD prefix. The slash is not part of the data set name.

Notation *vrs*, *vr*, or *v*: When used in this documentation, the notation *vrs* or *vr* stands for the relevant version of a product. For further information on product versions, see *version* in the *Glossary*.

2 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://techcommunity.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Introduction

3 Introduction

The Adabas Triggers and Stored Procedures Facility is used to define, process, and monitor triggers and stored procedures.

This chapter introduces both types of procedures and their characteristics. It introduces the components of the facility and describes how they function together to provide online services and background processing.

Procedures

A *procedure* is a Natural subprogram that is written and tested using standard Natural facilities. The primary difference between triggers and stored procedures is the way they execute:

- A *trigger* executes ("fires") automatically when a specified event occurs-usually a data access or update to the associated table. The event occurs when selection criteria are satisfied. Selection criteria include file number and possibly command type or the name of a field located in the format buffer, or both.
- A *stored procedure* executes when the database management system (that is, Adabas) receives a special user call.

The same parameters are passed to the subprogram whether it is a trigger or a stored procedure.

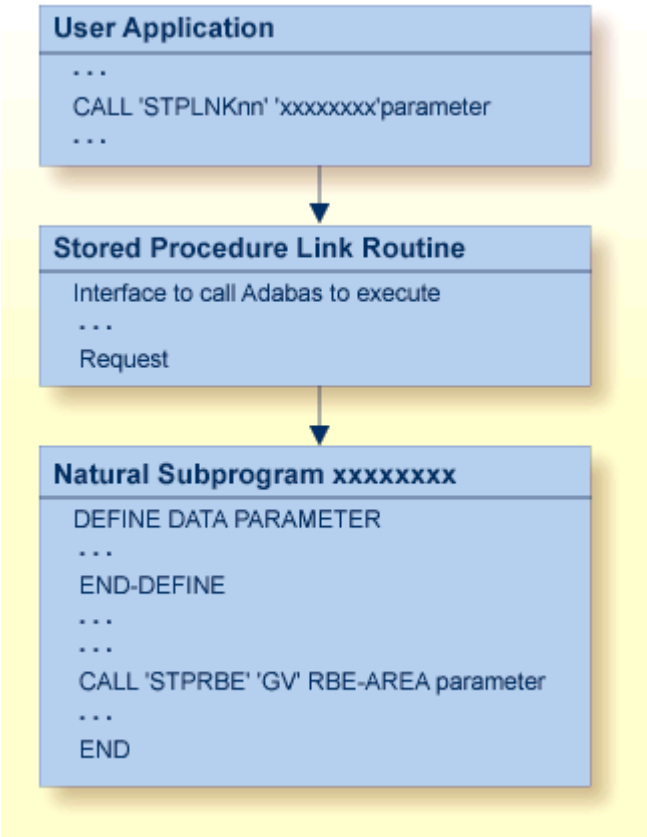
Stored Procedures

A stored procedure is invoked from a user application, which issues a special call to the procedure and may pass it one or more parameters. The procedure is executed by Adabas.

Because the procedure is stored in the Natural system file (an Adabas file) loaded on the database (server), it reduces the amount of data traffic to and from the server.

A sample stored procedure subprogram is provided; the call structure that is predefined in the stored procedure front-end may be modified.

The following figure illustrates stored procedure usage. STPLNK is the stored procedure link routine used to invoke a stored procedure request; STPRBE is the record buffer extraction routine, which is called by the procedure and used to retrieve the parameters passed by the calling routine.



Stored Procedure Usage

For more information about STPLNK, see the section *Stored Procedure Link Routine (STPLNKnn)*. For programming information about STPRBE, see the section *Record Buffer Extraction Routine (STPRBE)*.

Triggers

A trigger has two parts: the triggering event and the triggered procedure.

The triggering event is defined by a set of selection criteria such as an Adabas file number and optionally a command or a field name (or both). When the criteria are met, the event occurs and the triggered procedure is executed in response. For example, a trigger might be fired by an update command to the SALARY field in the EMPLOYEES file.

Note: A trigger cannot be fired by another trigger.

Triggers can be defined to execute before or after the initiating Adabas command is executed by the Adabas nucleus. The behavior of a trigger depends to some extent on whether the trigger and the Adabas command are synchronized and, if so, whether the trigger participates in the transaction logic of the command.

Triggers have three major characteristics:

- pre-command or post-command execution.

  A trigger can execute before or after the initiating Adabas command.
- asynchronous or synchronous execution.

  A trigger can execute independently of Adabas command processing or require suspension of Adabas command processing (that is, command processing must wait for the results of the triggered procedure).
- participation or non-participation.

  A synchronous trigger may participate or not participate in the user's transaction (that is, ET) logic.

Pre-command or Post-command

A *pre-command* (or "pre-") trigger is executed before the initiating Adabas command is processed by the Adabas nucleus. Before an Adabas command is executed, a check is made to verify whether a trigger should be fired.

For example, if a trigger is defined to fire every time an `N1` command is issued against a specified file, the `N1` is the initiating Adabas command. Before the `N1` is executed, a check determines that a pre-command trigger is to be fired. The `N1` is processed after the triggered procedure is successfully executed.

A *post-command* (or "post-") trigger executes after the initiating Adabas command is processed by the Adabas nucleus. The triggered procedure executes only if the return code for the command from the Adabas nucleus is zero. If the return code is nonzero, no checking for triggers is done, and processing continues in the normal way. For a successfully executed command, the command is checked for any triggers before processing of the command completes.

For example, if a trigger is defined to fire every time an `L3` command is issued against a specified file, the `L3` is the initiating Adabas command. After a zero return code is received, a check determines that a post-command trigger is to be fired. The triggered procedure is executed after the command is successfully executed and before the user is notified.

Asynchronous or Synchronous

An *asynchronous* trigger executes independently of the initiating Adabas command. Adabas command processing for the user continues uninterrupted while the triggered procedure executes as a separate process. The triggered procedure may execute after the user has already received the response from the initiating command.

When a command is issued that fulfills trigger criteria, the trigger is fired and processing of the Adabas command resumes. The Adabas command and the triggered procedure do not affect each other directly.

Asynchronous triggers are used when there is no interdependency between the procedure and the actual Adabas command.

A *synchronous* trigger requires a suspension of Adabas command processing for the user. The initiating Adabas command is suspended until the triggered procedure completes execution. It is possible that the results of the procedure will affect the Adabas command.

For a post-command trigger, the Adabas command executes before the triggered procedure and then is suspended; the results of the command are not returned to the user until the triggered procedure completes execution.

Participating or Non-participating

Synchronous triggers may participate or not in the logic of the initiating Adabas command.

A *participating* trigger results in the execution of a procedure that is assigned the same user (communication) ID as the Adabas command that caused the participating trigger to be fired; thus, it can participate fully in the logic of the transaction.

- An ET (end transaction) or BT (backout transaction) issued by the initiating command's process affects any transactions in flight from the trigger.
- An ET or BT issued by the triggered procedure affects any transactions in flight for the initiating command.

A *non-participating* trigger has a user (communication) ID that is different from that of the Adabas user queue element (UQE) that identifies the initiating command; thus, it does not participate in the initiating command's transaction logic.

- An ET or BT issued by the initiating command's process does not affect the triggered procedure.
- An ET or BT issued by the triggered procedure does not affect the initiating command's process.

Components

Adabas uses three major components to implement and process triggers and stored procedures:

- the Adabas trigger driver is part of the Adabas nucleus and has overall control of triggers and stored procedures. It detects procedure requests and initializes the Natural trigger driver to execute them.
- the Natural trigger driver runs as a batch Natural nucleus that actually executes both triggers and stored procedures. Operating in conjunction with the Adabas trigger driver, it handles the interprocess communications between the Adabas nucleus and the Natural subsystem that executes the procedure.

- Trigger Maintenance is a Natural application accessible from Adabas Online System (AOS). A system of menus allows you to create and maintain trigger definitions, define the triggers profile, and monitor and to some extent control trigger activity within the nucleus.

Adabas Trigger Driver

The Adabas trigger driver is executed as a part of the Adabas nucleus and generally controls the whole run-time processing of a trigger. It determines whether a trigger is to be fired, initiates the Natural trigger driver, and interacts with it to ensure the correct and timely processing of the procedures. For more detailed information about how procedures are processed, see section *Processing and Performance*.

Initialization

When the Adabas nucleus starts, it determines whether the ADARUN parameter `SPT=YES` has been specified; if so, the nucleus passes control to the Adabas trigger driver to allow it to initialize. During initialization, the Adabas trigger driver

- acquires storage for buffers;
- verifies that a valid trigger file is loaded;
- verifies the triggers profile on the trigger file and extracts the session parameters to be used in processing triggers and stored procedures for the session;
- verifies that the trigger file contains at least one trigger definition (a requirement for Adabas trigger driver initialization); and
- reads the trigger definitions and adds entries to the trigger table, which occupies a buffer in memory. In a nucleus cluster environment, the trigger table is obtained from an active nucleus. During procedure processing, the trigger table can then be checked for the existence of a trigger, instead of the more expensive alternative of reading the trigger file.

Starting Natural Subsystems

After the Adabas trigger driver is initialized, it starts the Natural subsystems that are responsible for the actual execution of the procedures.

The Natural subsystems execute user-written procedures. The maximum subsystems parameter in the Adabas triggers profile determines the number of Natural subsystems (1-10) to be started.

Each Natural subsystem is typically a minimally modified batch Natural nucleus that runs as a subtask in the Adabas address space. This affects the region size specified on the MPM startup JCL/JCS. The effect may be minimized by using a split Natural nucleus.

When a Natural subsystem is started, the Adabas trigger driver keeps track of any change in subsystem status or activity. The user can monitor these activities by using the Subsystem Activity function of the Triggers Maintenance facility.

When a Natural subsystem becomes active:

- the Natural trigger driver acquires control; and
- the Adabas trigger driver is informed that the subsystem is ready to start processing any procedures that may result from a stored procedure request or the firing of a trigger.

Checking for Triggers

For each command that the nucleus receives, the Adabas trigger driver determines whether a trigger needs to be fired.

For pre-command triggers, the Adabas trigger driver checks for triggers before the command is selected for processing by the Adabas thread. Once a command has been processed successfully by Adabas and the response code is zero, the Adabas trigger driver determines whether there are any post-command triggers to be fired. Only two triggers (one pre- and one post-command trigger) can be fired for any one command, regardless of the results.

If a command results in a trigger being fired, or if the Adabas trigger driver determines that the command is a stored procedure request, an entry is created in the

- pre-trigger queue if the command has not been executed; or the
- post-trigger queue if the command has been executed successfully.

The entry contains information obtained from both the command and the corresponding entry in the trigger table.

If a Natural subsystem is waiting for work, it is given the trigger request immediately. Otherwise, the trigger request remains in the pre- or post-trigger queue until the next subsystem is available. When a subsystem accepts a trigger request, processing continues under the control of the Natural trigger driver (see *Components*).

Processing Procedure Results

When the procedure completes execution, the Natural trigger driver places the results in the "trigger request entry" and the status is updated appropriately. When the Adabas trigger driver detects this, it takes responsibility for completing the trigger processing.

For both pre- and post-command triggers, the return code from the procedure determines how the results are processed. For more information, see the section *Processing the Results*.

Shutdown

The Adabas trigger driver keeps track of all Natural subsystems that fail. If all subsystems fail, it determines that no further processing of procedures is possible and terminates according to the error action value in the Adabas triggers profile. The error action Ignore or Reject in a nucleus cluster environment is passed to all other nuclei in the cluster.

The Adabas trigger driver also terminates if the Adabas nucleus receives the `ADAEND` or `HALT` operator command and instructs the Adabas trigger driver to shut down as well.

Natural Trigger Driver

The Natural trigger driver is initialized during the startup of the Natural subsystems. It is responsible for executing all triggered and stored procedures, and includes the following components:

SPAENA	(BS2000 only) retrieves the address of the database command queue.
STP	is invoked when a Natural subsystem is started. STP initializes the global data area STPGDA and establishes any necessary settings for the Natural session. Its primary function is to handle recovery from any errors and ensure that a restart is done.
STPEND	performs any back-end processing for the NATURAL Trigger Driver. Back-end processing requires this routine to determine if an error has occurred during subtask processing. If so, then an error message should be placed in the message error of the subtask table in the main task.
STPPDRIV	functions as the main routine for the Natural trigger driver, and is responsible for invoking the procedure. STPPDRIV calls STPNAT.
STPNAT	is responsible for any communication with the Adabas trigger driver and for servicing the trigger requests for the subsystem. It notifies the Adabas trigger driver that it is ready for work.
STPUES	performs field data conversion for UES databases.

Setting Up the Parameter List

When a trigger request is accepted by a Natural subsystem, STPPDRIV sets up the parameter list to be passed to the procedure, depending on which parameter list was specified by the trigger that was fired.

Invoking the Tracking Routine STPUTRAK

If the log trigger activity setting in the Adabas triggers profile is active, the routine STPUTRAK is invoked.

STPUTRAK is a user-defined routine that tracks every request to invoke a procedure, both before and after the procedure is invoked. You can use this routine to write trace messages or audit trigger processing for each subsystem. A default STPUTRAK routine is supplied.

Parameters that are similar to those of the procedure and contain information about the trigger are passed to the STPUTRAK routine, in addition to a 250-byte area that can be used as a work area. STPUTRAK uses the work area to retain information for the entire session; the work area is never changed by the Natural trigger driver. STPUTRAK contains an option that allows you to pass this work area to the procedure.

Processing the Procedure

After STPUTRAK has been processed and control has returned to STPPDRIV, the triggered procedure is invoked with a CALLNAT.

When the procedure completes processing, STPPDRIV again checks whether the "log trigger activity" option in the Adabas triggers profile is set to active before informing STPNAT of the results. If log activity is active, STPUTRAK is invoked so that the results of the procedure can be audited.

Recovering from Errors

The procedure does not terminate normally if it incurs an error such as NAT0954, NAT3009, or NAT1305. Instead, the Natural trigger driver performs error recovery.

Regardless of the log trigger activity setting, information about the error is conveyed to STPUTRAK. This information can be used by the database administrator (DBA) for debugging or problem analysis. Therefore, STPUTRAK should always be available for execution by the Natural trigger driver. Software AG recommends activating the trace to obtain this information.

Updating the Trigger Request Entry

After the results of a procedure are delivered to STPNAT, the trigger request is updated and its status is changed to "completed". When the Adabas trigger driver detects the updated entry, it takes responsibility for completing the trigger processing.

STPNAT waits for another trigger request to be made, and the entire cycle starts again.

Trigger Maintenance

Trigger Maintenance, which requires the full version of Adabas Online System, is an interactive facility for creating trigger definitions and monitoring system processing. For detailed information, see the section *Trigger Maintenance* in this documentation.

A *trigger definition* comprises the name and attributes of the procedure to be executed, and the selection criteria that compose the *triggering event* (Adabas command type, file name or number, and field name).

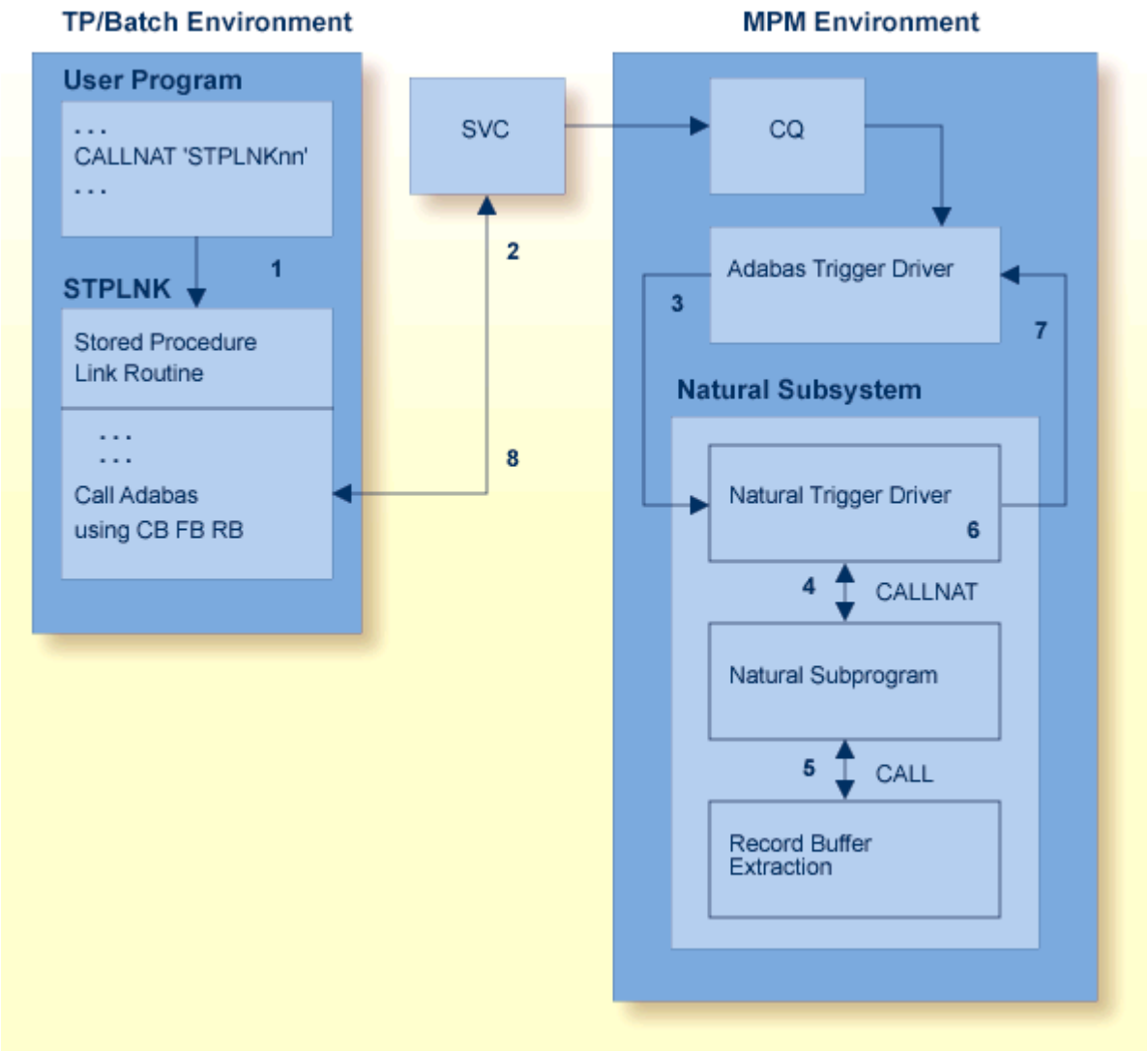
Trigger maintenance also provides functions for monitoring and controlling the execution of any trigger processing by the nucleus. You can examine the activities of both the Adabas and Natural trigger drivers and

- determine the state of the triggers environment in the nucleus; that is, determine which triggers are active and which are not, buffer sizes, and the status of the Natural subsystems;
- modify the settings of various parameters such as activity timeout, log trigger activity, and error action during an active session;
- examine the pre- and post-trigger queues, which contain the procedures waiting to be executed for the pre- and post-command triggers that have been fired;
- examine the activity of the Adabas triggers and stored procedures facility to determine what is executing, whether problems exist, the number of Natural subsystems that are currently active, and so on;
- refresh the trigger table in the Adabas nucleus with new, deleted, or updated trigger definitions; and
- individually activate or deactivate a trigger. If a problem is found with a particular trigger, it can be temporarily or permanently deactivated while the problem is resolved.

Processing Summary

Stored Procedure Processing

The steps involved in stored procedure processing are illustrated in the following figure and described in *Processing Steps*.



Stored Procedure Processing

Processing Steps

1 The user application sets up the required parameters and issues a CALL to the stored procedure link routine STPLNK*nn*.

2 STPLNK*nn* interprets the user's request and issues the call to the DBMS in the form of a standard Adabas API (direct call).

3 The Adabas trigger driver receives the stored procedure request and passes it on to the Natural trigger driver.

4 The specified Natural subprogram (the stored procedure) is invoked.

5 The caller's parameters are made available to the subprogram by the record buffer extraction routine, as required. The parameters can be modified if the option settings permit it.

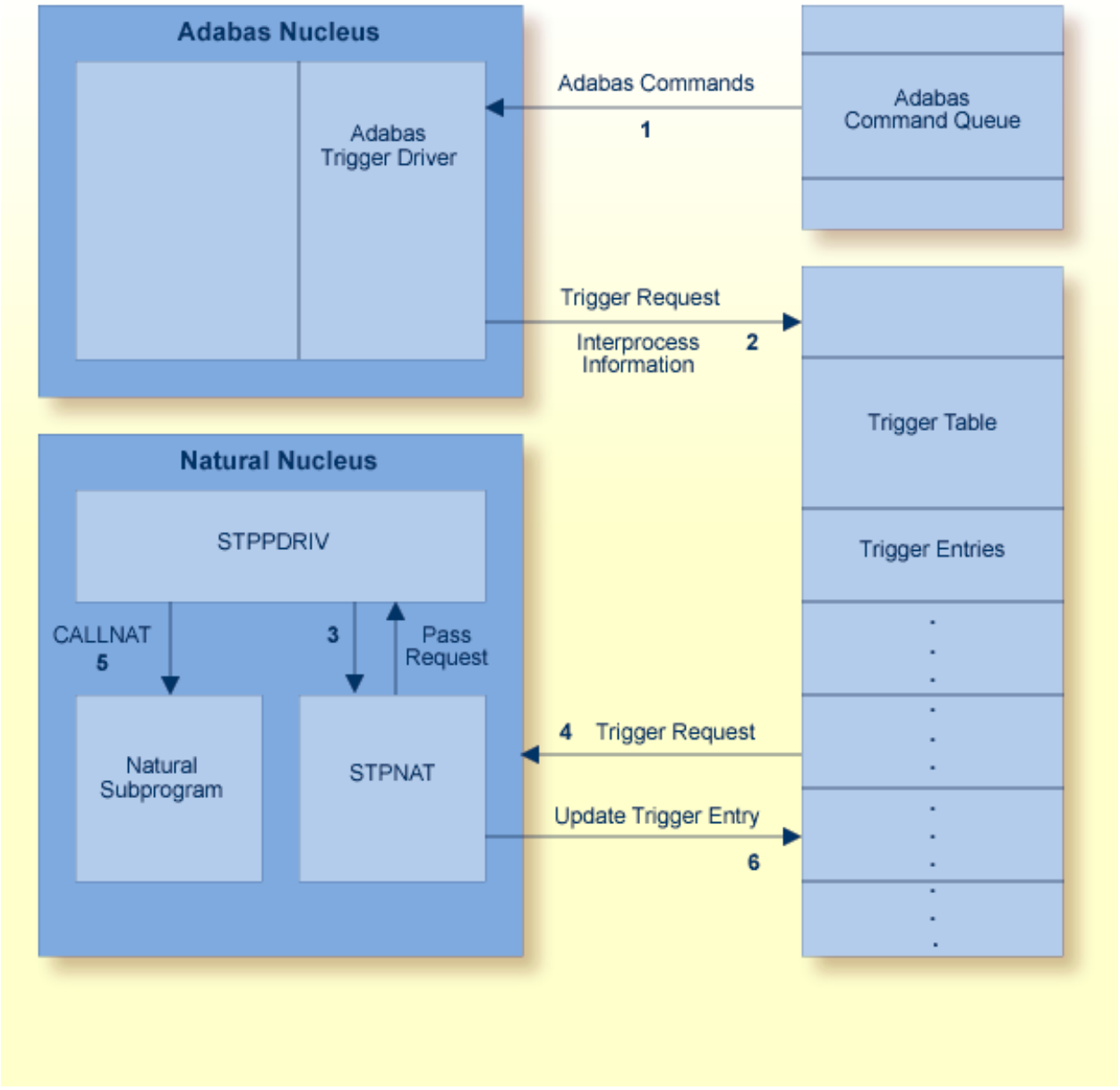
6 On completion, the Natural subprogram returns control to the Natural trigger driver.

7 The Natural trigger driver returns the results of the stored procedure request to the Adabas trigger driver.

8 The user is notified and any modified parameters are returned.

Trigger Processing

The steps involved in trigger processing are illustrated by the following figure and described in *Processing Steps*.



Trigger Processing

Processing Steps

1 The Adabas trigger driver receives all Adabas commands and inspects them for trigger events.

2 If a command meets trigger criteria, the Adabas trigger driver places the trigger request and interprocess information in the trigger table.

3 When the Natural trigger driver control routine STPPDRIV is ready to process another trigger request, it calls STPNAT.

4 STPNAT gets the next trigger request from the trigger table and passes it to STPPDRIV.

5 STPPDRIV determines the trigger request type and issues a CALLNAT to the Natural subprogram named in the trigger definition, passing the relevant parameters.

6 STPNAT updates the trigger entry in the trigger table to indicate when the trigger is completed.

II Installation and Configuration

4 Installation and Configuration

This chapter tells you how to install the Adabas triggers and stored procedures facility. It includes information about setting NATPARMs, assigning printers and work files, accommodating a Natural Security environment, and setting up the stored procedure link routine.

Software Requirements

The Adabas triggers and stored procedures facility delivered with Adabas is available with any supported version of Natural.

The facility requires Adabas Online System (AOS).

Natural Optimizer Compiler is not required, but can significantly improve performance when using triggers and stored procedures.

Overview

The Adabas triggers and stored procedures facility is installed in three parts:

- ***Install Trigger Maintenance***, the online user interface.
- ***Install the Adabas trigger driver***, the Adabas nucleus component.
- ***Install the Natural trigger driver***, the Natural nucleus component.

Install Trigger Maintenance

1. Install AOS

Install the Adabas Online System (AOS) add-on product. Use the instructions provided in the *Adabas Online System* documentation.

2. Load the trigger file

Use the ADALOD utility to load the trigger file to the Adabas database.

Specify the keyword TRIGGER in the ADALOD parameters to indicate that the trigger file is to be loaded. (Do this the same way you would specify the `CHECKPOINT` or `SECURITY` parameter when loading the checkpoint or security file, respectively.)

Omit the ADACMP step, which is not required because the FDT definitions are already known to the nucleus.

The trigger file is currently not permitted to have Adabas security; that is, no Adabas password and cipher or security-by-value.

3. Modify the NATPARMs and relink

Update the NATPARMs to use triggers and stored procedures in the online Natural nucleus and relink them to the Natural nucleus. Use either:

- a logical file definition with the NTLFILE macro specification in the NATPARM; or
- the `LFILE` parameter as a dynamic parameter at session initialization.

The logical file number for the trigger file is 154.

Note: If you do not use either the NTLFILE macro or the `LFILE` parameter, you will be required to enter the database ID and file number every time you start the Adabas triggers and stored procedures facility.

4. Start the facility and create the profile

Start the Adabas triggers and stored procedures facility from a Natural session and create the profile.

1. Log on to Adabas Online System and select the option "Trigger Maintenance". Press ENTER.
2. From the main menu, select "A" for Administrator Functions and press ENTER. If required, enter the database ID and file number, and press ENTER twice.

   Note: The database ID and file number are required only if you did not use either the NTLFILE macro or the `LFILE` parameter in step 3 above; or the value specified in the definition is incorrect.

3. From the Administrator Functions Menu, select "M" for Modify Profile Information and press ENTER.

   The Modify Profile Information screen contains a number of default values that can be overwritten now or later. However, the profile must be correctly installed *before* running the Adabas triggers and stored procedures facility. For more information, see section ***Display/ Modify Profile Information***.

   Note: Trigger Maintenance is not fully operational until the required file-field tables and trigger definitions are added. For more information, see section ***Trigger Maintenance***.

Install the Adabas Trigger Driver

The modules ADATSP and STPEND are located in the Adabas load library. TRGMPMJ is the JCL used to start the Adabas nucleus.

1. Set the ADARUN SPT parameter

Set ADARUN `SPT=YES` to activate the Adabas triggers and stored procedures facility in the nucleus, or `SPT=NO` to deactivate it.

Notes:

1. `SPT=YES` requires `PROG=ADANUC` and `MODE=MULTI`. `SPT` is a global parameter and must be set the same on all nuclei in a cluster.
2. Do not specify `OPENRQ=YES`. It causes problems with Natural subsystems; i.e., they receive response code 148 (ADARSP148) and are not able to initialize.

2. Set up each Natural subsystem

Specify labels and job control assignments for each subsystem.

These depend on the "maximum subsystems" value and on the CMPRINT assignments in the Adabas triggers profile, as well as the printer and work files used by the procedures.

3. Set up all Natural work files and print files

Specify additional labels for all specified Natural work files and print files.

Example for Steps 2 and 3

The following example is provided to illustrate steps 2 and 3 above. In this example:

- the dynamic CMPRINT assignment (an option to be specified in the Adabas triggers profile) is set to TSPRT, and
- the maximum number of subsystems defined in this example is 5.

The following labels must be specified for the JCL:

```
//TSPRT01  DD  SYSOUT=X
//TSPRT02  DD  SYSOUT=X
//TSPRT03  DD  SYSOUT=X
//TSPRT04  DD  SYSOUT=X
//TSPRT05  DD  SYSOUT=X
```

If each of the five subsystems defined in the profile can WRITE, PRINT, or DISPLAY to a print file, the following definitions must be provided in the MPM JCL

```
//CMPRT01  DD  SYSOUT=X
//CMPRT02  DD  SYSOUT=X
//CMPRT03  DD  SYSOUT=X
//CMPRT04  DD  SYSOUT=X
//CMPRT05  DD  SYSOUT=X
```

See the section ***Printer Considerations*** for information about CMPRTnn labels and assigning logical printers.

If each of the five subsystems defined in the profile can WRITE WORK or READ WORK to a work file, the following definitions must be provided in the MPM JCL

```
//CMWKF01 DD DISP=SHR,DSN=WORK.CMWKF01
//CMWKF02 DD DISP=SHR,DSN=WORK.CMWKF02
//CMWKF03 DD DISP=SHR,DSN=WORK.CMWKF03
//CMWKF04 DD DISP=SHR,DSN=WORK.CMWKF04
//CMWKF05 DD DISP=SHR,DSN=WORK.CMWKF05
```

Notes:

1. File name assignments are not required if the procedures do not need work file support or additional printer file definitions.
2. The MPM JCL should include any other assignments that are normally used in your batch Natural JCL to execute a particular Natural subprogram or program.

4. Specify the local libraries

Specify the correct local libraries in the MPM JCL STEPLIB. Include

- the Adabas load library containing the modules STPEND and ADATSP.
- the user load library containing the user exits.
- the Natural load library containing the specially linked batch Natural nucleus.

Install the Natural Trigger Driver

The Adabas triggers and stored procedures facility uses Natural subsystems to execute user-written procedures. These subsystems are run as subtasks in the Adabas address space; a maximum of ten subsystems can be active at any given time. A Natural subsystem is fundamentally a batch Natural nucleus.

To optimize resource usage during Adabas execution, it is important to configure the Natural trigger driver (the Natural nucleus component) in a way that minimizes the resources used by the Natural subsystems.

Adabas Example Jobs

As part of your Adabas install, you loaded the data set ADA*vrs*.JOBS from tape. As a result, the following examples are installed:

Name	Description
ASMNTOS	NATOS assembly (z/OS only)
ASMPARM	NATPARM assembly
LNKBATC8	Natural nucleus link for Natural version 8.2
LNKBATCH	Natural nucleus link
LNKNATN8	Non-shared Natural nucleus link for a split nucleus for Natural version 8.2
LNKNATNS	Non-shared Natural nucleus link for a split nucleus
LNKNATS8	Shared Natural nucleus link for a split nucleus for Natural version 8.2
LNKNATSH	Shared Natural nucleus link for a split nucleus
TRGMPMJ	Startup MPM JCL
TRGPARM	NATPARM with changes required for the Adabas triggers and stored procedures facility
TRGPARM8	NATPARM with changes required for the Adabas triggers and stored procedures facility for Natural version 8.2

Note: If the examples are modified, they should be kept in a different library; otherwise, they will be overwritten when you install subsequent versions.

Batch Natural Driver NATOS

As part of your Natural install, you loaded the data set NAT*vrs*.SRCE. The source library member NATOS contains the batch Natural driver that must be used to create the batch Natural nucleus for the Adabas triggers and stored procedures facility.

Prepare the Batch Natural Nucleus

Software AG recommends that you split Natural into two functional parts: an environment-independent nucleus and an environment-dependent nucleus. See the *Natural Installation* for more information.

- The *environment-independent* nucleus, also called "shared nucleus," resides in the shared area of the operating system in the link pack area (LPA) or the extended link pack area (ELPA).

  By executing from these special areas of the operating system, the independent nucleus can be commonly accessed (shared) by multiple address spaces (or partitions or regions) within the same operating system.

  The advantages of the shared nucleus are virtual storage relief; less paging activity since there is only one copy of the nucleus in the system; and less maintenance since ZAPs are applied only once.
- The *environment-dependent* nucleus significantly reduced in size by the removal of the environment-independent parts, is loaded into the batch address space and is designated specifically for use by triggers and stored procedures.

The batch Natural nucleus used for the Adabas triggers and stored procedures facility must include the NATPARM module with the entry points for STPDRV and STPRBE.

It must be linked with the STPNAT module. The link should not use the options RENT or REUSE; otherwise, results are unpredictable. In addition, ensure that the ADALNK routine used by stored procedures and triggers (ADARUN SPT=YES) is linked with NOREUSE and NORENT, or results will be unpredictable.

The link modules must be placed in a library that is concatenated to the MPM JCL STEPLIB.

1. Assemble the batch Natural driver

Important: For Natural version 8, this job is not needed.

Use the example job ASMNTOS from the ADA*vrs*.JOBS data set to assemble the batch Natural driver.

2. Assemble the NATPARM module

See the section ***NATPARM Considerations*** for more information.

Member STPNAT is supplied in the Adabas load library and has three entry points: STPDRV, STPRBE, and ADABAS.

STPNAT replaces ADAUSER. It has the same Adabas entry point and performs all the normal functionality of ADAUSER. In addition, STPNAT contains logic specifically for triggers and stored procedures.

Note: With Natural 8.2.6 and below, it is necessary to specify CSTATIC entries for STPDRV and STPRBE.

```
CSTATIC=(STPDRV,STPRBE)
```

3. Prepare the Natural license key

If your installation uses Natural version 8.2 or higher, the license key for Natural must be prepared and assembled and linked.

For more information, refer to your Natural documentation.

4. Link the batch Natural driver

When linking the batch Natural driver that was assembled in step 1, include the NATPARM assembled in step 2 and STPNAT to create the batch Natural nucleus for the Adabas triggers and stored procedures facility.

Ensure that the Natural nucleus link deck contains an INCLUDE STPNAT statement and does *not* contain the usual INCLUDE ADAUSER statement. The AOSASM module is not required.

5. Check the REGION / SIZE parameter setting

Ensure that the `REGION` / `SIZE` parameter in the Adabas startup JCL is *not* set to impose a size limitation.

Such a size limitation, such as 8 megabytes, may have been specified when the operating system environment was set up.

6. Make the Natural nucleus accessible to the Adabas nucleus

Ensure that the Natural module created in step 3 is accessible to the Adabas nucleus during Adabas initialization by

- placing it in the Adabas load library in the MPM JCL steplib or joblib; or
- concatenating the Natural load library (where the module is located) to the Adabas MPM JCL steplib or joblib.

The Adabas load library (data set ADABAS.ADA*vrs*.LOAD) also contains the module STPEND, which must be in one of the libraries of the MPM JCL steplib.

Example:

```
//STEPLIB  DD  DISP=SHR, DSN=ADABAS.ADAvrs.LOAD
           DD  DISP=SHR, DSN=NATURAL.NATvrs.LOAD
```

7. Check the ADALNK link options

Ensure that the ADALNK routine used by stored procedures and triggers (ADARUN SPT=YES) is linked with NOREUSE and NORENT, or results will be unpredictable.

NATPARM Considerations

The NATPARM definitions specified in the Natural parameter module for a Natural nucleus are used to tailor the environment for the Natural session.

To successfully execute the Natural nucleus component, you must specify the correct values for certain NATPARM parameters. For more information, see the Natural documentation.

Special Requirements

ADAMODE
Must be set to zero.

Buffer Pool
Use the local buffer pool or the global buffer pool depending on the needs and configuration of the local environment:

- If the local buffer pool is used, a procedure that is invoked by the user can remain in the local buffer pool for the duration of the Natural session; as a result, a new copy may be ignored.
- If the global buffer pool is used, the procedure can be deleted from the buffer pool if the DBA wants it to be activated at the earliest possible time.

Buffer Sizes
Because the Natural trigger driver is a run-time system only (not a development system), various buffers can be kept to a minimum size, depending on how your programs are written. For example, ESIZE can be set to the maximum GDA size that any Natural program may use. The size of the GDA used by the Adabas triggers and stored procedures facility is 12K.

CDYNAM
The parameter regulates dynamic loading of non-Natural programs. Set with no consideration for triggers and stored procedures because the procedures will be linked to the Natural nucleus. The default value is 5.

CSTATIC
The `CSTATIC` parameter regulates programs that are statically linked to Natural. There is no default value. The parameter must specify the routines used by the Adabas triggers and stored procedures facility:

STPDRV	trigger driver entry (see Note below)
STPEND	performs backend processing for the trigger driver
STPNAT	Adabas triggers user interface
STPRBE	the record buffer extraction routine (see Note below)

Note: With Natural 8.2.7 and above, it is not necessary to specify CSTATIC entries for STPDRV and STPRBE.

DU(mp)
The default `DU=OFF` prevents the generation of a memory dump for an abend. This setting ensures that the Natural ESTAE will be active for the duration of the session. When the Natural ESTAE is active, all program abends are trapped and the Natural session is restarted instead of terminated. This is an important performance consideration.

DYNPARM

The default DYNPARM=ON processes dynamic parameters that are supplied during Natural startup. This default *must* be used because each batch Natural subsystem is started with at least one parameter (`STACK=`). See section ***NATPARM Dynamic Overrides*** for more information.

ETA

ETA is the error transaction program. It must *not* be specified. The Adabas triggers and stored procedures facility sets up the error transaction program for the Natural session according to its own requirements.

Note: The Adabas triggers and stored procedures facility alone uses 1500 bytes; therefore, if the ETA program uses 4K in the GDA, the ESIZE is approximately 6K.

ETID

If ETID is used; that is, if `ETID=' '` (blank) is *not* specified, each task (maximum of ten tasks) must be given a unique ETID (Adabas user ID). If Natural Security is used, and if a NATPARM value is not specified for ETID, the RESTART option in the library profile and the ETID option in the user profile can be set to "N" to prevent error messages NAT3048 and NAT3009.

Limit Parameters

Because the batch Natural subsystems are long-running transactions, the `LE`, `LT`, `MADIO`, `MAXCL`, and `MT` parameters should *not* be set with limits. Setting these parameters with limits can result in an error during the execution of a procedure; in that case, the command that fired the trigger receives response code 155 (ADARSP155) or 156 (ADARSP156).

Note: A timeout parameter in the Adabas triggers profile automatically resolves the problem of long-running procedures. Alternatively, the DBA can cancel a subsystem that is busy executing a procedure.

NTLFILE

The `NTLFILE` parameter should be coded to point to the Adabas trigger file. The logical file number for the trigger file is 154. Another alternative to the `NTLFILE` parameter would be a dynamic override using the Natural `NTLFILE` parameter

PROFILE

The PROFILE parameter must *not* be specified. This parameter causes Adabas calls to be issued before all of the Natural control blocks are initialized. This will cause unpredictable results during the initialization of stored procedures and triggers.

PROGRAM

The `PROGRAM` parameter must *not* be specified. It is used when starting the batch Natural subsystem and is set to STPEND. This module is located in the Adabas triggers and stored procedures load library and should be added to the MPM JCL steplib so that it can be loaded as required.

STACK

The `STACK` parameter must *not* be specified. It is used in starting the batch Natural subsystems and is specified by the Adabas triggers and stored procedures facility.

NATPARM Dynamic Overrides

NATPARM values specified in the Natural nucleus can be defined dynamically by using the Adabas triggers profile, or a Natural Security user profile if Natural Security is installed.

`NATPARM` parameter values are obtained from the Adabas triggers profile and used when the Natural subsystems are initialized. `NATPARM` values specified in the profile override the values specified in the Natural nucleus.

Before modifying the `NATPARM` definitions, read the section ***Special Requirements***. Because each Natural subsystem can accept trigger requests pertaining to a variety of applications and/or files, it is essential that the Natural environment be set appropriately.

The `STACK` parameter must be passed; it ensures that the Natural subsystem will give control to the Natural trigger driver after it is initialized. The value of the `STACK` parameter is fixed as follows and cannot be changed by the user:

```
STACK=(LOGON:SYSSPT;STP)
```

where

SYSSPT	is the library where the stored procedure application was INPLed during the installation procedure. It is also the library where the executable procedures should be located.
STP	is the Natural trigger driver startup routine.

All other NATPARM values should be set by the user as required.

The Natural session normally continues to run as long as the Adabas nucleus is active. This means that each Natural subsystem is a long-running task; limit parameters for the Natural session must be set accordingly. Session settings cannot be modified while a Natural subsystem is running. However, certain session parameters can be modified using procedures in the same manner as normal Natural programming options.

Printer Considerations

For batch Natural subsystems, the printer is always determined by the setting of the CMPRINT label and the number of the subsystem. When running under a single address space, it becomes necessary to have different assignments for CMPRINT in order to prevent contention; multiple Natural subsystems are running and any or all of these may want to print.

Even if the different subsystems are doing opens and closes of the assigned printer, there is no guarantee that these will not conflict. Any conflicts result in an error from the operating system.

For example, if you use the WRITE (nn), PRINT (nn), or DISPLAY (nn) option, the "nn" is a permanent (unlike CMPRINT) assignment specifically for all subsystems.

If a procedure that is running in subsystem 01 executes a WRITE (01), and another procedure that is running in subsystem 02 executes a WRITE (01), an error is received from the operating system and ultimately from Natural: this should be prevented.

Dynamic CMPRINT Assignment

When the Natural subsystems are started, the Adabas trigger driver determines a dynamic printer assignment based on the "CMPRINT assignment" definition in the Adabas triggers profile.

The CMPRINT assignment in the triggers profile is a 1 to 6-byte field that must be consistent with the file assignments in the MPM JCL. This printer assignment is used as a prefix to the subsystem task number when the subsystem is started.

For example, if you specify printer TSPRT, subsystems 01 and 02 will expect labels TSPRT01 and TSPRT02 to be defined in the MPM startup JCL.

The Adabas triggers and stored procedures facility is a subtask or subsystem that runs in the background. There are multiple Natural subsystems, and it is not possible to force a procedure to be executed from any specific subsystem: a dynamic assignment for CMPRINT is made available when any WRITE, PRINT, or DISPLAY statements are used for printing information.

The PDA that is passed to the procedure contains a unique subsystem identifier, which allows you to print to CMPRT01 through CMPRT31 using a DECIDE statement.

Example:

```
DECIDE ON FIRST VALUE OF RQ-TASK    /*check subsystem number
  VALUE  '01' WRITE  (1)  NOTITLE NOHDR  text
  VALUE  '02' WRITE  (2)  NOTITLE NOHDR  text
  VALUE  '03' WRITE  (3)  NOTITLE NOHDR  text
  VALUE  '04' WRITE  (4)  NOTITLE NOHDR  text
  VALUE  '05' WRITE  (5)  NOTITLE NOHDR  text
  NONE        WRITE       NOTITLE NOHDR  text
  END-DECIDE
```

In the last case (NONE), the output will go to the printer specified by the dynamic assignment for the CMPRINT label.

If a procedure running in subsystem 01 does any printing, the output can go to the dynamic assignment of CMPRINT or, as the DECIDE statement shows, the output can go to printer 01 (CMPRT01), as specified by the MPM JCL.

If more than one printer is required, ranges of print files can be defined. For example, if five Natural subsystems are defined, the range of print files could be CMPRT01 through CMPRT05 or CMPRT06 through CMPRT10, and so on.

Work File Considerations

The label for a work file is in the following format:

```
CMWKFnn
```

where "nn" is the work file number.

Work file usage is an issue that may require some consideration; however, if there are no reads or writes to any work files, there is no need to put them in the JCL.

If work files are required, a method must be developed to ensure that contention does not occur. The same considerations that apply to printers also apply to work files.

For example, the read/write operations of a procedure could be directed to a particular work file, depending on the task number, i.e., the number of the Natural subsystem in which the procedure is run. In this case, task 01 would read/write to CMWKF01; task 02 would read/write to CMWKF02; and so on.

Since you can identify the subsystem where your procedure is executing, you can use a work file that is permanently assigned to that subsystem.

This can be accomplished using the DECIDE statement, as described in the section ***Printer Considerations***. Other solutions can be found, of course, depending on how the procedure is being used.

Natural Security Considerations

This section describes the security for running the Natural subsystems when using stored procedures. The information is relevant when using Adabas triggers and stored procedures in a Natural Security environment.

- Logging On
- Library Settings
- Security Limits

- Parameter Settings

Logging On

When logging on to Natural, you may use either AUTO=ON or AUTO=OFF.

- If you use AUTO=OFF, you must supply a user ID and password and specify "Y" for the "NATSEC LOGON Required" value in the Adabas triggers profile. Otherwise, problems occur.

  The password and user ID you provide can be variable or fixed. The value ** must be used with the variable name. It will be replaced by the Natural subsystem number to make the value unique. For example, if USER** and PSWD** are specified, the user IDs and passwords (assuming three tasks) are generated as follows:

Task	UserID	Password
01	USER01	PSWD01
02	USER02	PSWD02
03	USER03	PSWD03

  The user ID must be defined to Natural Security, with consideration given to the ETID and the batch user ID. If the same user ID is used, response code 9 (ADARSP009) or 48 (ADARSP048) may be received, thus invalidating the Natural session. It is important to remember that the Natural subsystems are all running concurrently and must be kept separate from each other.
- If you use AUTO=ON, the job name or the assigned batch user ID is used for the actual logon ID. More than one user will be signed on, depending on the maximum number of tasks. This should be taken into account in order to prevent response code 9 (ADARSP009) or 48 (ADARSP048) being issued by the Natural subsystems.

Library Settings

Define the library SYSTRG. Set startup to " " (blank) when logging on from SYSAOS, or to "menu" when logging on directly from the logon screen.

Define the library SYSSPT. The Natural subsystem logs on to this library.

Do not include any startup, error, or restart settings in the definition for either library. These settings are established automatically when the Adabas triggers and stored procedures facility initializes.

The libraries can be protected; however, the user(s) as defined using the Adabas triggers profile must have sufficient authority to log on and perform all the required processing. For example, it would not be useful to disallow a module for the subsystem user IDs and then have an error occur when the Natural trigger driver invokes that routine.

Security Limits

Natural Security allows you to override certain NATPARM settings, including

- non-activity logoff limit
- transaction duration
- CPU time (MT=)
- maximum Adabas calls (MADIO=)

The Natural subsystems are one long transaction that will execute multiple "subtransactions". If security limits are set for one or more of these, the limits apply to all programs for that session.

Parameter Settings

The following describes Natural Security parameter settings that affect the Adabas triggers and stored procedures subtasks:

Error Program
Do *not* specify an error program. Errors are handled internally by the Adabas triggers and stored procedures facility.

ETID
Any procedure that stores transaction data may subsequently be invoked from a different subsystem. In order to work, the routine GET TRANSACTION DATA must know which subsystem invoked the procedure. If this is a problem, the user may build in another form of data recovery. The ETID option should be used with caution to prevent response code 9 (ADARSP009) or 48 (ADARSP048).

Library Protection
Library protection is not required, but Adabas Online System users must be able to log on to SYSTRG and the batch Natural trigger driver must be able to log on to SYSSPT.

MADIO
Set to 0 (zero) to prevent any limits being exceeded since the Natural subsystems may run for an indefinite period of time.

MAXCL
Set to 0 (zero) to prevent any limits being exceeded since the Natural subsystems may run for an indefinite period of time.

Non-Activity Logoff Limit
Set to 0 (zero) to prevent any limit being exceeded since the Natural subsystems may run for an indefinite period of time.

Password Change Option
The Natural Security user profile includes an option that can be used to request that users change their passwords every "n" number of days. If this option is used, the NATSEC password in the Adabas triggers profile must also be modified.

Restart Program

Do *not* specify. Restarts are handled internally by the Adabas triggers and stored procedures facility.

Steplibs

No limitations. The Adabas triggers and stored procedures facility logs on to SYSSPT. Therefore, all procedures must be in the steplibs as required.

Startup Transaction

Do *not* specify. Startup is automatic during the logon by Adabas Online System or the Natural trigger driver.

III Processing and Performance

5 Processing and Performance

The Adabas trigger driver is executed as a part of the Adabas nucleus. It generally controls the whole run-time processing of a trigger. It determines whether a trigger is to be fired, initiates the Natural trigger driver, and interacts with it to ensure the correct and timely processing of the procedures.

Initialization

When the Adabas nucleus starts, it determines whether the ADARUN parameter `SPT=YES` has been specified; if so, it passes control to the Adabas trigger driver to allow it to initialize. During initialization, the Adabas trigger driver performs the activities described in the following paragraphs.

Verifying the Adabas Triggers Profile

The Adabas trigger driver verifies the Adabas triggers profile on the trigger file and extracts the session parameters to be used in processing triggers and stored procedures for the session. If no profile exists on the trigger file, the initialization of the Adabas triggers and stored procedures facility in the nucleus is terminated with an appropriate error message.

- The Adabas trigger driver checks the "triggers status" and "stored proc. status" parameter settings in the profile.

Triggers	Stored Procedure	Action
Inactive	Inactive	The Adabas triggers and stored procedures facility is not allowed to start. To the Adabas trigger driver, the "inactive" setting has the same meaning as the ADARUN parameter setting `SPT=NO`.
Active	Inactive	Triggers processing is started but any request to run a stored procedure is rejected with a response code 22 (ADARSP022).
Inactive	Active	Stored procedures processing is started but any request to run a trigger is rejected with a response code 22 (ADARSP022).

- The Adabas trigger driver obtains information about the Natural subsystems from the profile:
- The subsystem name, which must correspond to the name of the linked Natural nucleus that will process the procedures; that is, the name given to the Natural nucleus during Part III of the installation procedure.
- The number of subsystems that must be started in order to handle the work load generated by the triggers that will be fired.

Verifying the Presence of at Least One Trigger

The Adabas trigger driver verifies that the trigger file contains at least one trigger definition. If not, the Adabas triggers and stored procedures facility is not initialized and an appropriate error message is given, even if triggers status is set to "inactive".

Acquiring Storage

The total storage requirement for the Adabas triggers and stored procedures facility depends on

- the size of the work areas required;
- the buffer sizes required for the trigger table, pre- and post-trigger queues; and
- the space needed for the Natural subsystems.

The amount of space needed for the Natural subsystems is determined by the size of the Natural nucleus and the various buffers that the Natural environment needs; for example, ESIZE, DATSIZE, and FSIZE.

If the size of the overall region/address space for the Adabas nucleus is too small, the Natural subsystems will not be able to run. The error message will normally indicate "insufficient storage" or an abend of the subsystems (response code 40000109 or 40000008 may be returned). If this occurs, increase the size of the region/address space or decrease the number of Natural subsystems used to execute the procedures.

Software AG recommends splitting the Natural nucleus to minimize nucleus storage requirements.

Creating the Trigger Table

At least one trigger must be defined on the trigger file; otherwise, processing cannot continue.

After the Adabas trigger driver determines the validity of the trigger file, trigger definitions are read and entries are added to the trigger table. In a nucleus cluster environment, the trigger table is not reread for each nucleus, but is obtained from a nucleus that is already active.

In order to maintain the integrity of the system, the trigger table is not updated with new, modified, or deleted triggers unless a `REFRESH` command is issued from the Trigger Maintenance facility (see the section ***Updating the Trigger Table***).

The trigger table enhances performance. Instead of checking the trigger file itself for the existence of a trigger every time a command is processed, the Adabas trigger driver simply checks the trigger table, i.e., the buffer in memory. The sequence of the table enables the trigger driver to rapidly determine whether a trigger should be fired.

When reading the trigger file to determine entries for the trigger table, the Adabas trigger driver

- ignores any entries for the trigger, checkpoint, or security files;

- loads deactivated triggers, but ignores them until they have been activated from the Modify Trigger function in the Trigger Maintenance facility. See the section *Single Trigger Definition*.
- determines the maximum file number for the database (the highest file number used plus 10), and ignores any trigger for a file number greater than the maximum.

Ignoring out-of-range file numbers may cause a problem when the `REFRESH` command is used. One solution is to load a dummy file with a file number greater than the real maximum file number. You can then add new files that have a file number greater than the real maximum but less than the dummy file number. By having a fixed-size buffer established at nucleus initialization, storage requirements for this buffer are minimized. With a two-byte file number, the total maximum size could be very large.

Starting Natural Subsystems

After the Adabas trigger driver is initialized, it starts the Natural subsystems that are responsible for the actual execution of the procedures. The "maximum subsystems" parameter in the Adabas triggers profile determines the number of subsystems (1-10) to be started.

Each subsystem is typically a minimally modified batch Natural nucleus that runs in the Adabas address space. This affects the region size specified on the MPM startup JCL/JCS.

When a subsystem is started, the Adabas trigger driver keeps track of any change in subsystem status or activity; hence, each subsystem can uniquely identify itself to either the Adabas trigger driver or any procedure that the subsystem invokes. The user can monitor these activities by using the Subsystem Activity function, which is part of the Trigger Maintenance facility.

When a Natural subsystem becomes active:

- the Natural trigger driver gets control; and
- the Adabas trigger driver is informed that the subsystem is ready to start processing any procedures that may result from a stored procedure request or the firing of a trigger.

The subsystem queue contains an entry for each Natural subsystem that is waiting for work. When the Adabas trigger driver needs a subsystem, it examines the subsystem queue to find one that is available.

Checking for Procedures

Once the Adabas nucleus is initialized, user processing continues normally. For each command that the nucleus receives, the Adabas trigger driver determines whether a trigger needs to be fired. (Entries in the trigger table that are marked "inactive" are ignored.)

For pre-command triggers, the Adabas trigger driver checks for triggers before the command is selected for processing by the Adabas thread. This involves the `READ`, `FIND`, `STORE`, `DELETE`, and `UPDATE` commands. For these commands, the Adabas trigger driver determines whether there are any triggers to be fired; if not, command processing continues normally. Commands like end transaction (`ET`), close (`CL`), release command ID (`RC`) are not checked but are given directly to the nucleus for normal processing. The trigger check is, of course, not done for stored procedure requests.

Once a command has been processed successfully by Adabas and the response code is zero, the Adabas trigger driver determines whether there are any post-command triggers to be fired. If not, the user is informed in the usual manner. If pre- or post-command trigger checking *does* result in a trigger being fired, the Adabas trigger driver can proceed with the trigger processing.

Scanning the Trigger Table

Once it has been determined that a command is eligible for the firing of a trigger, the trigger table is scanned. For performance reasons, the order in which triggers are scanned is determined by the sequence or priority assigned to the triggers by the user (see the section ***Multiple Trigger Definitions***).

If two triggers exist for the same command class and have the same priority, they are scanned in the order in which they are read from the trigger file (that is, the ISN sequence of the records on the file). It is therefore important to specify the priority for each trigger correctly.

Triggers are scanned in the following general sequence:

Sequence	Field	Command
1	specific field	specific command
2	any field	specific command
3	specific field	any command
4	any field	any command

Creating Pre- and Post-Command Trigger Queue Entries

If a command results in a trigger being fired, or if the Adabas trigger driver determines that the command is a stored procedure request, an entry is created in the

- pre-command trigger queue if the command has not been executed; or the
- post-command trigger queue if the command has been executed successfully.

The entry contains information obtained from both the command that caused the trigger to be fired and the corresponding entry in the trigger table (for example, details about the procedure to be executed). The entry also allows the Adabas trigger driver to keep track of the status of triggers that are fired.

If a Natural subsystem is waiting for work, it is given the trigger request immediately. Otherwise, the trigger request remains in the pre- or post-command trigger queue until the next subsystem is available.

Processing the Procedures

When a trigger request is placed in the pre- or post-trigger queue and a subsystem accepts it, processing continues under the control of the Natural trigger driver.

Only two triggers (one pre- and one post-command trigger) can be fired for any one command, regardless of the results.

When a command results in a trigger being fired, the system checks whether the trigger is asynchronous or synchronous.

Asynchronous Triggers

If the trigger is asynchronous, the command does not wait for the triggered procedure to complete. The command is released and processing continues normally depending on whether the trigger is pre- or post-command:

pre-command	The command is made available for processing in the Adabas thread. The triggered procedure may be processed after the command has executed or simultaneously with command execution.
post-command	The triggered procedure and the command are processed independently. The user is informed when the trigger is fired, regardless of the results of the procedure.

Synchronous Triggers

If the trigger is synchronous (participating or non-participating), the command is held until the Natural trigger driver notifies the Adabas trigger driver that the execution of the procedure is complete.

If the return code is zero, the command is released to continue processing.

If the return code is non-zero, with user receives a response code 155 (ADARSP155) or 156 (ADARSP156) and the following information:

- the Additions 3 field contains the name of the procedure that was executed as a result of the trigger being fired.
- the first two bytes of the Additions 4 field contain the actual return code from the procedure.
- the second two bytes of the Additions 4 field contain a subcode that indicates the type of trigger that was fired:
  - subcode 15 indicates a pre-command trigger.
  - subcode 16 indicates a post-command trigger.

Processing the Results

After the procedure executes, the results are placed in the trigger request entry and the status is updated appropriately. When the Adabas trigger driver detects this, it "finalizes" the trigger processing of the command.

For both pre- and post-command triggers, the return code from the procedure determines how the results are processed as described in the following sections.

Pre-Command Triggers

Return Code	Action
zero	the command is "released" so that it can be executed.
non-zero	the command is not executed and the user receives response code 155 (ADARSP155) in the response code field of the Adabas control block.

Once the command has been executed by the Adabas thread, it may be selected again for any post-command trigger processing.

Special Processing for Synchronous Pre-Command Triggers

For synchronous triggers, a return code of "1" indicates that the procedure completed processing successfully. The response code field in the Adabas control block is set to zero to indicate the successful result. However, the command is not "released" for execution by the nucleus; instead, the results of the procedure are immediately returned to the user.

This special processing accommodates procedures that have read/write access to the record buffer and require the command to be processed in a way that is similar to a stored procedure. See the section *Using the Format and Record Buffers*.

It is possible for a procedure of a pre-command trigger to modify the contents of the record buffer before the command is executed; this can be useful with update and store commands.

Post-Command Triggers

Return Code	Action
zero	the command is considered to be successful and the user is informed with response code 0 (zero) in the Adabas control block.
non-zero	response code 156 (ADARSP156) is returned in the Adabas control block. Although the procedure returned a non-zero code, the actual command may have been successful; the results of the command execution must be interpreted by the application that issued the command.

Note: When a post-command trigger is fired and the return code from the procedure is non-zero, the data in the record buffer is not returned, even if the command was executed successfully.

Special Processing for Synchronous Post-Command Triggers

Whether it is successful or not, a post-command trigger that is synchronous and has read/write access to the record buffer may have modified the record buffer.

In the case of participating triggers, the results of the trigger may have changed the result of the command. For example, if a successfully executed `UPDATE` command fires a post-command trigger and the procedure for the trigger does not complete successfully, it may or may not perform a `BT` command. When the user is informed, response code 156 (ADARSP156) is given. The application that issued the original command must determine whether the `UPDATE` command is still in effect, and perform the appropriate action (`ET` or `BT`).

Shutdown

Shutdown may occur in the following situations:

- The Adabas trigger driver keeps track of the subsystems that fail. If *all* subsystems fail, it determines that no further processing of procedures is possible and terminates. Shut-down processing depends on the "error action" value (see the table ***Error Action***).
- The nucleus receives the `ADAEND` or `HALT` operator command and instructs the Adabas trigger driver to shut down as well:

`HALT`	the Adabas trigger driver terminates immediately.
`ADAEND`	the Adabas trigger driver terminates when all subtask activities are at ET status. If a subtask is busy and is involved in ET logic, it is allowed to finish if it is issuing commands to the current nucleus; otherwise, a message is displayed at the console (see message ADAN9M) and the subtask is terminated with the transaction incomplete.

Shut-down Processing Steps

Shut-down processing steps are as follows:

1 The pre- and post-trigger queues are checked for any waiting triggers. Response code 148 (ADARSP148) is issued to all users who are waiting for a synchronous trigger to complete execution.

2 The user is informed in the normal manner for any completed post-command triggers.

3 Response code 148 (ADARSP148) is issued for completed pre-command triggers because they will not be processed by the Adabas thread. If these commands are part of an ET transaction, the user should issue a `BT` and `ET` command as appropriate.

4 Response code 157 (ADARSP157 - command is rejected) is issued for any post-command trigger that is detected after the shutdown begins. The command was executed before shut-down began, but the triggered procedure will not be executed.

5 Any subsystem that remains active after five seconds is forced to terminate, and a message is displayed at the console. A subsystem is considered "active" if it contains a procedure that continues to run; the Natural programs are in the buffer pool and the program may be issuing database calls to another database or no database at all. In this situation, a "halt" issued to the current database may be ineffective.

6 All subsystems shut down.

7 The total numbers of triggers and stored procedures are written to the console, and the triggers status field in the Adabas triggers profile is set to "inactive". The nucleus continues shut-down processing in the normal way.

Error Action

If the shutdown is requested by the Adabas trigger driver itself, shut-down processing depends on the value assigned to the error action field in the Adabas triggers profile:

Action	Shut-down Processing
Halt	The nucleus must also be terminated. The ADAEND request originates from the Adabas trigger driver itself. Any user application still processing in the subsystem is terminated.
Ignore	The Adabas triggers and stored procedures facility terminates in the Adabas nucleus, but nucleus processing continues in the normal manner (as if the ADARUN parameter `SPT=NO` were specified). No triggers will be fired to perform any extended command processing, and certain integrity problems may result.
Reject	Any command that would normally result in a trigger being fired receives response code 157 (ADARSP157). The Adabas triggers and stored procedures facility remains active; however, all subsystems are shut down and procedure processing is discontinued.

In a nucleus cluster environment, when one nucleus is set to Ignore or Reject status, all nuclei in the cluster are also set to this status.

Abnormal Termination

See the section ***Shutdown*** for information about situations that shut down

- the Adabas nucleus, the Adabas trigger driver, and all subsystems;
- the Adabas trigger driver and all subsystems, but not the Adabas nucleus;
- all subsystems, but not the Adabas triggers and stored procedures facility or the Adabas nucleus.

Natural ESTAE / STXIT Processing

If DU=OFF (the default; no memory dump is generated for an abend) is specified in the NATPARMs, the Natural ESTAE / STXIT is active for the duration of the session.

If a program abend occurs within the Natural subsystem when the Natural ESTAE / STXIT is active, the abend is trapped: the Natural ESTAE / STXIT exit acquires control, cleans up, notifies the Adabas trigger driver, and restarts the Natural trigger driver.

Thus, the Natural session is restarted instead of terminated. This is an important performance consideration.

If DU=ON is used, the ESTAE / STXIT is not activated and the subsystem will terminate abnormally. Performance is slowed if the Adabas trigger driver must restart because it will terminate and restart the subsystem as appropriate.

Natural Subsystem Abends

An executing procedure may exceed the time-out limit, or be canceled by the DBA before it completes processing. (Time-out refers to elapsed time as opposed to CPU time usage.)

These two "abnormal" terminations of a procedure are similar, with the following exceptions:

- A cancellation is done manually from the Trigger Maintenance facility.
- A time-out occurs automatically based on the activity timeout setting in the Adabas triggers profile.

An executing procedure may need to be terminated at any stage, i.e., waiting, looping, or simply executing longer than is expected. The only sure way to intercept processing is to terminate the subsystem itself. (You can observe this termination by monitoring the subsystem from the Trigger Maintenance facility; see the section ***Subsystem Activity***.)

If the trigger is synchronous, the user must be informed that the subsystem has been terminated. Depending on the trigger type (pre- or post-command), response code 155 (ADARSP155) or 156 (ADARSP156) is set, with subcode 9 to indicate the timeout. The user is always informed, whether the subsystem timed out or was deliberately terminated, and additional information is provided in messages that are written to the console.

When an abnormal termination occurs, ascertain the reason and correct the problem. If this happens continuously, deactivate the trigger until the problem has been solved. Use the trigger activity logging option on the profile to determine the reason more easily.

Natural Subsystem Restart

If Natural is unable to recover and a subsystem terminates, the Adabas trigger driver is notified of the error and a message is written to the console.

After a Natural subsystem is terminated, it is restarted automatically.

- If the restart is successful, the subsystem is reinitialized in the usual way. If there is not currently work to be done, the subsystem is placed in a queue to wait until the Adabas trigger driver informs it that a new trigger was fired and the procedure needs to be processed.
- If the restart is unsuccessful, it is attempted two more times. The subsystem is permanently deactivated if it fails to start after three consecutive restart attempts, and a message is displayed at the console to inform the user. The subsystem cannot be reactivated until the nucleus is shut down and restarted.

The restart routine responsible for restarting the Natural trigger driver is STP.

Users who are waiting for a synchronous trigger to finish processing are notified that the trigger did not complete successfully. Response code 155 (ADARSP155) or 156 (ADARSP156) is returned, with additional information in the Additions 4 field.

Command Logging

Because a triggered procedure may reject a command, it is possible when running Adabas triggers and stored procedures that a command issued by the user is never run in an Adabas thread or even seen by the nucleus supervisor.

It is therefore necessary for the Adabas triggers and stored procedures facility to log

- a `PC` command (that is, a stored procedure request) when it is received;
- a command when it is selected for a pre-command or post-command trigger;
- a pre- or post-triggered command when it receives a non-zero response.

When processing a command log record, the log record types are

- X'0005' for pre-command triggers; and
- X'0006' for post-command triggers.

For the Adabas control block in the log record, the Additions 3 field contains the name of the procedure being invoked and the Additions 4 field contains the following information about the procedure:

Byte	Contains...
1-2	the response code from the procedure.
3	"P" for pre-trigger; "R" for procedure call; or "S" for post-trigger.
4	"A" for asynchronous; "N" for non-participating; or "P" for participating.
5	"R" for read; "F" for find; "U" for update; "S" for store/add; "D" for delete; or "P" for procedure call.
6	flag settings: "1" for no parameters; "2" for response code only; "4" for control information; "8" for special stored procedure parameters; "10" for record buffer access; or "80" for record buffer update.
7-8	the name of the field associated with the trigger.

Note: No log is created for an asynchronous trigger that returns a non-zero response. The CQE address (4th parameter) for an asynchronous trigger is set to zero if the command completes before the trigger is processed.

IV Programming and Performance

6 Programming and Performance

In this chapter, guidelines are suggested and options are provided for writing efficient procedures. Special requirements for using Natural are included as well as information about using the format and record buffers, and the record buffer extraction routine STPRBE.

Writing Procedures

It is important to consider performance when writing procedures. No more than ten subsystems are available to process all commands for all users. Procedures that require excessive processing time prevent the subsystem from servicing other requests.

If all subsystems incur a heavy load, it can cause excessive queuing of trigger and stored procedure requests. Commands for synchronous triggers and stored procedures are forced to wait, and users may experience poor response times.

Stored Procedures

Stored procedures may be created in library SYSSPT, or in another library that is a steplib to SYSSPT.

The stored procedure is a normal Natural subprogram, with the following characteristics:

- it must use the parameter definition in the supplied PDA.
- it should never go to level 1 in the Natural calling structure.
- it must not override the current setting of *ERROR-TA if errors are to be handled by the Natural trigger driver directly.

Triggers

The function performed by a triggered procedure is based on a specific file. It may also use a specific command and/or field. The function should be very specific in order to reduce the number of operations that the procedure must perform. A maximum of one pre-command trigger and one post-command trigger can be fired for any given command.

Asynchronous Triggers

Asynchronous triggers are useful for determining whether an actual event occurred.

- If file number is the only criterion, the procedure can be used to report and/or audit any activity for the file as a whole.

  Information about the command, i.e., the Adabas control block, is available; the contents of the record buffer are *not* available because the command does not wait for the procedure to execute

completely before continuing. However, the record may be read using the ISN in the control block.

- If 'command' is added as a criterion, this reduces the number of times a trigger is fired.

  An advantage of asynchronous triggers is that the command that results in the trigger being fired does not have to wait for the procedure to complete; however, asynchronous triggers should generally be used only if the procedure logic is independent of the application.

Synchronous Triggers

Synchronous triggers normally have a dependency associated with the application, i.e., there is a need to have some action take place either before or after the command is processed. The command cannot continue until the procedure has completed.

When a synchronous procedure receives control, it can perform processing that is similar to that of an asynchronous procedure. If 'field name' is specified as a criterion, the procedure can retrieve that value (read the ISN of the record or call STPRBE) and perform any required processing.

Implementing Support for Multi-Triggers

For each command, only one pre- and one post-trigger may be fired. If you require triggers for multiple fields, you must implement logic to support "multi-triggers" (a single trigger containing multiple, related triggers).

Software AG recommends that you use a procedure rather than the Trigger Maintenance facility to define multi-triggers for a specific file and command. Such a procedure

- takes control when the trigger is fired and then invokes other procedures to simulate triggers being fired.
- can define the exact sequence in which the procedures are processed.
- can define the action to be taken if a procedure returns a non-zero response code; for example, check for additional triggers or return the response code to the user.
- can introduce additional criteria; that is, criteria other than file number, command, and field name. For example, a procedure could be invoked only when a combination of two fields exists in the format buffer.

Example of a multi-trigger:

Trigger Criteria:	File 11 Command UPDATE Field '** any field **' Procedure PROC001
Command A1:	Format Buffer='AA,BB,AB,CA,DF,MT,DT.'
Procedure:	The subprogram handles multi-triggers in the following sequence: AB, GA, DT, AA, LT, CA. When PROC001 gets control, it checks for each field in the sequence. If a field is not found, the procedure for that field is ignored and processing continues with the next field. In this example, fields GA and LT are not found (are not in the format buffer). Processing can continue regardless of any nonzero response code returned by any of the procedures.

Single Triggers vs Multi-Trigger

If the list of triggers is very long, it may be more efficient to use multiple single triggers instead of one multi-trigger.

- A trigger should exist for each file-plus-command combination as required.
- If the fields can be grouped, the trigger field for each trigger can be one of the fields in a group; that is, the field that is always referenced by all accesses and/or updates.

This approach makes sense when a file has record typing or the applications read and update the data for a file in groups.

Example:

An application such as Employees has two separate functions (based on the Employees sample file supplied with the Adabas installation):

- Updating personnel details such as address and name. If telephone number is always included in the data, the trigger can be fired for an UPDATE to the EMPLOYEES file for the field TELEPHONE.
- Updating job details such as department, vacation status, and position. The trigger can be fired for an UPDATE to the EMPLOYEES file for the field JOB-TITLE or the field POSITION.

Natural Syntax Limitations

When coding procedures, you may use the normal Natural syntax. However, the limitations discussed in the following paragraphs should be considered.

Error Handling

STPPDRIV contains an ON ERROR clause so that any errors that result from the procedures are trapped in STPPDRIV.

- STPPDRIV passes control to the STP routine.
- The STP routine handles the restart processing.

As part of restart processing, STP informs the Adabas trigger driver that the trigger request in progress has been terminated and that the result should be returned to the user.

The following rules apply:

- The ON ERROR clause may be used, provided control is returned to the calling routine, i.e., STPPDRIV.
- For performance reasons, statements such as STOP or TERMINATE should not be used. If a TERMINATE NN 'message' statement is issued by the procedure, it causes the subsystem to terminate with a return code of NN and a message written to the console. The Adabas trigger driver must then restart the system.

If these rules are not followed, the Natural trigger driver may lose control, requiring that the user cancel the batch Natural subsystem from the Adabas trigger driver. When the Adabas trigger driver becomes involved in error handling, additional resources are required.

Printer Support

Although reports and messages can be printed from a Natural subsystem, the subsystem should not be expected to function as a batch processor. Printer files may be assigned to the various subsystems, but should be used with caution. Natural subsystems have no control over which procedures they execute for any application.

Reports are normally available in their complete form. While the nucleus is active, the latest messages or report output may not be available because the spool system has not flushed its buffers.

Work File Support

Work files are supported; however, it is not known which work files will be assigned to a subsystem at the time of execution. Therefore, procedures should be written to read and write to a specific work file based on the subsystem where the procedure must run.

ET Logic

End transaction (ET) logic works in the usual way; however, be aware of the effects of participating triggers:

- An END TRANSACTION (ET) or BACKOUT TRANSACTION (BT) statement issued from a procedure affects the "hold" status of any records of the user who issued the command that fired the trigger.
- The ET or BT statement should be issued in a way that allows the applications to be synchronized. For instance, if the procedure incurs an error, it should be able to issue a BT to back out any previously updated data; that is, data modified within the procedure as well as data modified by the application to which the trigger command belongs.

When an asynchronous or non-participating trigger completes execution, a BT should be issued to ensure that the subsystem is at ET status for the next trigger to be fired.

Natural Levels

Natural program levels are changed whenever a FETCH RETURN, CALLNAT, or PERFORM statement processes an external subroutine. In order for the Natural trigger driver to stay in control of the Natural session, it is important to maintain program levels; therefore, FETCH and STOP statements (assuming STACK TOP COMMAND) must *not* be used. Hence, STPPDRIV, which invokes the actual procedure/subprogram, must be at level 1.

Statements Appropriate for Batch Mode

Procedures are executed in batch mode. Thus, statements such as INPUT should be used appropriately with the STACK DATA statement. The `STACK` command is not appropriate.

CALLNAT Parameters

The Natural trigger driver invokes a subprogram with a CALLNAT statement. Depending on the definition in the Trigger entry, the subprogram should expect one of the following options to be used:

1. No parameters are passed.

   This is typically used for asynchronous triggers that perform a very specific function. That is, the trigger event criteria is the only information that the procedure needs in order to perform

the task. A response code is not passed because it is irrelevant; the command may already have completed, so a non-zero response code from the procedure cannot be returned to anyone.

2. The response code field only is passed.

   The response code field is a modifiable, four-byte binary (B4) field. The Natural trigger driver checks all four bytes to determine whether the call was successful. However, only the last two bytes contain the actual response code. The first two bytes may be used as a subcode for reporting reasons.

   The Natural trigger driver returns the response code for a synchronous trigger but ignores the response code for an asynchronous trigger. Because the asynchronous trigger may execute after the initiating command has finished executing, the response code is irrelevant.

   The response code value is placed in the first two bytes of the Additions 4 field of the command before it is returned to the user; it is not placed in the response code field of the Adabas control block. It is usually response code 155 (ADARSP155 - indication of pre-command triggers) or 156 (ADARSP156 - indication of post-command triggers).

3. The response code field and the control information are passed.

   The response code field is passed in the same manner as described above.

   Control information about a trigger may be vital to the successful execution of a procedure. It is provided in the `STPAPARM` or STPXPARM parameter data areas (PDAs), and includes

   - the command code of the Adabas command that initiated the trigger
   - the DBID of the database against which the command was issued
   - the file number
   - the length of the command's record buffer
   - a setting that indicates whether the trigger can modify the initiating command's record buffer
   - settings that indicate whether the trigger is asynchronous, non-participating, or participating
   - the task ID of the batch Natural subsystem that is executing the trigger
   - a copy of the command's Adabas control block (ACB) or extended Adabas control block (ACBX). With asynchronous triggers, this is only the first 48 bytes, i.e., everything up to the Additions 2 field.

4. The response code field, the control information, and a global user area are passed.

   The global user area is passed, in addition to the response code field and control information described above.

   The global user area is kept for the entire active Natural subsystem session. Because the global user area is never modified by the Natural trigger driver, it can be used to pass information between procedure calls. It can also be used as a working storage area.

Using the Format and Record Buffers

Record Buffer

- The record buffer is valid with stored procedures only if parameters are being passed.

  The record buffer is available for passing parameters from the caller to the stored procedure and/or from the stored procedure to the caller. The layout or DSECT of the record buffer must be coordinated between the caller and the actual stored procedure itself.
- The record buffer is valid with triggered procedures only if the trigger is participating or non-participating (synchronous) and only through the record buffer extraction routine (STPRBE). The record buffer can be passed either before or after the trigger command is processed.

  The record buffer to be used for access or update commands is specified by the caller using the Additions 4 field in the Adabas control block.

Parameters

When setting up the record buffer, the definition of the parameters is significant. Multiple parameters with differing lengths could be passed as

- a separate field with the length value before each parameter;
- a list of fields at the start or end of the parameters; or
- the first two bytes of each parameter, that is, inclusive length bytes.

The options for passing parameters are varied and flexible; you should choose the one that is most effective or consistent with your environment:

1. No parameters are passed.
2. Fixed Definition

   The parameters are placed in a contiguous piece of storage with a fixed length and fixed definition. Each parameter remains constant in the structure and also has a fixed length. The procedure that is invoked must therefore be able to interpret the structure or DSECT defined for the parameters. See example ***STPLNK01*** .
3. Fixed List of Parameters

   There is more than one parameter, but the number, sequence, format and length of each parameter is constant when STPLNK is invoked. STPLNK need only place these parameters in contiguous storage; that is, the record buffer, before issuing the request. See example ***STPLNK02***.
4. Variable List of Parameters

In the more flexible variation of passing parameters, the procedure that is invoked must understand the format of the parameters being passed in the record buffer. The format buffer can be very useful in providing this information; hence, the procedure events field DD. The record buffer extraction routine, STPRBE, must scan the format buffer for the field and step through the record buffer at the same time to position to the correct value (start and end position). See example ***STPLNK03***.

In either case, read or write access to the record buffer must be allowed using the `RBOPT` parameter option.

Format Buffer

The format buffer can optionally be used to convey the definition of parameters being passed in the record buffer.

- The syntax must be consistent with that of a format buffer for a normal command, or be set to "." if it is not used.
- The field names should normally be meaningful so that the procedure can get the values of each parameter from the record buffer extraction routine (STPRBE).
- Length must be used if the procedure does not provide a length specification. Alternatively, if the field names correspond to the actual file number specified in the ACB, then the STPRBE routine is able to determine the length of the field or parameter.
- If required, STPRBE can be used to extract the actual format buffer contents.
- When calling stored procedures across platforms, the field type of each parameter must be specified as
  - A for alphanumeric;
  - B for binary;
  - U for unpacked; and so on.

Record Buffer Extraction Routine (STPRBE)

STPRBE is the record buffer extraction routine. It may be used in Adabas procedures to request record buffer information. It can be called

- from a stored procedure if parameters are being passed; or
- from a command that results in a synchronous trigger being fired. STPRBE provides the only access to the record buffer for synchronous triggers.

For all functions, the record buffer extraction routine is called as follows:

```
CALL 'STPRBE' FUNCTION REQUEST-PARM REC-BUFFER
```

FUNCTION, REQUEST-PARM, and REC-BUFFER are described below.

REQUEST-PARM (A154)

REQUEST-PARM consists of the following fields:

Field Name	Length	Description
ERR-MESSAGE	A72	Error message text for any error received
RESPONSE-CODE	I4	Response code (subcode + actual error) from this routine
VERSION NUMBER	A4	Version of this structure
FIELD NAME	A32	Long name of the field to be extracted
FIELD FORMAT	A1	Format of the field to be extracted
FIELD OPTIONS	A3	Special options
FIELD LENGTH	I4	Length of field extracted
FIELD SHORT	A2	Adabas short name of field
Unused	A2	Unused
FIELD OCCUR	I4	Occur number for PE/MU field
GROUP OCCUR	I4	Occurrence number of MU within the PE Field
FIELD OFFSET	I4	Offset into record buffer for extraction
Unused	A18	Unused

FUNCTION (A4)

The following table describes the functions that should be specified.

Field	Description
GF	Get the contents of the format buffer.
GI	Get index value (obtain an index occurrence of an MU or PE).
GM	Get multi value (obtain an MU field within a PE).
GR	Get record buffer range (obtain the record buffer information according to the start position and length specified by the call).
GV	Get value (obtain a flat field value from the record buffer).
GX	Get UBX information. This is different from the "NR" function; i.e., it allows the user to obtain any information passed using the UB extension in accordance with the definition of user exit B in ADALNK.
NR	Get Natural information (obtain the Natural user information from the user buffer according to the offset and length passed). This is valid only if the user buffer extension is being used through user exit B in ADALNK.
UI	Update index value (change an index occurrence of an MU or PE).
UM	Update multi value (change an MU field within a PE).
UR	Update record buffer range (change the record buffer information according to the start position and length specified by the call).
UV	Update value (change a flat field value from the record buffer).

The following table contains the required value for each of the functions:

Field	GF	GI	GM	GR	GV	GX	NR	UI	UM	UR	UV
ERR-MESSAGE	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U
RESPONSE-CODE	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U	_/U
FIELD NAME		F/A	F/A		F/A			F/A	F/A		F/A
FIELD FORMAT		f/U	f/U		f/U			f/U	f/U		f/U
FIELD OPTIONS		G			C/G			G			G
FIELD LENGTH	F	f/U	f/U		f/U	F		f/U	f/U		f/U
FIELD SHORT		f/U	f/U		f/U			f/U	f/U		f/U
FIELD OCCUR		F/A	F/A					F/A	F/A		
GROUP OCCUR		f/A	F/A					f/A	F/A		
FIELD OFFSET	F		F/A			F/A		F/A			

Symbol	Meaning
_	Not used
A	Remains unchanged after the call
F	Filled in before the call
f	Optionally filled in before the call
C	Field count - valid only with MU or PE fields when the value of the count field is required.
G	Group option - extract/update an elementary field within a group field
U	Updated after the call unless specified; for example, length

REC-BUFFER

REC-BUFFER is a variable length field that contains one of the following:

- the record buffer that the user passed to Adabas with the original call; that is, the call that resulted in the triggered procedure being invoked; or
- the record buffer returned from the user after the actual processing of the triggered procedure.

Response Codes

Code	Meaning
0 (ADARSP000)	Call was 100% successful.
1	Internal error occurred; that is, Get Name mapping failed.
2	Long name not found in name mappings; that is, name did not exist in the file-field table definitions.
3	Reserved.
4	Field not found in format buffer; that is, the user requested the value of a field not present in the format buffer.
5	Field too deep in the record buffer. The length of the record buffer is determined by the record buffer length (RBL) in the Adabas control block (ACB). If the field exists, it is beyond the defined limit.
6	Requested occurrence not present in format buffer. Informs the user that a specific occurrence of an MU or PE was not found in the format buffer.
7	Invalid function type; that is, FUNCTION is not set to a valid value.
8	Record buffer access not available; that is, the record buffer option field was set to none, disallowing any access. Therefore, no record buffer extraction functions are available to this procedure.
9	Record buffer modification denied; that is, while access to the record buffer is permitted, write access has been disallowed. Therefore, only GET functions may be requested.
10	Record buffer length (RBL) exceeds the maximum allowable in the ACB; that is, the length specified extends past the total length of the RBL. Also applies to the option "NI". Only 232 bytes of the extended buffer may be retrieved.
11	Record buffer length is incorrectly set to 0; that is, the record buffer extraction routine was expecting the user to specify the length of the RBL for this function call, but it is set to zero. (example: RAngeReq)
12	Illegal syntax in the format buffer. Syntax that is valid for a command may not be resolved by the record buffer extraction routine. This normally applies when the "n" syntax is used; for example, AB1-n, ABn. However, if the trigger is a pre-command trigger, the problem could be that the format buffer itself is invalid.
13	Invalid parameter is specified; that is, one of the parameters that should have been set for the function call was not set. For example, field name function call was not set.
14	PE group occurrence was not specified; that is, the function call for accessing/updating a particular PE or MU occurrence was specified but no occurrence number was passed in the parameters.
15	Invalid edit mask specified in the format buffer. Only valid edit masks may be specified.
16	Reserved.
17	User information not available - link B missing; that is, the user is trying to access the user buffer extraction information but none is available.
18	Group definition record not found for the current file.

Code	Meaning
19	Field not found in the group definition record for the current file.
60	Syntax error in the format buffer; that is, like the nucleus, the record buffer extraction routine expects the format buffer to conform to the normal rules of definition; however, the current format buffer is invalid. Check the subcode in the error message (refer to the *Adabas Messages and Codes* documentation).
3xxx	Nonzero Adabas response code was returned, where "xxx" is the actual Adabas response code. This may occur while the record buffer extraction routine is accessing the trigger table to obtain more information.
9999	The GDA is incorrect or corrupted, or there is none. This indicates an internal error. Check for any messages from the Natural subsystem that indicate some kind of error, or contact your Software AG technical service representative. A dump will be needed.

Get Value by Offset

- Range request to move record buffer by offset.

  The user may access or update the record buffer in accordance with a specific offset for a length equal to or less than the original RBL (record buffer length). That is, RBE (record buffer extraction) should return the value of the record buffer at a certain position for a certain length, regardless of any "field" sizes or definitions that the record buffer contains.

- Request to move user buffer extraction (UBX) information by offset.

  The user may access the record buffer in accordance with a specific offset for a length equal to or less than the original RBL. For the UBX, this is a maximum value of 232.

- Request to move format buffer information by offset.

Get Field Value

An Adabas field name must be specified for this request:

- If the short name is specified, the long name should be set to '**'.
- Length may or may not be specified. If length is not specified, an override value less than the maximum defined size of the field may be specified instead.

During processing, RBE must step through the record buffer and the format buffer. For each field found in the format buffer prior to locating the desired field, RBE must step through the record buffer in order to be positioned correctly for the actual move.

Stepping forward through MU and PE fields requires that the total number of elements involved in the array be calculated, as follows:

- the number of PE occurrences times the number of MU occurrences times the actual length of the field in question.

Note: In the case of an MU where the user has not specified the occurrence number in the format buffer, the user receives the first occurrence; i.e., the request is treated like a request for an "elementary" field.

RBE determines whether the user has an MU or PE field only, or an MU within a PE field. The following are examples of supported PE fields and/or usage in format buffers (where "m" and "n" are actual occurrence values).

- MUn or PEn
- PEn(MUn)
- PEn(MUm-n)
- PEm-n(MUn)
- PEm-n(MUm-n)
- PEm-n or MUm-n

To obtain the MU count or PE count from the record buffer, the field name or short name must be specified; length to be returned may or may not be specified.

V Calling Stored Procedures

7 Calling Stored Procedures

Stored procedures allow you to directly invoke a procedure located on the database using the Adabas direct command `PC`.

Stored Procedure Link Routine (STPLNKnn)

The `PC` command is used in conjunction with the stored procedure link routine STPLNKnn to invoke a stored procedure.

STPLNKnn is provided in the SYSSPT library in source format.

The examples STPLNK01, STPLNK02, and STPLNK03 from the library SYSSPT illustrate the use of the PC command in calling stored procedures. Each example passes parameters to the routine in a different way.

You may use these examples or write your own routines. If you use the examples, you may change the routine code or name to meet standards or requirements at your site. You may choose to include the routine name as inline code in the main Natural program.

In the three examples, the PC command is invoked by calling the Natural routine CMADA. If you do not want to code this entry name directly, you can issue a CALLNAT to the Natural subprogram USR1043N in library SYSEXT instead. The advantage of using the CALLNAT alternative is to insulate your code from changes to the name "CMADA" that may occur across time or across platforms.

Setting Up the PC Command

The Adabas control block (ACB) for the PC command (direct call) must be set up before STPLNKnn is used to invoke a stored procedure request.

This section covers the following topics:

- PC Command Function and Use
- ACB Interface Direct Call Control Block and Buffer Overview
- ACBX Interface Direct Call Control Block and Buffer Overview

- Buffers

PC Command Function and Use

The PC command provides a mechanism for invoking stored procedures.

Parameters are passed using the record buffer; they are subsequently updated by the stored procedure and returned to the caller.

The format buffer may be used to define the parameters to the procedure. Such information may be relevant when calling the record buffer extraction routine.

ACB Interface Direct Call Control Block and Buffer Overview

- Control Block
- Buffer Areas
- Control Block Field Descriptions

Control Block

Field	Position	Format	Before Adabas Call	After Adabas Call
	1-2	Not used	Not used	Not used
Command Code	3-4	alphanumeric	F	U
Command ID	5-8	alphanumeric	F	U
File Number	9-10	alphanumeric	F	U
Response Code	11-12	binary	Not used	A
	13-24	Not used	Not used	Not used
Format Buffer Length	25-26	binary	F	U
Record Buffer Length	27-28	binary	F	U
	29-36	Not used	Not used	Not used
Additions 1	37-44	alphanumeric	F	U
Additions 2	45-48	binary / binary		A
Additions 3	49-56	alphanumeric	F	A
Additions 4	57-64	alphanumeric	Not used	A
	65-76	Not used	Not used	Not used
User Area	77-80			U

Buffer Areas

Buffer	Before Adabas Call	After Adabas Call
Format	F	U
Record	F	A

where:

F	Supplied by user before Adabas call
A	Supplied by Adabas
U	Unchanged after Adabas call

Control Block Field Descriptions

Command Code (ACBCMD)

PC

Command ID (ACBCID)

Set this field to the value 'STPx' where "x" is any value.

File Number (ACBFNR)

By default, indicates the trigger file database ID and file number.

For a one-byte database ID, set CB-DBID; for a two-byte database ID, set CB-RSP with CB-CALL-TYPE set to X'30'.

You may specify the file number of any other user file in conjunction with the format buffer. File number should be consistent with the format buffer so that the record buffer extraction (STPRBE) routine may be used to interpret or retrieve field values according to the file-field definitions.

Specify the binary number of the file to be read in this field. For the physical direct calls, specify the file number as follows:

- For a one-byte file number, enter the file number in the rightmost byte (10); the leftmost byte (9), should be set to binary zero (B'0000 0000').
- For a two-byte file number, use both bytes (9 and 10) of the field.

Note: When using two-byte file numbers and database IDs, a X'30' must be coded in the first byte of the control block.

Response Code (ACBRSP)

Adabas returns the response code for the command in this field. Response code 0 (ADARSP000) indicates that the command was executed successfully. Non-zero response codes, which can also have accompanying subcodes returned in the rightmost half of the Additions 2 field, are described in the *Adabas Messages and Codes* documentation.

Format Buffer Length (ACBFBL)

The format buffer length (in bytes). The actual format buffer area defined in the user program must be at least as large as the length specified.

Record Buffer Length (ACBRBL)

The record buffer length (in bytes). The actual record buffer area defined in the user program must be at least as large as the length specified.

Additions 1 - Name of the Stored Procedure - (ACBADD1)

The name of the stored procedure.

Additions 2 - Length of Compressed and Decompressed Record - (ACBADD2)

The PC command returns a response from the procedure executed in bytes 1 and 2 of this field.

Additions 3 - Stored Procedure Options- (ACBADD3)

This field indicates the options to be used when the stored procedure request is issued:

Byte 1 is Type:	A=asynchronous, P=participating, N=non-participating
Byte 2 is Parm:	N=none, C=control, E=error/response, X=control with ACBX
Byte 3 is RecB:	N=none, A=access, U=update

Additions 4 (ACBADD4)

The PC command returns a response from the procedure executed in bytes 1 and 2 of this field. Bytes 3 and 4 are set to X'0011 (17) to indicate "stored procedure".

ACBX Interface Direct Call Control Block and Buffer Overview

Control Block

Field	Position	Format	Before Adabas Call	After Adabas Call
	1-2	binary	---	---
Version Indicator	3-4	binary	F	
	5-6	binary	---	---
Command Code	7-8	alphanumeric	F	U
	9-10	binary	---	---
Response Code	11-12	binary	---	A
Command ID	13-16	alphanumeric/ binary	F	U
Database ID	17-20	numeric	F	U
File Number	21-24	numeric	F	U

Field	Position	Format	Before Adabas Call	After Adabas Call
	25-56	---	---	---
Additions 1	57-64	alphanumeric	F	U
Additions 2	65-68	binary	---	A
Additions 3	69-76	alphanumeric	F	A
Additions 4	77-84	alphanumeric	---	A
	85-114	---	---	---
Error Subcode	115-116	binary	---	A
	117-144	---	---	---
Command Time	145-152	binary	---	A
User Area	153-168	not applicable	---	U
	169-193	---	---	---

Buffer Areas

Buffer	Before Adabas Call	After Adabas Call
Format	F	U
Record	F	A

where:

F	Supplied by user before Adabas call
A	Supplied by Adabas
U	Unchanged after Adabas call

Control Block Field Descriptions

Version Indicator (ACBXVER)
F2

Command Code (ACBXCMD)
PC

Response Code (ACBXRSP)
Adabas returns the response code for the command in this field. Response code 0 (ADARSP000) indicates that the command was executed successfully. Non-zero response codes, which can also have accompanying subcodes returned in the Error Subcode (ACBXERRC) field, are described in the *Adabas Messages and Codes Manual* documentation.

Command ID (ACBXCID)
Set this field to the value `STPx` where *x* is any value.

Database ID (ACBXDBID)

Specify the database ID for a call.

If this field is set to binary zeros, the Adabas API uses either the database ID from the ADARUN cards provided in DDCARD input data, or the default database ID value provided in the LNKGBLS module linked with or loaded by the link routine.

File Number (ACBXFNR)

By default, indicates the trigger file number.

You may specify the file number of any other user file in conjunction with the format buffer. File number should be consistent with the format buffer so that the record buffer extraction (STPRBE) routine may be used to interpret or retrieve field values according to the file-field definitions.

Specify the binary number of the file to be read in this field. For the physical direct calls, specify the file number as follows:

- For a one-byte file number, enter the file number in the rightmost byte (10); the leftmost byte (9), should be set to binary zero (B'0000 0000').
- For a two-byte file number, use both bytes (9 and 10) of the field.

Additions 1 - Name of the Stored Procedure - (ACBXADD1)

The name of the stored procedure.

Additions 2 - Length of Compressed and Decompressed Record - (ACBXADD2)

If the command is processed successfully, the following information is returned in this field:

- If the record buffer contains at least one valid field value, the leftmost two bytes contain the length (in binary form) of the compressed record accessed;
- The rightmost two bytes contain the length (in binary form) of the decompressed fields selected by the format buffer and accessed.

Note: This length information is not returned when the prefetch feature is being used.

Additions 3 - Stored Procedure Options- (ACBXADD3)

This field indicates the options to be used when the stored procedure request is issued:

Byte 1 is Type:	A=asynchronous, P=participating, N=non-participating
Byte 2 is Parm:	N=none, C=control, E=error/response, X=control with ACBX
Byte 3 is RecB:	N=none, A=access, U=update

Additions 4 (ACBXADD4)

The `PC` command returns a response from the procedure executed in bytes 1 and 2 of this field. Bytes 3 and 4 are set to X'0011 (17) to indicate "stored procedure".

Error Subcode (ACBXERRC)

If the command returns a nonzero response code, this field contains a subcode defining the exact response code meaning. Response codes and their subcodes are defined in the *Adabas Messages and Codes Manual* documentation.

Buffers

The following buffers apply to the PC command:

- Format Buffer
- Record Buffer

Format Buffer

The user specifies the fields to be read in this buffer. Format buffer syntax and examples are provided in the *Adabas Command Reference* documentation.

The format buffer may optionally be used to convey the definition of the parameters being passed in the record buffer. The syntax must be consistent with that of a format buffer for a normal command, or be set to "." if it is not to be used.

The field names used in the format buffer should normally be meaningful so that the stored procedure can acquire the values of each parameter from the record buffer extraction (STPRBE) routine (see ***Record Buffer Extraction Routine (STPRBE)***). Length must be used if the stored procedure routine does not provide one. Alternatively, if the field names correspond to the actual file number specified in the ACB, then the STPRBE routine will be able to determine the length of the field/parameter.

When issuing stored procedures across platforms, it is essential to also specify the field type of each parameter; i.e., "A" - alphanumeric, "B" - binary, "U" - unpacked, etc.

See ***Format Buffer*** for more information.

Record Buffer

Adabas returns the requested field values in this buffer. All values are returned according to the standard format and length of the field unless the user specifies a different length and/or format in the format buffer.

The record buffer is available for passing any parameters from the caller to the stored procedure and/or from the stored procedure to the caller. The layout or DSECT of the record buffer must be coordinated between the caller and the actual stored procedure routine itself.

The record buffer is available for participating and non-participating (sync) type requests *only* by using the record buffer extraction (STPRBE) routine. See ***Record Buffer Extraction Routine (STPRBE)***.

Whether the record buffer is used for access or update is specified by the caller using the Additions 4 field.

See *Using the Format and Record Buffers* for more information

Examples

This section contains the example programs and data areas listed in the following table. Source code is provided during the installation procedure and is located in the library SYSSPT.

Name	Description
STPLNK01	This stored procedure link routine passes parameters as fixed length and fixed number.
STPLNK02	In this stored procedure link routine, a maximum of five parameters may be passed to the procedure; the length of each parameter is contained in the first two bytes of the parameter.
STPLNK03	Like STPLNK03, a maximum of five parameters may be passed to the procedure; however, the length of each parameter is contained in a preceding, separate parameter.

STPLNK01

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *  Subprogram : STORPROC/STPLNK01
0040 *  Author     : Adabas Development
0050 *
0060 *  Function   : Sample Routine 01 to invoke a stored procedure
0070 *               This example expects fixed parameter definitions
0080 *  Remarks    : This routine will set up the buffers and issue the call
0090 *               to invoke a stored procedure routine directly.
0100 *               Once processing is completed, control is returned to
0110 *               the caller.
0120 *               Parameter RESP must be set to zero if processing is
0130 *               successful.
0140 *
0150 *  Parameters : The following fields in the ACB must be set up to invoke
0160 *               the stored procedure request.
0170 *
0180 *               Command Code: 'PC'
0190 *               Command ID  : 'STPx' - where x is any value
0200 *               Database ID : Database of the respective trigger file
0210 *                             Set CB-DBID for a one byte DBID
0220 *                             Set CB-RSP  for a two byte DBID with
0230 *                               CB-CALL-TYPE set to H'30'
0240 *               File Number : Set to the trigger file number of the
0250 *                             target database (normal one-byte versus
0260 *                             two-byte FNRs is applicable) by default
```

```
0270 *                              or any other file used in conjunction with
0280 *                              the format buffer.
0290 *               FB Length   : Length of the format buffer
0300 *               RB Length   : Length of the record buffer
0310 *               Additions 1 : Name of the stored procedure
0320 *               Additions 3 :
0330 *                  Byte 1   : Type ("A"sync, "P"art, "N"on-Partic)
0340 *                  Byte 2   : Parm ("N"one, "C"ntl, "E"rror/Resp)
0350 *                  Byte 3   : RecB ("N"one, "A"ccess, "U"pdate)
0360 *
0370 *
0380 *  Format Buff: The format buffer is an optional buffer that may be used
0390 *               to convey the definition of the parameters being passed
0400 *               in the record buffer. The syntax must be consistent with
0410 *               that of a format buffer for a normal command, or be set
0420 *               to "." if it not to be used.
0430 *
0440 *               The field names used in the format buffer should
0450 *               normally be meaningful so that the stored procedure can
0460 *               get the values of each parameter from the record buffer
0470 *               extraction (STPRBE) routine. Length must be used if the
0480 *               stored procedure routine does not provide one.
0490 *               Alternatively, if the field names correspond to the
0500 *               actual file number specified in the ACB, then the STPRBE
0510 *               routine will be able to determine the length of the
0520 *               field/parameter.
0530 *
0540 *               When issuing stored procedures across platforms, it is
0550 *               essential to also specify the field type of each
0560 *               parameter; i.e., "A" - alphanumeric, "B" - binary, "U"
0570 *               - unpacked etc.
0580 *
0590 *
0600 *  Record Buff: The record buffer is available for passing any
0610 *               parameters from the caller to the stored procedure
0620 *               and(or) from the stored procedure to the caller.
0630 *               The layout/DSECT of the record buffer must be
0640 *               coordinated between the caller and the actual stored
0650 *               procedure routine itself.
0660 *
0670 *               The record buffer is available for participating
0680 *               and non-participating (sync) type requests via the
0690 *               the record buffer extraction (STPRBE) routine, only.
0700 *
0710 *               Determination of the record buffer being for access or
0720 *               update is specified by the caller via the additions 3
0730 *               field (see above).
0740 *
0750 ************************************************************************
0760 DEFINE DATA PARAMETER
0770  01  REQ-TYPE  (A1)              /* Optional request ID type
0780  01  P-PROC    (A8)              /* Procedure name
```

```
0790  01  P-PARM1 (A100)              /* Single parameter
0800  01  P-MSG    (A72)              /* Message corresponding to the RESP
0810  01  RESP      (N4)              /* Response code of proc. request
0820            LOCAL USING STPLCB
0830            LOCAL
0840  01  FB    (A16) INIT<'AA,100,A.'>
0850  01  ET-CNT         (P3)
0860 END-DEFINE
0870 FORMAT PS=0
0880 *
0890 RESET CB                         /* Clear the ACB
0900 MOVE 'STP'   TO CB-CID           /* Command ID
0910 MOVE 'PC'    TO CB-CMD           /* Command code
0920 MOVE 222     TO CB-DBID          /* Database ID
0930 MOVE 12      TO CB-FNR           /* Default to TRG file number
0940 MOVE 9       TO CB-FBL           /* FB length
0950 MOVE 100     TO CB-RBL           /* RB length
0960 IF P-PROC = ' '                  /* Did we get a procedure name?
0970   DO
0980     MOVE 1 TO RESP
0990     MOVE 'Invalid Procedure Name specified' TO P-MSG
1000     ESCAPE ROUTINE
1010   DOEND
1020 MOVE P-PROC     TO CB-ADD1       /* Stored procedure name
1030 MOVE 'NCA     ' TO CB-ADD3       /* Options: N - Sync (non-partic)
1040 *                                /*          C - Control parms
1050 *                                /*          A - RecBuff for access
1060 *
1070 CALL 'CMADA' USING CB FB P-PARM1 /* Invoke the stored procedure
1080 *
1090 MOVE CB-RSP TO RESP
1100 MOVE 'Check Response code returned for this request' TO P-MSG
1110 * PRINT (CD=YE) 'Resp ..' (YEI) CB-RSP(EM=HH) 'Add2' CB-ADD2(EM=H(8))
1120 *              'Add4' CB-ADD4(EM=H(8))
1130 *
1140 END
```

STPLNK02

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *  Subprogram : STORPROC/STPLNK02
0040 *  Author     : Adabas Development
0050 *
0060 *  Function   : Sample routine 02 to invoke a stored procedure
0070 *               This example expects up to 5 different variable-length
0080 *               parameters. The length of each parameter is specified
0090 *               as the first two bytes of each parameter. Length is
0100 *               inclusive of the two-byte length itself.
0110 *  Remarks    : This routine will set up the buffers and issue the call
```

```
0120 *               to invoke a stored procedure routine directly.
0130 *               Once processing is completed, control is returned to
0140 *               the caller.
0150 *               Parameter RESP must be set to zero if processing is
0160 *               successful.
0170 *
0180 *  Parameters : The following fields in the ACB must be set up to invoke
0190 *               the stored procedure request.
0200 *
0210 *               Command Code: 'PC'
0220 *               Command ID  : 'STPx' - where x is any value
0230 *               Database ID : Database of the respective trigger file
0240 *                             Set CB-DBID for a one-byte DBID
0250 *                             Set CB-RSP  for a two-byte DBID with
0260 *                               CB-CALL-TYPE set to H'30'
0270 *               File Number : Set to the trigger file number of the
0280 *                             target database (normal one-byte versus
0290 *                             two-byte FNRs is applicable) by default
0300 *                             or any other file used in conjunction with
0310 *                             the format buffer.
0320 *               FB Length   : Length of the format buffer
0330 *               RB Length   : Length of the record buffer
0340 *               Additions 1 : Name of the stored procedure
0350 *               Additions 3 :
0360 *                  Byte 1   : Type ("A"sync, "P"art, "N"on-Partic)
0370 *                  Byte 2   : Parm ("N"one, "C"ntl, "E"rror/Resp)
0380 *                  Byte 3   : RecB ("N"one, "A"ccess, "U"pdate)
0390 *
0400 *
0410 *  Format Buff: The format buffer is an optional buffer that may be used
0420 *               to convey the definition of the parameters being passed
0430 *               in the record buffer. The syntax must be consistent with
0440 *               that of a format buffer for a normal command, or be set
0450 *               to "." if it not to be used.
0460 *
0470 *               The field names used in the format buffer should
0480 *               normally be meaningful so that the stored procedure can
0490 *               get the values of each parameter via the record buffer
0500 *               extraction (STPRBE) routine. Length must be used if the
0510 *               stored procedure routine does not provide one.
0520 *               Alternatively, if the field names correspond to the
0530 *               actual file number specified in the ACB, then the STPRBE
0540 *               routine will be able to determine the length of the
0550 *               field/parameter.
0560 *
0570 *               When issuing stored procedures across platforms, it is
0580 *               essential to also specify the field type of each
0590 *               parameter; i.e., "A" - alphanumeric, "B" - binary, "U"
0600 *               - unpacked etc.
0610 *
0620 *
0630 *  Record Buff: The record buffer is available for passing any
```

```
0640 *               parameters from the caller to the stored procedure
0650 *               and(or) from the stored procedure to the caller.
0660 *               The layout/DSECT of the record buffer must be
0670 *               coordinated between the caller and the actual stored
0680 *               procedure routine itself.
0690 *
0700 *               The record buffer is available for participating
0710 *               and non-participating (sync) type requests via the
0720 *               the record buffer extraction (STPRBE) routine, only.
0730 *
0740 *               Determination of the record buffer being for access or
0750 *               update is specified by the caller via the additions 3
0760 *               field (see above).
0770 *
0780 ************************************************************************
0790 DEFINE DATA PARAMETER
0800  01  REQ-TYPE   (A1)
0810  01  P-PROC     (A8)              /* Procedure name
0820  01  P-OPTIONS  (A8)
0830  01  REDEFINE P-OPTIONS
0840    02  P-TYPE   (A1)             /* Async versus sync procedure
0850    02  P-PARMS  (A1)             /* Parm type for procedure
0860    02  P-RECB   (A1)             /* Rec buffer access
0870  01  P-PARM1(A1/1:V)             /* Variable-length parameter
0880 *                                     first 2 bytes set to incl. length
0890  01  P-PARM2(A1/1:V)             /* Variable-length parameter 2
0900  01  P-PARM3(A1/1:V)             /* Variable-length parameter 3
0910  01  P-PARM4(A1/1:V)             /* Variable-length parameter 4
0920  01  P-PARM5(A1/1:V)             /* Variable-length parameter 5
0930  01  P-MSG     (A72)             /* Message corresponding to the RESP
0940  01  RESP       (N4)             /* Response code of proc request
0950            LOCAL USING STPLCB
0960            LOCAL
0970  01  SUB        (I2)
0980  01  SUB1       (I2)
0990  01  SUB2       (I2)
1000  01  SUB3       (I2)
1010  01  SUB4       (I2)
1020  01  FB        (A48)
1030  01  REDEFINE FB
1040    02  FB-FIELD  (8)
1050      03  FB-FLD (A3)
1060      03  FB-LEN (N3)
1070  01  RB    (A1/1000)            /* Max length for all parms
1080  01  W-ADD3     (A8)
1090  01  REDEFINE W-ADD3
1100    02  W-TYPE   (A1)
1110    02  W-PARMS  (A1)
1120    02  W-RECB   (A1)
1130  01  #LENGTH    (B2)
1140  01  REDEFINE #LENGTH
1150    02  #LENG  (A1/2)
```

```
1160  01  W-LENG   (P5/5)
1170 END-DEFINE
1180 FORMAT PS=0
1190 *
1200 * In this example, we will say that each parameter has an individual
1210 * maximum length of 200; however, the limit may be established as a
1220 * total of all parameters. Since our max. record buffer is 1000 then the
1230 * maximum of all parameters cannot exceed 1000. This may be changed as
1240 * required by the user.
1250 *
1260 FOR SUB1 1 5                        /* Get all the parameter lengths
1270   DECIDE ON FIRST VALUE OF SUB1
1280     VALUE 1  MOVE P-PARM1(1:2) TO #LENG(1:2) /* Get Parm1 length
1290              IF #LENGTH < 3 /* Min length with inclusive length
1300                DO
1310                  MOVE 16 TO RESP
1320                  MOVE 'Invalid Length for Parameter 1. Must be 3-200'
1330                       TO P-MSG
1340                  ESCAPE ROUTINE
1350                DOEND
1360     VALUE 2  MOVE P-PARM2(1:2) TO #LENG(1:2) /* Get Parm2 length
1370     VALUE 3  MOVE P-PARM3(1:2) TO #LENG(1:2) /* Get Parm3 length
1380    VALUE 4  MOVE P-PARM5(1:2) TO #LENG(1:2) /* Get Parm4 length
1390    VALUE 5  MOVE P-PARM1(1:2) TO #LENG(1:2) /* Get Parm5 length
1400    ANY      IF #LENGTH = H'4040'  /* Is length Blanks?
1410               RESET #LENGTH       /*   yes, then treat as dummy parm
1420             MOVE #LENGTH TO W-LENG(SUB1)
1430             IF W-LENG(SUB1) > 202 /* For our example, we limit the length
1440               DO
1450                 MOVE 4 TO RESP
1460                 MOVE SUB1 TO FB-LEN(SUB1)
1470                 COMPRESS 'Invalid Length for Parameter' FB-LEN(SUB1)
1480                          '. Max is 200.' INTO P-MSG
1490                 ESCAPE ROUTINE    /* Terminate processing with error
1500               DOEND
1510             SUBTRACT 2 FROM W-LENG(SUB1) /* ACTUAL parm length
1520    NONE     IGNORE
1530  END-DECIDE
1540 CLOSE LOOP (1260)
1550 *
1560 IF P-PROC = ' '                    /* Did we get a procedure name?
1570   DO
1580     MOVE 1 TO RESP
1590     MOVE 'Invalid Procedure Name specified' TO P-MSG
1600     ESCAPE ROUTINE
1610   DOEND
1620 IF NOT (P-TYPE = 'A' OR= 'N' OR= 'P' OR= ' ')
1630   DO                               /* Async, participating, non-partic.
1640     MOVE 2 TO RESP
1650     MOVE 'Proc Type must be A, N, P or " "' TO P-MSG
1660     ESCAPE ROUTINE
1670   DOEND
```

```
1680 IF NOT (P-PARMS = 'C' OR= 'E' OR= 'N' OR= ' ')
1690   DO                                /* Cntrl, Error/Resp, None
1700     MOVE 3 TO RESP
1710     MOVE 'Parameter Type must be C, E, N or " "' TO P-MSG
1720     ESCAPE ROUTINE
1730   DOEND
1740 IF NOT (P-RECB = 'A' OR= 'N' OR= 'U' OR= ' ')
1750   DO                                /* Access, None, Update
1760     MOVE 3 TO RESP
1770     MOVE 'Parameter access must be Access, None or Update' TO P-MSG
1780     ESCAPE ROUTINE
1790   DOEND
1800 *
1810 * Next we merge all the passed parameters into a single contiguous
1820 * buffer which will be used as the record buffer for the call. The
1830 * format buffer will also be set up to indicate the 'structure' of the
1840 * record buffer for use by the invoked procedure.
1850 *
1860 MOVE 1 TO SUB
1870 *
1880 FOR SUB3 1 5                       /* Step through all parameters
1890   IF W-LENG(SUB3) < 3              /* Check min. length of a parameter
1900     DO
1910       MOVE '.' TO FB-FLD(SUB3)
1920       ESCAPE BOTTOM                /* None, so assume we have all parms
1930     DOEND
1940   MOVE W-LENG(SUB3) TO SUB1
1950   ADD SUB1 TO SUB2
1960   DECIDE ON FIRST VALUE OF SUB1    /* Move parms into the RB
1970     VALUE 1 MOVE 'P1,' TO FB-FLD(1)
1980             MOVE P-PARM1 (3:SUB1) TO RB(SUB:SUB2)
1990     VALUE 2 MOVE 'P2,' TO FB-FLD(2)
2000             MOVE P-PARM2 (3:SUB1) TO RB(SUB:SUB2)
2010     VALUE 3 MOVE 'P3,' TO FB-FLD(3)
2020             MOVE P-PARM3 (3:SUB1) TO RB(SUB:SUB2)
2030     VALUE 4 MOVE 'P4,' TO FB-FLD(4)
2040             MOVE P-PARM4 (3:SUB1) TO RB(SUB:SUB2)
2050     VALUE 5 MOVE 'P5,' TO FB-FLD(5)
2060             MOVE P-PARM5 (3:SUB1) TO RB(SUB:SUB2)
2070     ANY     ADD SUB1 TO SUB
2080             MOVE SUB1 TO FB-LEN(SUB3)
2090     NONE    IGNORE
2100   END-DECIDE
2110 *
2120 CLOSE LOOP (1880)
2130 *
2140 * Now we start setting up the CB and do some additional validation.
2150 * When moving in the procedure options, we allow for defaults.
2160 *
2170 RESET CB                          /* Clear the ACB
2180 MOVE 'STP'   TO CB-CID            /* Command ID
2190 MOVE 'PC'    TO CB-CMD            /* Command code
```

```
2200 MOVE 77      TO CB-DBID           /* Database ID
2210 MOVE 22      TO CB-FNR            /* File number
2220 MOVE 48      TO CB-FBL            /* FB length
2230 MOVE 1000    TO CB-RBL            /* RB length
2240 MOVE P-PROC  TO CB-ADD1           /* Stored procedure name
2250 *
2260 MOVE 'A' TO W-TYPE                 /* Set the default options
2270 MOVE 'C' TO W-PARMS
2280 MOVE 'N' TO W-RECB
2290 IF NOT (P-TYPE = ' ')             /* Should we default to Async?
2300   MOVE P-TYPE TO W-TYPE
2310 IF NOT (P-PARMS = ' ')            /* Should we default to Contrl?
2320   MOVE P-PARMS TO W-PARMS
2330 IF NOT (P-RECB = ' ')             /* Should we default to None?
2340   MOVE P-RECB TO W-RECB
2350 MOVE W-ADD3  TO CB-ADD3           /* Options for request
2360 *
2370 CALL 'CMADA' USING CB FB RB(1)   /* Invoke the stored procedure
2380 *
2390 IF CB-RSP NE 0
2400   DO
2410     PRINT (CD=YE) 'Resp ..' (YEI) CB-RSP(EM=HH) 'Add2' CB-ADD2(EM=H(4))
2420                   'Add3' CB-ADD3(EM=H(8)) 'Add4' CB-ADD4(EM=H(8))
2430     ESCAPE ROUTINE
2440   DOEND
2450 *
2460 * Now we need to restore the parameters back into the user's area,
2470 * in case the data was modified. This can happen only if the record
2480 * buffer was modifiable; i.e., P-RECB was set to 'U'.
2490 *
2500 IF  CB-RSP = 0                    /* Was everything okay
2510 AND P-RECB = 'U'                  /* Update: Parms may have been updated
2520   DO
2530     MOVE 1 TO SUB
2540     RESET SUB2
2550     FOR SUB1 1 5
2560       ADD  W-LENG(SUB1) TO SUB2
2570       MOVE W-LENG(SUB1) TO SUB3
2580       DECIDE ON FIRST VALUE OF SUB1 /* Restore parm from RB
2590         VALUE 1   ASSIGN P-PARM1 (3:SUB3) = RB(SUB:SUB2)
2600         VALUE 2   ASSIGN P-PARM2 (3:SUB3) = RB(SUB:SUB2)
2610         VALUE 3   ASSIGN P-PARM3 (3:SUB3) = RB(SUB:SUB2)
2620         VALUE 4   ASSIGN P-PARM4 (3:SUB3) = RB(SUB:SUB2)
2630         VALUE 5   ASSIGN P-PARM5 (3:SUB3) = RB(SUB:SUB2)
2640         ANY       ADD W-LENG(SUB1) TO SUB /* Get next position
2650         NONE      IGNORE
2660       END-DECIDE
2670     CLOSE LOOP(2550)
2680   DOEND
2690 *
2700 END
```

STPLNK03

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *  Subprogram : STORPROC/STPLNK03
0040 *  Author     : Adabas Development
0050 *
0060 *  Function   : Sample routine 03 to invoke a stored procedure
0070 *               This example expects up to five different variable-length
0080 *               parameters. Parameter lengths are passed as extra
0090 *               parameters.
0100 *  Remarks    : This routine will set up the buffers and issue the call
0110 *               to invoke a stored procedure routine directly.
0120 *               Once processing is completed, control is returned to
0130 *               the caller.
0140 *               Parameter RESP must be set to zero if processing is
0150 *               successful.
0160 *
0170 *  Parameters : The following fields in the ACB must be set up to invoke
0180 *               the stored procedure request.
0190 *
0200 *               Command Code: 'PC'
0210 *               Command ID  : 'STPx' - where x is any value
0220 *               Database ID : Database of the respective trigger file
0230 *                             Set CB-DBID for a one-byte DBID
0240 *                             Set CB-RSP  for a two-byte DBID with
0250 *                               CB-CALL-TYPE set to H'30'
0260 *               File Number : Set to the trigger file number of the
0270 *                             target database (normal one-byte versus
0280 *                             two-byte FNRs is applicable) by default
0290 *                             or any other file used in conjunction with
0300 *                             the format buffer.
0310 *               FB Length   : Length of the format buffer
0320 *               RB Length   : Length of the record buffer
0330 *               Additions 1 : Name of the stored procedure
0340 *               Additions 3 :
0350 *                  Byte 1   : Type ("A"sync, "P"art, "N"on-Partic)
0360 *                  Byte 2   : Parm ("N"one, "C"ntl, "E"rror/Resp)
0370 *                  Byte 3   : RecB ("N"one, "A"ccess, "U"pdate)
0380 *
0390 *
0400 *  Format Buff: The format buffer is an optional buffer that may be used
0410 *               to convey the definition of the parameters being passed
0420 *               in the record buffer. The syntax must be consistent with
0430 *               that of a format buffer for a normal command, or be set
0440 *               to "." if it not to be used.
0450 *
0460 *               The field names used in the format buffer should
0470 *               normally be meaningful so that the stored procedure can
0480 *               obtain the values of each parameter via the record buffer
```

```
0490 *               extraction (STPRBE) routine. Length must be used if the
0500 *               stored procedure routine does not provide one.
0510 *               Alternatively, if the field names correspond to the
0520 *               actual file number specified in the ACB, then the STPRBE
0530 *               routine will be able to determine the length of the
0540 *               field/parameter.
0550 *
0560 *               When issuing stored procedures across platforms, it is
0570 *               essential to also specify the field type of each
0580 *               parameter; i.e., "A" - alphanumeric, "B" - binary, "U"
0590 *               - unpacked etc.
0600 *
0610 *
0620 *  Record Buff: The record buffer is available for passing any
0630 *               parameters from the caller to the stored procedure
0640 *               and(or) from the stored procedure to the caller.
0650 *               The layout/DSECT of the record buffer must be
0660 *               coordinated between the caller and the actual stored
0670 *               procedure routine itself.
0680 *
0690 *               The record buffer will be available for participating
0700 *               and non-participating (sync) type requests via the
0710 *               the record buffer extraction (STPRBE) routine, only.
0720 *
0730 *               Determination of the record buffer being for access or
0740 *               update is specified by the caller via the additions 3
0750 *               field (see above).
0760 *
0770 ************************************************************************
0780 DEFINE DATA PARAMETER
0790  01  REQ-TYPE   (A1)
0800  01  P-PROC     (A8)             /* Procedure name
0810  01  P-OPTIONS  (A8)
0820  01  REDEFINE P-OPTIONS
0830    02  P-TYPE   (A1)             /* Async versus sync procedure
0840    02  P-PARMS  (A1)             /* Parm type for procedure
0850    02  P-RECB   (A1)             /* Rec buffer access
0860  01  P-LEN1     (P3)             /* Length of Parm1
0870  01  P-PARM1(A1/1:V)             /* Variable-length parameter 1
0880  01  P-LEN2     (P3)             /* Length of Parm2
0890  01  P-PARM2(A1/1:V)             /* Variable-length parameter 2
0900  01  P-LEN3     (P3)             /* Length of Parm3
0910  01  P-PARM3(A1/1:V)             /* Variable-length parameter 3
0920  01  P-LEN4     (P3)             /* Length of Parm4
0930  01  P-PARM4(A1/1:V)             /* Variable-length parameter 4
0940  01  P-LEN5     (P3)             /* Length of Parm5
0950  01  P-PARM5(A1/1:V)             /* Variable-length parameter 5
0960  01  P-MSG     (A72)             /* Message corresponding to the RESP
0970  01  RESP       (N4)             /* Response code of proc request
0980            LOCAL USING STPLCB
0990            LOCAL
1000  01  SUB        (I2)
```

```
1010  01  SUB1       (I2)
1020  01  SUB2       (I2)
1030  01  SUB3       (I2)
1040  01  FB        (A64)
1050  01  REDEFINE FB
1060    02  FB-FIELD  (8)
1070      03  FB-FLD (A3)
1080      03  FB-LEN (N3)
1090      03  FB-COMM(A1)
1100  01  RB    (A1/1000)             /* Max length for all parms
1110  01  W-ADD3     (A8)
1120  01  REDEFINE W-ADD3
1130    02  W-TYPE   (A1)
1140    02  W-PARMS  (A1)
1150    02  W-RECB   (A1)
1160  01  #LENGTH    (B2)
1170  01  REDEFINE #LENGTH
1180    02  #LENG  (A1/2)
1190  01  W-LENG   (P3/5)
1200 END-DEFINE
1210 FORMAT PS=0
1220 *
1230 MOVE P-LEN1 TO W-LENG(1)
1240 MOVE P-LEN2 TO W-LENG(2)
1250 MOVE P-LEN3 TO W-LENG(3)
1260 MOVE P-LEN4 TO W-LENG(4)
1270 MOVE P-LEN5 TO W-LENG(5)
1280 *
1290 * In this example, we will say that each parameter has an individual
1300 * maximum length of 200; however, the limit may be established as a
1310 * total of all parameters. Since our max. record buffer is 1000, the
1320 * maximum of all parameters cannot exceed 1000. This may be changed as
1330 * required by the user.
1340 *
1350 FOR SUB1 1 5                       /* Validate all parameter lengths
1360  IF W-LENG(SUB1) > 16448           /* Does length contain X'4040'
1370    RESET W-LENG(SUB1)              /*   yes, then must be dummy parm
1380  IF W-LENG(SUB1) > 200             /* For our example we limit the length
1390    DO
1400      MOVE 15   TO RESP
1410      MOVE SUB1 TO FB-LEN(SUB1)
1420      COMPRESS 'Invalid Length for Parameter' FB-LEN(SUB1)
1430               '. Max is 200.' INTO P-MSG
1440      ESCAPE ROUTINE                /* Terminate processing with error
1450    DOEND
1460 CLOSE LOOP
1470 *
1480 * Now we validate the parameters, as required. Of course, these may
1490 * be changed as per the user's requirement and may vary from one stored
1500 * procedure link routine to another.
1510 *
1520 IF P-PROC = ' '                    /* Did we get a procedure name?
```

```
1530  DO
1540    MOVE 1 TO RESP
1550    MOVE 'Invalid Procedure Name specified' TO P-MSG
1560    ESCAPE ROUTINE
1570  DOEND
1580 IF NOT (P-TYPE = 'A' OR= 'N' OR= 'P' OR= ' ')
1590  DO                              /* Async, Participating, Non-Partic
1600    MOVE 2 TO RESP
1610    MOVE 'Proc Type must be A, N, P or " "' TO P-MSG
1620    ESCAPE ROUTINE
1630  DOEND
1640 IF NOT (P-PARMS = 'C' OR= 'E' OR= 'N' OR= ' ')
1650  DO                              /* Cntrl, Error/Resp, None
1660    MOVE 3 TO RESP
1670    MOVE 'Parameter Type must be C, E, N or " "' TO P-MSG
1680    ESCAPE ROUTINE
1690  DOEND
1700 IF NOT (P-RECB = 'A' OR= 'N' OR= 'U' OR= ' ')
1710  DO                              /* Access, None, Update
1720    MOVE 3 TO RESP
1730    MOVE 'Parameter access must be Access, None or Update' TO P-MSG
1740    ESCAPE ROUTINE
1750  DOEND
1760 IF P-LEN1 < 3                     /* Min. length with inclusive length
1770  DO                              /* Anything less indicates no parm
1780    MOVE 4 TO RESP
1790    MOVE 'First Parameter MUST be valid. Length must be
3-200' TO P-MSG
1800    ESCAPE ROUTINE
1810  DOEND
1820 *
1830 * Next we merge all the passed parameters into a single contiguous
1840 * buffer which will be used as the record buffer for the call. The
1850 * format buffer will also be set up to indicate the 'structure' of the
1860 * record buffer for use by the invoked procedure.
1870 *
1880 MOVE 1 TO SUB
1890 RESET SUB2
1900 *
1910 FOR SUB1 1 5                        /* Step through all parameters
1920  IF W-LENG(SUB1) < 3               /* Check min. length of a parameter
1930    DO
1940      MOVE '.' TO FB-FLD(SUB1)
1950      ESCAPE BOTTOM                 /* None, so assume we have all parms
1960    DOEND
1970  ADD W-LENG(SUB1)  TO SUB2         /* Get end position
1980  MOVE W-LENG(SUB1) TO SUB3         /* Set index for MOVE statement
1990  DECIDE ON FIRST VALUE OF SUB1     /* Move next parm into the RB
2000     VALUE 1   MOVE P-PARM1 (1:SUB3) TO RB(SUB:SUB2)
2010               MOVE 'P1,' TO FB-FLD(1)
2020     VALUE 2   MOVE P-PARM2 (1:SUB3) TO RB(SUB:SUB2)
2030               MOVE ','   TO FB-COMM(SUB1 - 1)
```

```
2040               MOVE 'P2,' TO FB-FLD(2)
2050     VALUE 3   MOVE P-PARM3 (1:SUB3) TO RB(SUB:SUB2)
2060               MOVE ','   TO FB-COMM(SUB1 - 1)
2070               MOVE 'P3,' TO FB-FLD(3)
2080     VALUE 4   MOVE P-PARM4 (1:SUB3) TO RB(SUB:SUB2)
2090               MOVE ','   TO FB-COMM(SUB1 - 1)
2100               MOVE 'P4,' TO FB-FLD(4)
2110     VALUE 5   MOVE P-PARM5 (1:SUB3) TO RB(SUB:SUB2)
2120               MOVE ','   TO FB-COMM(SUB1 - 1)
2130               MOVE 'P5,' TO FB-FLD(5)
2140               MOVE '.'   TO FB-COMM(5)
2150     ANY       ADD W-LENG(SUB1) TO SUB  /* Get new position
2160               MOVE W-LENG(SUB1) TO FB-LEN(SUB1)
2170     NONE      IGNORE
2180  END-DECIDE
2190 *
2200 CLOSE LOOP
2210 *
2220 * Now we set up the CB for the actual stored procedure call.
2230 *
2240 RESET CB                          /* Clear the ACB
2250 MOVE 'STP'   TO CB-CID            /* Command ID
2260 MOVE 'PC'    TO CB-CMD            /* Command Code
2270 MOVE 77      TO CB-DBID           /* Database ID
2280 MOVE 22      TO CB-FNR            /* File Number
2290 MOVE 64      TO CB-FBL            /* FB Length
2300 MOVE 1000    TO CB-RBL            /* RB length
2310 MOVE P-PROC  TO CB-ADD1           /* Stored procedure name
2320 *
2330 * If any options were not passed, we use a pre-specified default.
2340 *
2350 MOVE 'A' TO W-TYPE                 /* Set the default options
2360 MOVE 'C' TO W-PARMS
2370 MOVE 'N' TO W-RECB
2380 IF NOT (P-TYPE = ' ')             /* Should we default to Async?
2390  MOVE P-TYPE TO W-TYPE
2400 IF NOT (P-PARMS = ' ')            /* Should we default to Contrl?
2410  MOVE P-PARMS TO W-PARMS
2420 IF NOT (P-RECB = ' ')             /* Should we default to None?
2430  MOVE P-RECB TO W-RECB
2440 MOVE W-ADD3  TO CB-ADD3           /* Options for request 2450 *
2460 CALL 'CMADA' USING CB FB RB(1)   /* Invoke the stored procedure
2470 *
2480 IF CB-RSP NE 0
2490  DO
2500    PRINT (CD=YE) 'Resp ..' (YEI) CB-RSP(EM=HH) 'Add2' CB-ADD2(EM=H(4))
2510                  'Add3' CB-ADD3(EM=H(8)) 'Add4' CB-ADD4(EM=H(8))
2520    ESCAPE ROUTINE
2530  DOEND
2540 *
2550 * Now we need to restore the parameters back into the user's area,
2560 * in case the data was modified. This can happen only if the record
```

```
2570 * buffer was modifiable; i.e., P-RECB was set to 'U'.
2580 *
2590 IF  CB-RSP = 0                    /* Was everything okay
2600 AND P-RECB = 'U'                 /* Update: Parms may have been updated
2610  DO
2620    MOVE 1 TO SUB
2630    RESET SUB2
2640    FOR SUB1 1 5
2650      ADD  W-LENG(SUB1) TO SUB2
2660      MOVE W-LENG(SUB1) TO SUB3
2670      DECIDE ON FIRST VALUE OF SUB1 /* Restore parm from RB
2680        VALUE 1   ASSIGN P-PARM1 (1:SUB3) = RB(SUB:SUB2)
2690        VALUE 2   ASSIGN P-PARM2 (1:SUB3) = RB(SUB:SUB2)
2700        VALUE 3   ASSIGN P-PARM3 (1:SUB3) = RB(SUB:SUB2)
2710        VALUE 4   ASSIGN P-PARM4 (1:SUB3) = RB(SUB:SUB2)
2720        VALUE 5   ASSIGN P-PARM5 (1:SUB3) = RB(SUB:SUB2)
2730        ANY       ADD W-LENG(SUB1) TO SUB /* Get next position
2740        NONE      IGNORE
2750      END-DECIDE
2760    CLOSE LOOP(2640)
2770  DOEND
2780 *
2790 END
```

VI Trigger Maintenance

8 Trigger Maintenance

Note: The Trigger Maintenance subsystem is available only when the Adabas Online System (AOS) add-on product is installed. It is not available with the AOS demo version.

The Trigger Maintenance facility is a Natural application that allows you to

- add, delete, modify, and list trigger definitions;
- generate, modify, display, and delete file-field tables;
- display and modify profile information for the definition of the run-time triggers system;
- monitor trigger operations and permanently or temporarily activate or deactivate triggers.

To start Trigger Maintenance

- Select "Trigger Maintenance" from the Adabas Online System (AOS) main menu or enter `MENU` at the Natural NEXT prompt in library SYSTRG.

  The Trigger Maintenance main menu appears:

```
HH:MM:SS             ***** TRIGGER MAINTENANCE *****       YYYY-MM-DD
User USR01                     - Main Menu -               DBnr 105

                    Code  Function
                    ----  ---------------------------------
                      A   Administrator Functions
                      F   File-Field Table Definitions
                      R   Procedure Reports
                      T   Create/Modify Trigger Definitions
                      ?   Help
                      .   Exit
                    ----  ---------------------------------

               Code ...


Command ==>
Enter--PF1--PF2--PF3--PF4---PF5---PF6---PF7--PF8--PF9--PF10--PF11--PF12
       Help      Exit Field Trigr Admin Procs          FTRG  FDIC  Canc
```

  The main menu functions are described briefly in the following table:

Code	Function	Description
A	Administrator Functions	Display or modify the profile information, i.e., the run-time parameter settings of the triggers and stored procedures. Obtain Natural subsystem information. Monitor subsystem activity and trigger activity.
F	File-Field Table Definitions	Display, generate, modify, delete, or select a file-field table. View the origin of the file-field table.
R	Procedure Reports	Display a list of procedures for the defined triggers, sorted by file or by name.
T	Create/Modify Trigger Definitions	Add, display, modify, delete, or select a trigger definition.
?	Help	Display help information about the Trigger Maintenance facility.
.	Exit	Exit to the Natural NEXT screen. Enter `TRIGGERS` to return to the main menu.

Note: The database ID and file number are required by the Trigger Maintenance facility. If these values were not specified by default using the `NATPARM LFILE` parameters or the NTLFILE macro, they must be entered before the main menu functions can be used. You can enter these values by pressing PF10 on the main menu; otherwise, the system automatically prompts you for the information.

Overview

The Trigger Maintenance facility comprises a set of menus and submenus that lead to data screens and pop-up windows. Each menu contains a list of functions and function codes, a group of input fields, a command line, a set of PF keys, and a message area. Adabas Online System (AOS) must be installed in the same environment.

Wildcard Notation

Wildcard notation is allowed wherever possible. For example, most menus require that you specify a file name. If you do not know the file name, you can enter a wildcard character instead. A range of values will appear in a pop-up window, allowing you to select an item or entry from the range.

The term "wildcard" refers to the use of a special character as follows:

Char.	Example	Select any name . .
*	PERS*	beginning with "PERS"
>	PERS>	with a value greater than "PERS"
<	PERS<	with a value less than "PERS"

Input Fields

Each menu in the Trigger Maintenance facility includes the following input fields:

Field	Entry
Code	Enter the code for the function. For example, codes on the main menu are A — administrator functions F — file-field table definitions R — procedure reports T — create/modify trigger definitions Depending on the function, a submenu, a pop-up, or a data screen appears.
File Name	Enter the file name, or enter a wildcard to display a selection list of file names and numbers (for example, the entry "a*" returns a list of all files with names that begin with the letter "a").
File Number	Enter the file number. When generating a file-field table from an FDT, you may enter "99999" instead of the actual file number to get a list of valid files.

Other input fields vary. For example:

- The File-Field Table Definitions Menu includes a Generation Type input field. Generation Type identifies the source from which the file-field table is to be generated (Natural DDM, Adabas FDT, or Predict Adabas file definition).
- The Trigger Definitions Menu includes the Field Name and Command Type input fields. File Name, Command Type, and Field Name comprise the selection criteria that you specify when you create the trigger definition.

Messages

The message area at the top or bottom of the screen is used to display one-line error or action messages. Each message includes a message number and a brief explanation. For more information, see the *Adabas Messages and Codes* documentation.

Commands

The following table describes the commands that can be entered at the command line. Not all commands apply to all screens.

Commands are not case-sensitive and can be entered in upper-, lower-, or upper- and lower-case. Commands are converted to upper-case before being processed.

The underlined portion of the command indicates the minimum abbreviation allowed.

Command	Display...
ACTIVITY or DISPLAY ACTIVITY	the Current Trigger Activity screen, which contains information about currently executing triggers.
DISPLAY PROFILE	the Display Profile Information screen, which contains the Trigger Maintenance facility profile.
DISPLAY TASKS or DISPLAY SYSTEMS	the Subsystem Activity screen, which contains information about currently executing Natural subsystems.
MENU	the Main Menu.
EXIT, QUIT, or . (period)	the previous screen.
FIELD	the File-Field Table Definitions Menu.
TRIGGER	the Trigger Definitions Menu.
ADMIN	the Administrator Menu.
PROFILE	the Profile Information Menu.
SET FILE	the database and file assignments for the current trigger file. You can enter a different database and file, which must be a valid trigger file.
SET FDIC	the database and file assignments for the current Predict file. You can enter a different Predict system file.
FORWARD or +	the next screen.
TOP, or --	from the top (beginning); for example, the start of a file-field table list.
BACK, or -	the previous screen.

PF Keys

The following table describes the standard PF keys.

Key	Label	Description	Menu or Screen
PF1	Help	Display information about the current function.	All screens
PF2	Menu	Return to the Main Menu.	All screens
PF3	Exit	Return to the previous screen. In most cases, if the update option has not been selected, this results in any updates being ignored.	All screens
PF4	Field	Display the File-Field Table Definitions Menu.	Main, Profile Information, Procedure Reporting, and Trigger Definitions menus
PF5	Trigr	Display the Trigger Definitions Menu.	Main, File-Field Table Definitions, Profile Information, and Procedure Reporting menus; Define Trigger Info, Add Function pop-up
PF6	Admin	Display the Administrator Menu.	Main, File-Field Table Definitions, and Trigger Definitions menus
PF7	Procs	Display the Procedure Reporting Menu.	Main Menu
PF10	FTRG	Display the File Assignments for Triggers pop-up, where you can enter a database and file different from the one currently used.	Main Menu and File-Field Table Definitions Menu
PF11	FDIC	Change the setting of the Predict file (for triggers only) from one value to another.	Main Menu and File-Field Table Definitions Menu
PF12	Canc	Cancel the current process and return to the previous screen. If the update option has not been selected, this results in any updates being ignored.	All screens

File-Field Tables

Triggers require access to a file-field table that maps long file names to file numbers and field names to Adabas two-character field identifiers.

The file-field table should contain an entry for

- all fields used in trigger definitions;
- all fields that will be referenced by an access or update command in a triggered procedure;
- all fields that must be queried by the record buffer extraction routine (STPRBE).

File-field tables can be generated from an Adabas FDT, a Natural DDM, or a Predict Adabas file, with the following restrictions:

- Superdescriptors, subdescriptors, hyperdescriptors, and phonetic descriptors are supported for histograms (L9) only.

After the file-field table entries are defined, the external field names and the internal field names and numbers can be correlated, and a trigger can be defined for any of the fields in the file-field table. If, for example, you refer to the EMPLOYEES file and the SALARY field, Adabas will be able to identify this as file 1 and field AS.

Record Buffer Extraction

When the record buffer extraction routine (STPRBE) extracts the value of a field in the record buffer, it must extract the value from the correct position and length. In order to position to the correct place, STPRBE must step past each superfluous field and must therefore know the length of each field.

- If the field length is not explicitly specified in the format buffer, it must be obtained from the file-field table definitions.
- If the length of a field is always found in the format buffer, as in Natural, then there is no need to include the field in the file-field table.

If the field length is not in the format buffer and a definition does not exist on the trigger file, an error will occur. The procedure will be unable to continue processing and will terminate.

Group-Field Table

When the group name is specified in the format buffer, STPRBE uses the group-field table to locate the elementary fields belonging to the group. Each entry in the group-field table contains specific information about an elementary field and its offset within a group. Offsets are maintained for up to seven group levels. Up to 50 elementary fields may have entries in the group-field table for a specific file.

Entries for the group-field table can be generated from either the Generate File-Field Table function or the Modify File-Field Table function.

File-Field Table Definitions Menu

File-field tables are generated and maintained using the File-Field Table Definitions function of the Trigger Maintenance facility. Any combination of fields can be added or subsequently deleted from the file-field table.

To access the File-Field Table Definitions function

- Enter F on the Main Menu.

  The screen that appears is similar to the example shown below:

```
HH:MM:SS            ***** TRIGGER MAINTENANCE *****        YYYY-MM-DD
 User USR01       - File-Field Table Definitions Menu -     DBnr 105

                 Code  Function
                 ----  -----------------------------------
                   D   Display File-Field Table
                   G   Generate File-Field Table
                   M   Modify File-Field Table
                   P   Delete File-Field Table
                   R   Rename File-Field Table
                   S   Select File-Field Table
                   V   View the Origin for File-Field Table
                   ?   Help
                   .   Exit
                 ----  -----------------------------------

     Code ......... _
     File Name .... ____________________
     File Number .. _____
     Gen. Type .... __

 Command ==>
 Enter-PF1--PF2--PF3--PF4--PF5---PF6---PF7---PF8--PF9--PF10--PF11--PF12
       Help Menu Exit      Trigr Admin Procs           FTRG  FDIC  Canc
```

The following table briefly describes each of the functions on the File-Field Table Definitions Menu:

Function	Description
Display File-Field Table	Display the table for a specified file.
Generate File-Field Table	Generate a new table by adding or deleting fields. Requires a generation type code.
Modify File-Field Table	Modify the table by deleting one or more individual fields. A field cannot be deleted if it is used in a trigger definition.
Delete File-Field Table	Delete an entire table. A table cannot be deleted if it contains one or more fields that are used in a trigger definition.
Rename File-Field Table	Rename a table. All fields and triggers for the specified table are renamed.
Select a File-Field Table	Select a table from a selection list.
View the Origin for File-Field Table	Display the FDT, DDM, or Predict file from which the table was generated. Allows you to add selected fields to the table.

Display a File-Field Table

To display a file-field table

- Enter `D` on the File-Field Table Definitions Menu.

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****       YYYY-MM-DD
  User USR01               - Display File-Field Table -         DBnr 105

  File Name ... AUTOMOBILES                                     Fnr ... 4

 Field Name  Long Field Name                    Status  Message
 ----------  --------------------------------  ------  ------------------
     AC      BODY-TYPE                         Triggr
     CA      COLOR                             Triggr
     AA      MAKE                              Active
     FB      MILEAGE                           Triggr
     AB      MODEL                             Active
     BD      WEIGHT                            Active
     DA      YEAR                              Active



                Enter 'F'(Fwd), 'T'(Top), 'B'(Bck), '.'(Exit)
 Command==>
 Enter-PF1---PF2---PF3---PF4--PF5---PF6--PF7--PF8--PF9---PF10---PF11--PF12
       Help  Menu  Exit       Next  --   -    +    Grp   GFld         Canc
```

  - Press ENTER to scroll through the information on this screen.
  - Press PF9 or continue scrolling to display the group table. If a group table has been generated for the file, the table is displayed at the end of the file-field table.
  - Press PF10 to display the group-field table.

Display a Group Table

To display the group table

- Press PF9 (Grp) or scroll to the end on the Display File-Field Table screen:

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****            YYYY-MM-DD
 User USR01                   - Display Group Table -               DBnr 105

 File Name ... AUTOMOBILES                                         Fnr ... 4

 Name Level Type Length      Name Level Type Length      Name Level Type Length
 ---- ----- ---- ------      ---- ----- ---- ------      ---- ----- ---- ------
  AB    1    G       60
  A1    1    G       53
  A2    1    G       21
  AQ    1    P       13
  A3    1    G        4
  AW    1    P       12



                Enter 'F'(Fwd), 'T'(Top), 'B'(Bck), '.'(Exit)
 Command ===>
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       Help  Menu  Exit              --    -     +           GFld  FFT   Canc
```

- Press ENTER to scroll through the information on this screen.
- Press PF10 or continue scrolling to display the group-field table. The group-field table is displayed at the end of the group table.
- Press PF11 to return to the Display File-Field Table screen.

Each group field on the file is listed. A "G" in the Type column represents a simple group field; a "P" represents a periodic (PE) group. Group-field table entries can only be created for fields that are part of a group or a PE group.

The Length column displays

- for a simple group, the total length of all fields included in the group
- for a PE group, the total length for each occurrence.

Display a Group-Field Table

To display the group-field table

- Press PF10 (GFld) on the Display File-Field Table screen;

  Or:

  scroll to the end on the Display Group Table screen

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****          YYY-MM-DD
 User USR01               - Display Group-Field Table -           DBnr 105

 File Name ... AUTOMOBILES                                        Fnr ... 4

 Field              Group Offsets                          Message
----------------   ------------------------------------ --------------------
 2,AS,5,P           AQ(3)
 2,AT,5,P           AQ(8)
 2,AU,2,U           A3(0)
 2,AV,2,U           A3(2)
 2,AX,6,U           AW(0)
 2,AY,6,U           AW(6)



            Enter 'F'(Fwd), 'T' (Top), 'B' (Bck), or '.'(Exit)

 Command ===>
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
```

- Press ENTER to scroll through the information on this screen.
- Press PF9 to display the group table.
- Press PF11 to return to the Display File-Field Table screen.

Fields are shown in ADACMP FNDEF format: level, name, length, format [options]. See the *Adabas Utilities* documentation for more information.

The Group Offsets column names the group in which the field participates and the offset of the field within that group. If a field is a member of more than one group, the additional groups (and the field's offset within) are also listed.

For example, field AS belongs to group AQ and is located at offset 3. Field AT also belongs to group AQ but is located at offset 8.

Modify a File-Field Table

To modify a file-field table

- Enter `M` on the File-Field Table Definitions Menu.

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****       YYYY-MM-DD
User USR01                - Modify File-Field Table -         DBnr 105

 File Name ... AUTOMOBILES                                     Fnr ...4

 Sel  Field Name  Long Field Name                  Status  Message
----  ----------  -------------------------------  ------  --------------
 __       AC      BODY-TYPE                        Triggr
 __       CA      COLOR                            Triggr
 __       AA      MAKE                             Active
 __       FB      MILEAGE                          Triggr
 __       AB      MODEL                            Active
 __       BD      WEIGHT                           Active
 __       DA      YEAR                             Active




Mark fields with 'A' Add, 'D' Delete, 'G' Generate, 'I' Info, or '.' Exit

Command==>
Enter-PF1---PF2---PF3---PF4--PF5---PF6--PF7--PF8--PF9---PF10---PF11--PF12
      Help  Menu  Exit       Next  --    -    +    Grp   GFld         Canc
```

- Press ENTER to scroll through the information on this screen.
- Press PF9 or scroll to the end to display the group table. The group table cannot be modified.
- Press PF10 to modify the group-field table.

On the Modify File-Field Table screen, fields with a "Status" of

- "Triggr" are used in a trigger definition.
- "Active" are included in the file-field table but are not used in any trigger definition.

Modifying Fields

From the Modify File-Field Table screen, you can generate group-field table entries, display field attributes, or delete fields from the file-field table.

To generate group-field table entries

- Mark each field for which an entry is to be generated by typing a `G` in the Sel column next to the field name. Then press ENTER.

  Group-field table entries can only be created for fields that are part of a group or a PE group.

To display the attributes for an individual field

- Mark the field by typing an `I` in the Sel column next to the field name. Then press ENTER.

  The resulting window displays the field name, format, length, type (SP for superdescriptor; SB for subdescriptor), and an indicator if a group-field entry has been generated for the field.

To delete one or more fields

- Mark each field to be deleted by typing a `D` in the Sel column next to the field name. Then press ENTER.

  The following rules apply:

  - You can cancel a field deletion by typing an `A` in the Sel column next to the field name *before* pressing ENTER.
  - Only "Active" fields can be deleted.
  - "Triggr" fields cannot be deleted unless the trigger definition is deleted first.

Modifying the Group-Field Table

To modify the group-field table

- Press PF10 (GFld) on the Modify File-Field Table screen;

  Or:

  scroll to the end on the Display Group Table screen

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****          YYY-MM-DD
User USR01               - Modify Group-Field Table -            DBnr 105

 File Name ... AUTOMOBILES                                       Fnr ... 4

 Sel Field              Group Offsets                          Message
 --- ----------------   ------------------------------------ ---------------
  _   2,AS,5,P           AQ(3)
  _   2,AT,5,P           AQ(8)
  _   2,AU,2,U           A3(0)
  _   2,AV,2,U           A3(2)
  _   2,AX,6,U           AW(0)
  _   2,AY,6,U           AW(6)




            Mark Fields with 'D'(Delete) or '.'(Exit)

 Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Menu  Exit              --    -     +     Grp          FFT   Canc
```

- Press ENTER to scroll through the information on this screen.
- Press PF9 to display the group table. The group table cannot be modified.
- Press PF11 to return to the Modify File-Field Table screen.

Fields are shown in ADACMP FNDEF format: level, name, length, format [options]. See the *Adabas Utilities* documentation for more information.

The Group Offsets column names the group in which the field participates and the offset of the field within that group. If a field is a member of more than one group, the additional groups (and the field's offset within) are also listed.

For example, field AS belongs to group AQ and is located at offset 3. Field AT also belongs to group AQ but is located at offset 8.

To delete one or more entries

- Mark each entry to be deleted by typing a `D` in the Sel column next to the field name. Then press ENTER

Delete a File-Field Table

Use the delete function on the File-Field Table Definitions Menu only if the *entire* file-field table is to be deleted. Deleting the file-field table also deletes any associated group-field table for that file.

To delete an entire file-field table

- Enter `P` for Delete File-Field Table on the File-Field Table Definitions Menu.

  If a TRG0109 error occurs, it indicates that the table cannot be deleted until all triggers are removed. Display the file-field table and determine whether any of the fields in the table are marked "Triggr" (used in a trigger definition). If not, then the trigger definitions should be checked. It is likely that a trigger with the "all fields" option (i.e., field name `**`) has been defined for this file.

Generate a File-Field Table

To generate a file-field table

- Enter `G` on the File-Field Table Definitions Menu as well as the file name and a generation type code as shown in the following table:

Code	Source
F	Adabas FDT
D	Natural DDM
P	Predict file

  - FDT generations always originate from the DBID of the database for the current setting of the Trigger File.
  - DDM and Predict generations always originate from the database and file number of the Predict file. This may be taken from the current setting of FDIC; or it may be overwritten using `PF10` or by issuing the command `SET FDIC` from the command line of the Trigger Maintenance facility.

  Note: Wildcard notation cannot be used for the FDT file name. However, for the generate function only, "99999" can be entered for file number to display a list of all files loaded to the database.

  The screen that appears contains the source (FDT, DDM, or Predict file) from which the file-field table will be generated. You can select fields to be added to or deleted from the file-field table. See the ***sample FDT***.

When the file-field table is generated, a check is done to see if the field name (long) changed for a field (determined by the Adabas short name). If so, the status field contains the value "Alias" and the name of the new field is placed in the message field, preceded by a colon (:). Type an `A` in the Sel column next to the field name to update the entry with the new name.

In the case of FDT generations, the field name is always generated as the Adabas short name followed by "-FIELD". When the Trigger Maintenance facility finds a name that was not generated, it is used as the long field name with the "xx-FIELD" name in the message. Again, type an "A" in the Sel column to confirm the change. The ***sample FDT*** includes the status for each field. Fields marked "Active" are already contained in the file-field table.

If the generation type is "F" (Adabas FDT), field names are generated as "xx-FIELD", where "xx" is the Adabas field name. Subsequently, if a DDM or Predict Adabas file has the same file number and file name as that given for the FDT definition, these fields may be updated with the user-defined names of the DDM or Adabas file.

Note: Fields in the Field Definition column are shown in ADACMP FNDEF format where type "U" stands for unpacked numeric and field lengths are in bytes, not digits.

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****         YYYY-MM-DD
 User USR01         - Generate/Modify File-Field Table -       DBnr 105

  FDT File Name .. AUTOMOBILES                              FDT Fnr ... 4

  Sel Field Definition    Long Field Name   Status   Message
  --- -----------------   ---------------   ------   --------------
   _  01,AA,020,A,DE,NU   MAKE              Active
   _  01,AB,020,A,DE,NU   MODEL             Active
   _  01,AC,015,A,DE,NU   BODY-TYPE         Active
   _  01,BA,002,U,DE,NU   BA-FIELD
   _  01,BB,003,U,DE,NU   BB-FIELD
   _  01,BC,005,U,NU      BC-FIELD
   _  01,BD,005,U,NU      WEIGHT            Active
   _  01,CA,010,A,DE,NU   COLOR             Active
   _  01,DA,002,U,DE,NU   YEAR              Active
   _  01,DB,016,A,NU      DB-FIELD
   _  01,FA,006,U,DE,NU   FA-FIELD
   _  01,FB,006,U,DE,NU   MILEAGE           Active

   Mark Fields with 'A' Add, 'D' Delete, 'G'Generate, 'I' Info, or '.' Exit

 Command==>
 Enter-PF1---PF2---PF3---PF4--PF5---PF6--PF7--PF8--PF9--PF10--PF11--PF12
       Help  Menu  Exit       Next  --    -    +                  Canc
```

To add one or more fields

- Mark each field to be added by typing an `A` in the Sel column next to the field name. Then press ENTER.

  If you want to add all fields to the file-field table, enter `ALL` at the command line.

To delete one or more fields

- Mark each field to be deleted by typing a `D` in the Sel column next to the field name. Then press ENTER.

To generate group-field table entries

- Mark each field for which an entry is to be generated by typing a `G` in the Sel column next to the field name. Then press ENTER.

  Group-field table entries can only be created for fields that are part of a group or a PE group.

To display the attributes for an individual field

- Mark the field by typing an `I` in the Sel column next to the field name. Then press ENTER.

  The resulting window displays the field name, format, length, type (SP for superdescriptor; SB for subdescriptor), and an indicator if a group-field entry has been generated for the field.

Trigger Definitions

Note: For information about TRGMAIN, an API for maintaining triggers from a user program, see *TRGMAIN: An API To Maintain Triggers*.

Conceptually, a trigger has two parts: the triggering event and the triggered procedure. The triggering event is defined by a set of selection criteria. When the criteria are fulfilled, the triggered procedure is executed in response.

The selection criteria, i.e., file name, command type, and field name, are stored in the target database as part of the trigger definition. The file-field table maps them to the corresponding physical Adabas file number and two-character field ID. The file name and the field name are meaningful names of up to 32 characters.

Note: Database commands from the procedure that is executed as a result of a trigger are not limited to the database against which the initiating Adabas command is executed.

The following sections describe the selection criteria in more detail.

- File Name
- Command Type
- Field Name
- Trigger Definitions Menu

File Name

File name specifies the file against which the initiating Adabas command operates. A trigger is defined for one and only one Adabas file. If you want a trigger to apply to more than one file, define multiple triggers (one for each file) that are identical except for the file name.

Command Type

Command type specifies the command class of the initiating Adabas command. Triggers are defined to execute based on the presence of the specified command type.

One or all of the following Adabas command classes can be specified for a single trigger definition:

Command Class	Command Code
FIND	S1, S2, S4
READ	L1, L2, L3, L4, L5, L6, L9
UPDATE	A1, A4
INSERT	N1, N2
DELETE	E1, E4

If you want a trigger to apply to all command classes, leave the command type field blank. It defaults to "All" (all commands).

If you want a trigger to apply to more than one but not all command classes, define multiple triggers (one for each class) that are identical except for the command class.

Field Name

A trigger can be associated with a command that operates on a single field in the file or all fields in the file. It is possible, for example, to define a trigger that fires every time an `UPDATE` command is executed against the SALARY field in the EMPLOYEES File.

It is not always appropriate to specify a field. For example:

- It does not make sense to associate a specific field with a `DELETE` command, because the `DELETE` command does not require a format buffer.

- In the case of a pre-command trigger read command, the field will contain no data. Therefore, there is no need to specify a field unless you want to validate either the fields to be read or the user ID of the user issuing the request.

If you want a trigger to apply to more than one but not all fields, define multiple triggers (one for each field) that are identical except for the field name.

A trigger will be fired for only one field, i.e., the field that is specified in the format buffer, depending on the event selection criteria. However, if triggers need to be fired for multiple fields, it is possible to define the trigger file for a specific command and field, then have the procedure itself check for the existence of other fields. See the section *Implementing Support for Multi-Triggers*.

The procedure can verify whether additional procedures should be invoked and, if so, for what fields. This mechanism also allows the "main" procedure to handle errors and decide whether another procedure should be invoked even if the previously executed procedure resulted in an error. The user therefore has the flexibility to control situations where a precise set of rules is needed to determine whether a procedure should or should not be fired.

Trigger Definitions Menu

The Trigger Definitions Menu (shown below) contains the functions that allow you to create and maintain trigger definitions.

To access the Trigger Definitions Menu

- Enter `T` for Create/Modify Trigger Definitions on the main menu.

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****          YYYY-MM-DD
 User USR01               - Trigger Definitions Menu -            DBnr 105

                            Code  Function
                            ----  --------------------------
                              A   Add Trigger Definition
                              D   Display Trigger Definition
                              M   Modify Trigger Definition
                              P   Delete Trigger Definition
                              S   Select Trigger Definition
                              ?   Help
                              .   Exit
                            ----  --------------------------
                Code ........ _       Active/Deactive Opt... _
                File Name ... ____________________
                Cmd Type .... _
                Field Name .. ____________________

 Command ==>
 Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7--PF8--PF9--PF10--PF11--PF12--
       Help  Menu  Exit  Field       Admin Procs          FTRG  FDIC  Canc
```

The functions on the Trigger Definitions Menu allow you to add, display, modify, or delete trigger definitions. You can display a screen that contains multiple trigger definitions for the same file, or a pop-up window that contains a single trigger definition for a specific field within the file.

To access all trigger definitions for a particular file

- Enter the function code (A for add, D for display, M for modify, or P for purge/delete) and the file name. Enter a wild card value for command type and field name.

  A screen containing all trigger definitions for the file appears. Depending on the function code entered to access the screen (A, D, M, or P), you can add, display, modify, or delete one or more trigger definitions. See the section *Multiple Trigger Definitions*.

To access a specific trigger definition

- Enter the function code (A, D, M, or P), the file name, and the field name.

  A pop-up window containing the trigger definition appears. Depending on the function (A, D, M, or P), you can add, display, modify, or delete the trigger definition. See the section *Single Trigger Definition*.

You can also use the Trigger Definitions Menu to select one or more triggers.

To select a trigger or triggers

1 Enter function code `S` and a file name or a wildcard value for file name, e.g., `>G`.

  The screen that appears is similar to the following example:

  Note: If a specific file name is selected, the screen contains the long field names and not the file names.

```
HH:MM:SS          ***** TRIGGER MAINTENANCE *****             YYYY-MM-DD
User USR01             - List Trigger Definitions -           DBnr 105

Sel File Name / Long Field Name    Commnd  Type      ProcName Para   RecB
--- --------------------------    ------- ----      -------- ------ ----
_ MISCELLANEOUS                   Delete  Pre Non-P SYMP0003 Cont   None
   ** Any Field **
_  VEHICLES-FILE                  Delete  Pre Non-P SYMP0003 Cont   None
   ** Any Field **
_  VEHICLES-FILE                  Read    Pre Async ANYONE   Contrl None
   ** Any Field **
_  VEHICLES-FILE                  Read    Pre Async MAKE0001 Cont   None
   MAKE
_  VEHICLES-FILE                  Read    Pre Async MODEL001 Cont   None
   MODEL
_  VEHICLES-FILE                  Read    Pre Async COLOR001 Contrl None
   COLOR
_  VEHICLES-FILE                  Read    Pre Async CLASS001 Contrl None
   CLASS
_  GDMUSIC                        (All)   Pre Async PROC008  Contrl None
   ** Any Field **

Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7--PF8---PF9---PF10--PF11--PF12
           Menu  Exit              --    -    +                      Canc
```

2 In the Sel field next to the file name, enter `D` for display, `M` for modify, `S` for select, or `P` for purge.

If you enter D, P, or M, a pop-up window containing the trigger definition appears. See the section ***Single Trigger Definition***.

If you enter S, the trigger is selected and the Trigger Definitions Menu appears.

This section also covers the following topics:

- Multiple Trigger Definitions
- Single Trigger Definition

Multiple Trigger Definitions

To add, display, modify, or delete one or more trigger definitions

1 On the Trigger Definitions Menu, enter the code (A, D, M, or P) and the file name. The command type is optional; if you leave it blank, it defaults to "All" (all commands).

Depending on the code entered, one of the following screens appears:

- Add Trigger Definitions

- Display Trigger Definitions
- Modify Trigger Definitions
- Delete Trigger Definitions

These screens contain the trigger definitions for the specified file. For example, the Modify Trigger Definitions screen is similar to the following:

```
HH:MM:SS                ***** TRIGGER MAINTENANCE *****          YYYY-MM-DD
User USR01              - Modify Trigger Definitions -           DBnr 105

                                                          Pre-Post .. Pre
File ... VEHICLES-FILE (3)                                Command ... Read

Prty Long Field Name                  Fld Type   ProcName Params RecBuff Msg
---- -------------------------------- --- -----  -------- ------ -------
010  COLOR____________________________ AF  Async  COLOR001 Contrl None___
020  CLASS____________________________ AH  Async  CLASS001 Contrl None___
030  MAKE_____________________________ AD  Async  MAKE0001 Contrl None___
040  MODEL____________________________ AE  Async  MODEL001 Contrl None___
050  ** Any Field **__________________ **  Async  SAMP0002 Contrl None___
___  ________________________________ __  _____  ________ ______ _______
___  ________________________________ __  _____  ________ ______ _______
___  ________________________________ __  _____  ________ ______ _______
___  ________________________________ __  _____  ________ ______ _______
___  ________________________________ __  _____  ________ ______ _______
___  ________________________________ __  _____  ________ ______ _______
Modify Entries, or enter '.'(Exit), or '?'(Help) to see options ... _

Command ==>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12
      Help  Menu  Exit        Updat       -     +           Reseq Post  Canc
```

2 After entering the appropriate information, press PF5 to update the trigger table.

A message informs you that the update has been confirmed. An error message appears if you enter information that is invalid or incomplete.

Entry Fields

The following paragraphs describe the entry fields in the Trigger Definitions screens.

Note: You are not required to enter both the long field name and the Fld (short field name). If either is entered, the other is derived from the file-field table entry.

Prty (priority)

The Adabas trigger driver scans the format buffer for a match with the selection criteria defined for each trigger. When multiple field names are specified for the same file and command, the

priority assigned to each field determines the order in which it is processed. When a match is found, the trigger is fired. The Prty field allows you to set or modify the sequence.

The highest priority is 1. Priorities are represented, however, in steps of ten from 10-900. Other values are entered to change the sequence or "priority" of the fields. Values entered between the steps of ten are resequenced (using the `RESEQ` command; see the section ***Commands***) to the next higher 10s values; for example, entering priorities of 11 and 12 for fields resequences them to represented values 20 and 30, respectively.

To change a field's priority, specify a value between 1 and 9

- higher than the represented value of the field you want it to follow; or
- lower than the represented value of the field you want it to precede.

Example:

To change the priority of MAKE on the List Trigger Definitions screen from the represented value 30 to 10 with the other fields changing priority accordingly, assign MAKE a value between 1 and 9.

When the `RESEQ` command is used, the priorities of all fields are changed to 10-900 with MAKE as 10, COLOR as 20, CLASS as 30, etc.

Long Field Name
The Adabas long field name for the field. Enter a wildcard to display a selection list of field names. If the long field name is not known, it can be derived from the Fld (short field name).

Fld (short field name)
The Adabas short field name, that is, the unique name used by the DBMS to identify a particular field for a particular file. It must correspond to the long field name for the field. If the short field name is not known, it can be derived from the long field name.

Type
Type is asynchronous, participating, or non-participating. The default value is asynchronous.

ProcName
The name of the Natural subprogram that should be invoked when the selection criteria for the trigger are met. The value must be a valid Natural subprogram name of 1-8 characters. There is no default value.

Important: The name of the user job that calls the trigger must be different from the trigger's ProcName.

Params
When the trigger procedure is invoked, the parameters passed may be:

Contrl	Using the ACB interface, control parameters are used to pass information about the trigger request as well as the trigger command and a modifiable response code field. Contrl is the default value.
Contrx	Using either the ACB or ACBX interface, control parameters are used to pass information about the trigger request as well as the trigger command and a modifiable response code field.
Resp	A modifiable response code field is used to prevent the execution of a command in the case of a pre-command trigger. Response code is used with synchronous triggers only; it has no value or meaning with asynchronous triggers, which may already have completed.
None	No parameters are passed.

RecBuff

Value may be access (read only), update (read and write), or none.

Msg

An error message text may be displayed in this field when an error occurs. An explanation of the error is displayed at the bottom of the screen.

Commands

The following table describes the commands that can be entered at the command line in the Trigger Definitions screens:

Note: Commands can be entered in upper-, lower-, or mixed-case.

Command	Description
PRTY	Display the Modify Trigger Definitions screen, where you can modify the priority assigned to a trigger.
UPDATE	Update the trigger file with the values entered.
BACK, -	Display the previous screen.
FWD, +	Display the next screen.
RESEQ	Resequence the list of trigger definitions, in order by priority.
POST	Display the post-command triggers for this file and command.
PRE	Display the pre-command triggers for this file and command.
ACTIVATE	Activate pre-command triggers or post-command triggers.
DEACTIVATE	Deactivate pre-command triggers or post-command triggers.
DELETE	Delete selected trigger definitions from the trigger file.
MODIFY	Modify the trigger definitions.

Single Trigger Definition

To add, display, modify, or delete a single trigger definition

- Enter the code (A, D, M, or P), the file name, and the field name on the Trigger Definitions Menu.

  The command type is optional; if you leave it blank, it defaults to "All" (all commands).

  Depending on the code entered, an add, display, modify, or delete function pop-up window appears. These windows contain trigger information and procedure information about the specified trigger definition (see the example **Modify Function Screen**):

  - Trigger information includes the file name and number, command type (read, update, etc.), the long field name and the Adabas field name. Also displayed is the current status of the trigger, as shown in the following table:

Status	Description
Active	The trigger is currently active on the trigger file and for the currently active nucleus.
Inactive	A permanent deactivation of the trigger was requested; the trigger will remain in this state until activated.
Temp Active	The trigger is active for the running nucleus only. A temporary "activate" was requested; the trigger remains in this state until a "deactivate" is issued.
Temp Inactive	The trigger is inactive for the running nucleus only. A temporary "deactivate" was requested; the trigger remains in this state until an "activate" is issued.
File Not Used	The file number specified in the trigger definition is greater than the maximum file number for the database (highest file number used plus 10). See *Creating the Trigger Table*.
Not Loaded	A trigger has been added to the file since the last time the nucleus was activated. The trigger will become active the next time a `REFRESH` command is issued or the nucleus is restarted.
Not Checked	The Trigger Maintenance facility is not active in the nucleus; therefore, no checking for the status of a trigger can be done.
Inaccessible	The Adabas nucleus is not accepting any requests for trigger status.

  - Procedure information includes the procedure name, pre-command or post-command status, the trigger type (asynchronous, participating or non-participating), the `CALLNAT` parameters type (cntrl, resp, or none) and the record buffer option (access, update, or none).

To add a trigger definition

1 Enter the name of the procedure.

2 Modify options as required.

3 Press PF5 to confirm the addition.

To modify a trigger definition

1 The Modify Function pop-up is similar to the following:

```
Modify Details and press 'PF5' to Confirm Update.

   HH:MM:SS   *** Define Trigger Info ***   YYYY-MM-DD
                   - Modify Function -

   Trigger Information   ** Active **
     File Number ....... 4
     File Name ......... AUTOMOBILES
     Command Type ...... Read
     Long Field Name ... BODY-TYPE
     Adabas Field ...... AC
     Field Prty/Seq .... 010
   Procedure Information
     Name (Subpgm)...... _________
     Pre Cmd Select .... N  (Post)
     Trigger Type ...... N  (Non-Participating)
     CALLNAT Params .... C  (Cntl Info + Resp)
     RecBuffer Access .. A  (May be Accessed)

   Command ==>
   Enter-PF1--PF2--PF3--PF4--PF5--PF6--PF7--PF8--
         Help Menu Exit Prty Updat     Act  Deact
```

2 Modify the Field Prty/Seq (priority or sequence) and/or any of the fields under Procedure Information.

3 If you want to activate or deactivate the trigger, enter `ACTIVATE` or `DEACTIVATE` at the command line. Then choose whether the trigger's status should be changed temporarily or permanently:

Command		means...
Activate	temporarily	activate the trigger in the nucleus but retain its inactive status in the trigger file.
	permanently	activate the trigger in the nucleus and remove its inactive status from the trigger file.
Deactivate	temporarily	ignore the trigger for any checking of event criteria for any command issued to the database; that is, no trigger can be fired for this definition.
	permanently	store the trigger on the trigger file with the status "deactive"; do not activate when the nucleus is started. The trigger may, however, be started at a later time. An active Adabas session is deactivated immediately.

4 Press PF5 to confirm the update.

To delete a trigger definition

- Press PF5 to confirm the deletion.

To display all fields for the file in sequence by priority

- Press PF4 or enter PRTY at the command line.

  Note: If the trigger specifies the command type "Delete", the screen that appears allows the user to define a pre- and/or post-trigger definition at the same time.

Procedure Reports

The Procedure Reporting Menu shown below allows you to obtain an alphabetic list of triggered procedures sorted by file or by procedure name. You can limit the report to a specific file or include all files. Procedure reports can be used, for example, to locate duplicate procedures or identify each instance where a particular procedure is used.

To access the Procedure Reporting Menu

- Enter R for Procedure Reports on the Main Menu.

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****       YYYY-MM-DD
User USR01             - Procedure Reporting Menu -         DBnr 105

                          Code  Function
                          ----  --------------------------
                            F   Display Procedures by File
                            N   Display Procedures by Name
                            ?   Help
                            .   Exit
                          ----  --------------------------

             Code ......... N
             File Number .. _____
             Procedure .... PROC0001




 Command ==>
 Enter PF1---PF2--PF3--PF4---PF5---PF6---PF7---PF8--PF9--PF10--PF11--PF12
       Help  Menu Exit Field Trigr Admin                           Canc
```

To list the procedures for a particular file only

- Enter code `F` for Display Procedures by File and the file number.

To list procedures beginning with a particular name

- Enter code `N` for Display Procedures by Name, and the procedure name.

In either case, the screen displayed contains the type of information shown in the following example.

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****        YYYY-MM-DD
User USR01               - List Procedure Names -            DBnr 105

 File Name ... AUTOMOBILES                                       Fnr ... 4


 Sel  ProcName   Command  Field Name                   When  Type   ParmTy
 --- --------   -------  ---------------------------  ----  -----  ------
  _   PROC0001   Read     BODY-TYPE                     Post  NonP   Cntrl
  _   PROC0001   Update   ** Any Field **               Pre   Async  Cntrl
      PR0001     (All)    MILEAGE                      Pre   Async  Cntrl
  _   SUBPGM     Read     COLOR                        Pre   Async  Cntrl




 Select 'D' to Display, or enter 'F'(Fwd), 'T'(Top), 'B'(Bck), '.'(Exit)
 Command ==>
Enter-PF1---PF2---PF3---PF4---PF5---PF6--PF7--PF8---PF9--PF10--PF11--PF12
      Help  Menu  Exit              --    -    +                    Canc
```

The procedure report contains the database number, the file number, the file name, and the following information for each procedure:

Field	Description
`ProcName`	The name of the triggered procedure.
`Command`	The Adabas command that initiates the trigger.
Field Name	The trigger field name.
`When`	When the procedure executes: ■ "pre-command": The procedure executes before the initiating Adabas command. ■ "post-command": The procedure executes after the initiating Adabas command.
`Type`	The type of trigger: ■ "non-participating": The triggered procedure does not participate in the initiating Adabas command's transaction logic.

Field	Description
	■ "participating": The triggered procedure may participate in the initiating Adabas command's transaction logic. ■ "asynchronous": The Adabas command and the triggered procedure execute separately and do not affect each other. The triggered procedure's transaction logic does not participate in the Adabas command transaction logic.
ParmTy	The type of parameters passed when the procedure is invoked: ■ "control": ACB-style control parameters are used to pass information about the trigger request as well as the trigger command and a modifiable response code field. This is the default value. ■ "controlx": ACB- or ACBX-style control parameters are used to pass information about the trigger request as well as the trigger command and a modifiable response code field. ■ "response": A modifiable response code field is used to prevent the execution of a command in the case of a pre-command trigger. Response code is used with synchronous triggers only; it has no value or meaning with asynchronous triggers, which may already have completed. ■ "none": No parameters are passed.

To obtain more information about a specific procedure

■ Enter `D` in the Sel column next to the procedure name.

A pop-up window displays information about the trigger definition as well as the procedure, and is similar to the following example:

```
Trigger Information currently displayed

               HH:MM:SS   *** Define Trigger Info ***   YYYY-MM-DD
                             - Display  Function -
               Trigger Information
                 File Number ....... 4
                 File Name ......... AUTOMOBILES
                 Command Type ...... Read
                 Long Field Name ... BODY-TYPE
                 Adabas Field ...... AC
                 Field Prty/Seq .... ____
               Procedure Information
                 Name (Subpgm)...... PROC0001
                 Pre Cmd Select .... N  (Post)
                 Trigger Type ...... N  (Non-Participating)
                 CALLNAT Params .... C  (Cntl Info + Resp)
                 RecBuffer Access .. A  (May be Accessed)

               Command ==>
               Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8-
                     Help  Menu  Exit  Prty
```

- Trigger information includes the trigger event criteria, i.e., file name and number, command type, long field name and the Adabas field name.
- Procedure information includes the Natural subprogram name, whether the procedure is pre-command or post-command, the trigger type (asynchronous, participating or non-participating), the `CALLNAT` parameters category (cntrl, resp, none) and the RecBuffer access status (A=access, N=no access, U=access/update).

Administrator Functions

The Administrator Functions allow you to monitor trigger activity, display and modify the profile, and maintain job status settings and buffer sizes.

To access the Administrator Functions Menu

- Enter `A` for Administrator on the Main Menu.

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****          YYYY-MM-DD
User USR01            - Administrator Functions Menu -         DBnr 105


                      Code  Function
                      ----  --------------------------
                        A   Active Session Settings
                        D   Display Profile Information
                        M   Modify Profile Information
                        S   Subsystem Activity
                        T   Trigger Activity
                        ?   Help
                        .   Exit
                      ----  --------------------------

               Code ... D

Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6--PF7--PF8---PF9---PF10--PF11--PF12--
      Help  Menu  Exit  Field Trigr      Procs           FTRG        Canc
```

Active Session Settings

Active session settings includes job status settings and buffer sizes.

To modify the active session settings

1 On the Administrator Functions Menu, enter `A` for Active Session Settings.

2 Modify the field values and press PF5 to update the settings.

In a nucleus cluster environment, any changes to the active session settings are populated to all active nuclei in the cluster.

The Active Session Settings screen is similar to the example shown below:

```
HH:MM:SS               ***** TRIGGER MAINTENANCE *****           YYYY-MM-DD
User USR01               - Active Session Settings -             DBnr 105

 Job Name ..........SAGDT077           Trigger File Number ......12

 SVC Number ........217                 Max File to be accepted...60
 Nucleus ..........  Active__
 Triggers .........  Active__           Session Buffer Sizes in Bytes
 Stored Proc. .....  Active__             Trigger Table Buffer....8192
 Error Action .....  Halt                 Pre Trigger Queue.......15244
 Trigger Logging ..  Active__             Post Trigger Queue......2960
 Activity Timeout..  600                  Waiting Subsys Queue....80
 Subsystems                               Acquired Storage........31232
   Maximum ........5                      Used Storage............31232
   Active .........5
   Inactive ......  0
   Waiting .........5
   In Progress ....0

Change Parameters as required or press 'PF3' to Exit

Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
      Help  Menu  Exit  Sact  Updat                                    Canc
```

Note: Press PF4 to display the Subsystem Activity screen.

The following table describes the Active Session Settings fields:

Field	Description
Job Name	The job name for the Adabas nucleus.
SVC Number	The SVC number being used by the database.
Nucleus	Indicates when Adabas Triggers and Stored Procedures is active for the current database.
Triggers	If "active" (the default), triggers can be executed for this database; if "inactive", they cannot. This field can be modified. You may enter "refresh" to update the trigger table (see the section *Updating the Trigger Table*).
Stored Procedures	If "active" (the default), stored procedures can be executed for this database; if "inactive", they cannot. This field can be modified.
Error Action	The action ("reject", "halt", or "ignore") to be taken by Trigger Maintenance when a processing error occurs. This field can be modified.
Trigger Logging	The logging function is "active" or "inactive".
Activity Timeout	The number of seconds before a task is canceled. The default value is 60; the maximum value is 9999. This field can be modified. If set to zero, it will default to the Adabas TT parameter value.

Field	Description	
Subsystems	Maximum	The number of assigned Natural subsystems. The value can be 01-10.
	Active	The number of Natural subsystems initiated. The value may be less than the maximum allowed if an error from which the system cannot recover occurs and the subsystem becomes inactive.
	Inactive	The number of Natural subsystems currently terminated or shut down due to the occurrence of errors from which the system could not recover.
	Waiting	The number of subsystems currently waiting for work.
	In Progress	The number of subsystems currently processing triggers.
Trigger File Number	The file number of the trigger file for the database.	
Max File to be accepted	The maximum file number that Adabas Triggers and Stored Procedures will accept. This is set as the highest file loaded plus 10 when the database comes up and/or is initialized. If a higher file number is subsequently added to the database, any triggers found for the file are ignored. To activate triggers beyond the Max File value, the database must be shut down and restarted.	
Session Buffer Sizes in Bytes	Trigger Table Buffer	The size of the trigger table buffer. This field can be modified using the Modify Profile function.
	Pre-Trigger Queue	The size of the queue for all pre-command triggers being processed at any given time.
	Post-Trigger Queue	The size of the queue for all post-command triggers being processed at any given time.
	Waiting Subsys Queue	The size of the queue for tasks waiting to process triggers; currently fixed at 80 bytes.
	Acquired Storage	The total amount of storage to be acquired for use by Adabas Triggers and Stored Procedures.
	Used Storage	The actual amount of storage used by Adabas Triggers and Stored Procedures during processing.

Updating the Trigger Table

By using the REFRESH command, you can add new triggers to the trigger table without shutting down the database. In nucleus cluster environments, the REFRESH command updates the trigger table on all cluster nuclei.

If the number of additional triggers to be loaded is far greater than the number loaded initially, manually allocate enough additional space to handle the increased number of triggers. If the trigger table buffer is not large enough, an inconsistency may occur that terminates Adabas Triggers and Stored Procedures based on the value of the error action field in the Adabas triggers profile.

The need to update the trigger table can be avoided by preloading the required triggers before the applications that use them are implemented.

To update the trigger table,

- Enter `REFRESH` at the command line, or enter "refresh" in the triggers field.

Buffer Size Calculations

Buffers are required for the trigger table, the pre-trigger queue, and the post-trigger queue:

Trigger Table

The trigger table buffer size is calculated using the following formula.

```
((((HIFNRLD+10)*4)+HIFNRLD+10+TOTTRG)*24)+4096
```

where:

HIFNRLD	is the highest file number loaded
TOTTRG	is the total number of triggers defined on the trigger table

After the Adabas trigger driver has calculated the total buffer size, the result is rounded down to a multiple of 256.

Pre-Trigger Queue and Post-Trigger Queue

Depending on how many commands per second are passed to the database, how long the actual procedures run, and whether the triggers are synchronous or asynchronous, queuing may or may not occur. Each buffer is set up to independently queue the pre- and post-triggers. If the queues become full, subsequent commands that would result in triggers being fired receive a response code 154 (ADARSP154). After the queuing has eased, the DBA should consider increasing the queue size.

In setting up the buffer sizes, consideration must also be given to the ratio of pre-triggers to post-triggers. For instance, if no pre-triggers are used, the pre-trigger queue is not required; all buffer space should be allocated to the post-trigger queue.

- The pre-trigger queue is required only if pre-command triggers are defined on the trigger file. Its buffer size is calculated as follows:

  ```
  (NC / 2) * 352  = TOTAL SIZE
  ```

  where "NC" is the value of the ADARUN parameter `NC`.

  This calculation is valid if all triggers are synchronous. If asynchronous triggers are used, commands may be issued continuously before the procedure resulting from the previous command has completed. This results in queuing, which requires a larger buffer size.

Whether the amount the buffer should be increased depends on the number of commands being issued and the speed at which they are issued (i.e., the amount of time between the response from one command and the issue of the next command), and the amount of time required for each trigger's procedure to complete. For example, a batch job issuing thousands of commands that fire asynchronous triggers would require a very large buffer.

- The post-trigger queue is required only if post-command triggers are defined on the trigger file. Buffer size is calculated exactly the same as the pre-trigger queue buffer size, and the same size considerations apply.

Display/ Modify Profile Information

The profile contains system information required by Adabas Triggers and Stored Procedures and is generated from the database ID and file number that you specify. After the profile is generated, you can display it and modify the values it contains.

To access the profile,

- Enter `D` for Display Profile Information on the Administrator Functions Menu.

  The Display Profile Information screen appears, and is similar to the following example:

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****         YYYY-MM-DD
User USR01            - Display Profile Information -         DBnr 105

 Triggers Status ..... Active__                 Total Triggers ..34
 Stored Proc. Status ..... Active__

 Natural Subsystem Parameters
   Batch Natural Name ..... NATAPT
   Maximum Subsystems ..... 6_
   Activity Timeout ....... 600/80
   NATPARM Parameters ..... DU=OFF,INTENS=1,ETID='  ' __________
                            ____________________________________

   Fixed NATPARM Parm ..... STACK=(LOGON:SYSSPT;STP),PROGRAM=STPEND
   CMPRINT Assignment ..... TSPRT

 Required .. N     UserID ... USER**__  Password .. PSWD**__

 Adabas Session Parameters
   Error Action ..........  Halt__   Trigger Table Size ....... __10K
   Log Trigger Activity ... Active__ Pre Trigger Queue Size ... __35K
                                     Post Trigger Queue Size .. __50K


Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
      Help  Menu  Exit        Mod                                    Canc
```

Adabas Triggers and Stored Procedures assigns default values to the fields in the profile and uses these values at initialization time. This information is valid only at initialization time, i.e., when the Adabas nucleus is started. If the profile is modified, the new values take effect the next time the nucleus is bounced.

To modify the profile

1 On the Display Profile Information screen, press PF5. On the Administrator Functions Menu, enter M for Modify Profile Information.

The Modify Profile Information screen appears.

2 Enter the new values and press PF5 or enter UPDATE at the command line.

A message informs you that the profile has been updated successfully.

Profile Fields

The following table describes the fields in the profile:

Field	Description
DBnr	The database ID of the trigger file to which this profile applies. The value is entered using NTLFILE, LFILE, or PF10 (FTRG).
Trigger Status	This field can be modified. active: (the default) triggers can be executed for this database. inactive: triggers cannot be executed for this database.
Stored Proc. Status	This field can be modified. active: (the default) stored procedures can be executed for this database. inactive: stored procedures cannot be executed for this database.
Total Triggers	The total number of triggers defined for this database ID. May be used, by default, to calculate the size of the trigger table buffer. Value is derived from the trigger definitions added to the trigger file. As a safety mechanism, this number is verified as correct when the NUMBER command is issued.
Batch Natural Name	The Natural nucleus that will be started by the Adabas trigger driver to run the 1-10 Natural subsystems that are responsible for the actual execution of the procedures. The name is assigned to the Natural nucleus component during the installation procedure.
Maximum Subsystems	This field can be modified. The number of Natural subsystems that should be activated for a given Adabas session. The value may be 01-10.
Activity Timeout	This field can be modified. The number of seconds before a task is canceled. The default is 60; the maximum value is 9999. If set to zero, it defaults to the Adabas TT parameter value.

Field	Description
NATPARM Parameters	Dynamic parameter overrides for the NATPARM module linked to the Natural nucleus (the NATPARM module specifies the options to be in effect for the Natural session). See the section *NATPARM Considerations*.
Fixed NATPARM	This value is generated by Adabas Triggers and Stored Procedures. `STACK=(LOGON:SYSSPT;STP)` where SYSSPT is the library where the procedures are executed and STP is used to invoke the Natural driver.
CMPRINT Assignment	This field can be modified. Dynamic assignment for the CMPRINT label. The default value is TSPRT. Printing from any procedure within a specific subsystem must be directed to this label unless the specific printer number notation is used in Natural syntax; for example, PRINT(01), DISPLAY(02), or WRITE(03).
NATSEC LOGON Required	This field applies only if the Natural subsystems are running under Natural Security. It indicates whether logon to Natural Security is required (Y) or not (N). Pertains to `AUTO=OFF`, `AUTO=ON`, respectively.
UserID	This field can be modified. This field applies only if the Natural subsystems are running under Natural Security. This is the Natural Security user ID. The default value is USER**. The ** is replaced by the dynamic task number or subsystem number (value 01-10). Although the default uses ** as a suffix, it may occur anywhere in the user ID provided that at least one nonblank character is specified as well.
Password	This field can be modified. This field applies only if the Natural subsystems are running under Natural Security. This is the Natural Security password. The default value is PSWD**. The ** is replaced by the dynamic task number or subsystem number. Although the default uses ** as a suffix, it may occur anywhere in the password provided that at least one nonblank character is specified as well.
Error Action	This field can be modified. This is the action to be taken by Adabas Triggers and Stored Procedures after a processing error from which the system cannot recover. **Halt** — (the default) terminates the Adabas nucleus as if ADAEND were issued for both cluster and noncluster nuclei. **Reject** — keeps Adabas Triggers and Stored Procedures active, but any command for which a trigger is fired receives a response code 157 (ADARSP157); In a cluster environment, if one nucleus switches to "reject", all other nuclei also switch. However, if there is a problem during a refresh and the error action is "reject", the nucleus shuts down: it cannot reject commands if it has a damaged or incomplete trigger table. **Ignore** — shuts down Adabas Triggers and Stored Procedures, but keep the nucleus active as if Adabas Triggers and Stored Procedures had never been activated. In a cluster environment, if one nucleus switches to "ignore", all other nuclei also switch. If there is a problem during a refresh and the error action is "ignore", then all nuclei switch to "ignore".

Field	Description
	Note: The "ignore" option may cause application integrity problems.
Log Trigger Activity	This field can be modified. active: trigger activity is recorded in the log. inactive: (the default) trigger activity is not recorded in the log. Whenever a trigger or a stored procedure is invoked, information about the procedure can be printed to the print spool, as defined by the user. This information may be useful for auditing or debugging purposes.
Trigger Table Size	This field can be modified. This is the size (in bytes) of the trigger table buffer. If additional triggers are loaded to the trigger table, you may need to increase the buffer size (see the section *Updating the Trigger Table*). The default value is calculated based on the number of triggers defined in the trigger file. (See the section *Buffer Size Calculations*.)
Pre-Trigger Queue Size	This field can be modified. This is the size (in bytes) of the pre-trigger queue buffer, that contains pre-command triggers prior to selection for processing. The default value is calculated (see the section *Buffer Size Calculations*.)
Post-Trigger Queue Size	This field can be modified. This is the size (in bytes) of the post-trigger queue buffer that contains post-command triggers prior to selection for processing. The default value is calculated (see the section *Buffer Size Calculations*.)

Subsystem Activity

Subsystem activity provides information about currently executing Natural subsystems.

To access subsystem activity information

- Enter `S` for Subsystem Activity on the Administrator Functions Menu.

  The Subsystem Activity screen is similar to the example shown below:

```
HH:MM:SS               - Subsystem Activity -            YYYY-MM-DD

 Nmbr  Started Status Type  Start  / TimeOut of Trigger  Trig Count
 ----  ------- ------ ----- ---------------------------- ----------
 01   09:25:57 Busy   Sync  09:41:37 09:46:37                  350
 02   09:25:57 Busy   Async 09:41:39 09:46:39                  280
 03   09:25:57 Wait                                          212
 04   09:25:57 Wait                                          156
 05   09:25:57 Wait                                           51




Subsystems . .  5 of  5
Command ==>
```

The following table describes the information in the Subsystem Activity screen:

Field	Description
Nmbr	The number of the Natural subsystem.
Started	The time that the subsystem was initialized. If different from the Started time for the other subsystems listed, it indicates that a timeout, cancellation, or termination occurred. This should be cross-checked.
Status	The current status of the subsystem: busy, active, wait (waiting for work), shutdown, canceled, or abended (if an error occurred).
Type	The type of trigger: sync (synchronous) or async (asynchronous).
Start	The time when the procedure started executing.
Timeout	The time when a procedure will be canceled if it executes beyond the maximum time limit. If an error occurs prior to the timeout, the error is displayed.
Trig Count	The number of triggers that have been executed by the subsystem since initialization.
Command	The DBA may cancel any active or waiting subsystem by first entering the `CANCEL` command and then entering a `C` in the cursor position under "Status" for the subsystem that should be canceled.

To cancel an active or waiting subsystem

1 Enter `CANCEL` at the command line.

2 Enter `C` (for cancel) in the Status field of the subsystem that is to be canceled.

3 Press PF3 to exit.

Trigger Activity

Trigger activity provides information about currently executing triggers.

To access trigger activity information

- Enter "T" for Trigger Activity on the Administrator Functions Menu.

  The Current Trigger Activity screen is similar to the example shown below:

```
HH:MM:SS              ***** TRIGGER MAINTENANCE *****          YYYY-MM-DD
User USR01             - Current Trigger Activity -            DBnr 105

Nmbr Status   Cmd  Fnr  Field  ProcName Type        RecBu   UserID (hex)
---- ------   ---  ---  -----  -------- ----------  -----  -------------
 01  Active                   PROC0002  Pre Non-P  None   ABD6EA375DE96A01
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  L3   11   **     SYMP0002  Pre Async  None
     Waiting  S4   11   **     SYMP0002  Pre Async  None
     Waiting  N1   11   LE     SYMP0001  Pre Non-P  Upd


Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
            Menu  Exit  Sact  Refr  --      -     +           >          Canc
```

The following table describes the information in the Current Trigger Activity screen:

Field	Description
Nmbr	The number of the Natural subsystem executing the trigger. Valid values are 01-10.
Status	The current status of the subsystem: active or waiting.
Cmd	The Adabas command that initiated the trigger.
Fnr	The file number against which the command is invoked. If the value of this field is the same as the trigger file, then this is a stored procedure.
Field	The name of the field that resulted in the trigger being fired.
ProcName	The name of the triggered procedure.
Type	The type of trigger: pre (pre-command) or post (post-command), non-P (non-participating), part (participating), or async (asynchronous).
RecBu	The RecBuff setting for the trigger: access (read only), update (read and write), or none.
UserID	The (hexadecimal) ID of the user who issued the actual Adabas command.

To display additional information

- Press PF10.

  The screen scrolls to the right, as shown below:

```
HH:MM:SS               ***** TRIGGER MAINTENANCE *****          YYYY-MM-DD
User USR01              - Current Trigger Activity -            DBnr 105

Nmbr Status   Cmd  Fnr  Field  ProcName CID  CID (hex) ISN in ACB Timeout
---- ------   ---  ---  -----  -------- ---  --------- ---------- ---------
 01  Active             **     PROC0002      00000000              11:30:41
     Waiting  L3   11   **     SYMP0002  ??? 00200101
     Waiting  L3   11   **     SYMP0002  ??? 00200101  1
     Waiting  L3   11   **     SYMP0002  ??? 00200101  16
     Waiting  L3   11   **     SYMP0002  ??? 00200101  15
     Waiting  L3   11   **     SYMP0002  ??? 00200101  14
     Waiting  L3   11   **     SYMP0002  ??? 00200101  20
     Waiting  L3   11   **     SYMP0002  ??? 00200101  23
     Waiting  L3   11   **     SYMP0002  ??? 00200101  2
     Waiting  S4   11   **     SYMP0002 ???? 02100101
     Waiting  N1   11   LE     SYMP0001      00000000


Command ==>
Enter PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
            Menu  Exit  Sact  Refr  --      -     +           <          Canc
```

Field	Description
Nmbr	The number of the Natural subsystem executing the trigger. Valid values are 01-10.
Status	The current status of the subsystem: active or waiting.
Cmd	The Adabas command that initiated the trigger.
Fnr	The file number against which the command is invoked. If the value of this field is the same as the trigger file, then this is a stored procedure.
Field	The name of the field that resulted in the trigger being fired.
ProcName	The name of the triggered procedure.
CID	The command ID in the Adabas control block (ACB).
CID (hex)	The command ID repeated in hexadecimal format.
ISN in ACB	The ISN from the Adabas control block (ACB).
Timeout	The time when a procedure will be canceled if it executes beyond the maximum time limit.

VII TRGMAIN: An API To Maintain Triggers

9 TRGMAIN: An API To Maintain Triggers

The Adabas triggers and stored procedures facility provides a callable routine TRGMAIN to maintain trigger definitions from user-written programs. TRGMAIN functions include

- add new triggers
- modify existing triggers
- purge existing triggers
- display a specific trigger definition
- activate triggers
- deactivate triggers

To call the trigger maintenance routine

- Enter the following:

```
CALL 'TRGMAIN' function calling-parameters
```

where the functions and calling parameters are as described in the following sections.

Functions (Format A5)

The `FUNC` parameter specifies what action is to be performed on the trigger definition. Valid values are

ADD	Add
MOD	Modify
DEL	Purge
DISP	Display
ACT	Activate
DEACT	Deactivate

Calling Parameters (Format A209)

The parameters that compose CALLING-PARMS provides the information needed to complete the actions. These parameters are consistent with the information requested by the online Trigger Maintenance facility and include the following fields (default values are underlined):

Field Name	Length	Description
TRG-DBNR	N5	Trigger file DBnr
FILE-NAME	A32	File name defined in the FFT
CMD-TYPE	A1	Command type that causes trigger to fire: D: delete F: find I: insert R: read U: update *: all commands
FIELD-NAME	A32	Field name defined in the FFT
TRIGGER-PROG	A8	Name of the Natural trigger program
PRIORITY	N3	Trigger priority
PRE-CMD-SELECT	A1	Trigger timing relative to the command execution: N: post-command trigger Y: pre-command trigger
PART-NON-FLAG	A1	Participation flag. Valid values are A: asynchronous (default) N: nonparticipating P: participating
CALLNAT-FLAG	A1	CALLNAT parameters. Valid values are C: control parameters with ACB and response code (default) E: response code only N: no parameters passed X: control parameters with ACBX and response code
RECBUFF-OPTION	A1	Record buffer access. Valid values are A: access only N: no access to record buffer (default) U: updates allowed
ACT-TYPE	A4	Type of activation/deactivation to be processed: PERM: permanent TEMP: temporary
TRG-RSV	A39	Reserved for future use
RESP	N4	Response code returned from the API

Field Name	Length	Description
RETURN-MSG	A68	Text description of the response code
VERSION	N3	GCB version
PRODVERN	N3	Database version
NUCID	P5	Nucleus ID

Sample User Program

```
0010 ************************************************************************
0020 * Application .. ADABAS Triggers Subsystem
0030 * Program ...... UMAINT
0040 * Function ..... Sample call program to call API TRGMAIN that
0050 * maintains trigger definitions.
0060 * (Add,Delete,Modify,Display,Activate,Deactivate).
0070 *
0080 * Parameters ... The following parameters are passed when calling
0090 * the API:
0100 * FUNCTION (A5) Action to perform on Trigger Definition
0110 * Valid values are ADD - Add
0120 * MOD - Modify
0130 * DEL - Delete
0140 * DISP - Display
0150 * ACT - Activate
0160 * DEACT - De-activate
0170 * TRG-DBNR (N5) Trigger File DBnr
0180 * FILE-NAME (A32) File Name defined in the FFT
0190 * CMD-TYPE (A1) Command type that causes trigger to fire
0200 * Valid values are R - Read
0210 * F - Find
0220 * I - Insert
0230 * U - Update
0240 * D - Delete
0250 * * - All commands
0260 * FIELD-NAME (A32) Field Name defined in the FFT
0270 * TRIGGER-PROG (A8) Name of the NATURAL Trigger Program
0280 * PRIORITY (N3) Trigger Priority
0290 * PRE-POST-FLAG (A1) Pre-trigger or Post-trigger
0300 * Valid values are Y - Pre trigger
0310 * N - Post trigger
0320 * PART-NON-FLAG (A1) Participation flag
0330 * Valid values are A - Asyncronous
0340 * N - Non-Participating
0350 * P - Participating
0360 * CALLNAT-FLAG (A1) CALLNAT Parameters
0370 * Valid values are N - No parmeters passed
0380 * E - Response code only
0390 * C - Control Parms and
```

```
0400 * Response code
0410 * RECBUFF-OPTION (A1) Record Buffer Access
0420 * Valid values are A - Access only
0430 * U - Updates allowed
0440 * N - No access to RB
0450 * ACT-TYPE (A4) Type of Activation/De-activation
0460 * Valid values are: TEMP - Temporary
0470 * PERM - Permanent
0480 * VERSION (N3) Version of the GCB
0490 * Valid values are: 201 for v81
0500 * 202 for v82
0510 * PRODVERN (N3) Version of the Adabas
0520 * Valid values are: 812, 813, 814 for v81x
0530 * 822, 823, 824, 825, 826 for v82x
0540 * NUCID (P5) NUCID
0550 * RESP (N4) Response Code returned from the API
0560 * RETURN-MSG (A68) Text description of the response code
0570 ************************************************************************
0580 DEFINE DATA
0590 LOCAL USING TRGPMAIN
0600 LOCAL
0610 01 MAP-MSG (A68)
0620 01 HOLD-FUNCTION (A5)
0630 01 HOLD-PRE-POST-FLAG (A1)
0640 01 PAGE-TITLE (A50)
0650 01 #ATTR (C)
0660 END-DEFINE
0670 RESET CALLING-PARMS MAP-MSG
0680 MOVE 233 TO TRG-DBNR
0690 VERSION := 202
0700 PRODVERN := 822
0710 NUCID := 0
0720 SET KEY PF3
0730 **
0740 ** Request function and required fields
0750 **
0760 REPEAT
0770 INPUT (AD=TMIL'_' CD=NE)
0780 MAP-MSG (IP=OFF AD=O)
0790 / 9T 'API Maintenance of Trigger Definitions' (YEI)
0800 // 'Function.........' FUNCTION 30T '(Add, Del, Mod, Disp, or ".")'
0810 // 'File Name.........' FILE-NAME
0820 / 'Field Name........' FIELD-NAME
0830 / 'Command Type......' CMD-TYPE 30T '(R, F, I, U, D, or *)'
0840 / 'Pre-Command.......' PRE-POST-FLAG 30T '(Y, N, or blank)'
0850 RESET MAP-MSG
0860 **
0870 ** Escape out of here
0880 **
0890 IF FUNCTION = MASK('.') OR *PF-KEY = 'PF3'
0900 ESCAPE BOTTOM
0910 ** Set up Page Titles
```

```
0920 DECIDE ON FIRST VALUE OF FUNCTION
0930 VALUE 'DISP' MOVE 'Display' TO PAGE-TITLE
0940 VALUE 'ADD' MOVE 'Add' TO PAGE-TITLE
0950 VALUE 'MOD' MOVE 'Modify' TO PAGE-TITLE
0960 VALUE 'DEL' MOVE 'Delete' TO PAGE-TITLE
0970 VALUE 'X' ESCAPE BOTTOM
0980 NONE REINPUT 'Invalid Function Code'
0990 END-DECIDE
1000 COMPRESS PAGE-TITLE 'Trigger Definition' INTO PAGE-TITLE
1010 IF FILE-NAME = ' '
1020 REINPUT 'File Name can not be BLANK' MARK *FILE-NAME
1030 **
1040 ** Handle Request to Display a Trigger
1050 **
1060 IF FUNCTION = 'DISP'
1070 DO
1080 PERFORM GET-TRIGGER
1090 INPUT (AD=O CD=NE)
1100 MAP-MSG (IP=OFF AD=O)
1110 / 9T 'API Maintenance of Trigger Definitions' (YEI)
1120 // 10T PAGE-TITLE (IP=OFF)
1130 // 'File name.........' FILE-NAME
1140 / 'Field Name........' FIELD-NAME
1150 / 'Command Type......' CMD-TYPE
1160 // 'Trigger Program...' TRIGGER-PROGRAM
1170 / 'Priority..........' PRIORITY
1180 / 'Pre-Command.......' PRE-POST-FLAG
1190 / 'Trigger Type......' PART-NON-FLAG
1200 / 'CALLNAT Params....' CALLNAT-FLAG
1210 / 'RecBuffer Access..' RECBUFF-OPTION
1220 RESET FUNCTION
1230 MOVE RETURN-MSG TO MAP-MSG
1240 DOEND
1250 **
1260 ** Handle Request to Alter Trigger Definitions
1270 **
1280 IF FUNCTION = 'ADD' OR= 'MOD' OR= 'DEL'
1290 DO
1300 IF (FUNCTION = 'MOD' OR= 'DEL')
1310 PERFORM GET-TRIGGER
1320 IF FUNCTION = 'DEL'
1330 DO
1340 MOVE 'Press ENTER to Delete or PF-3 to Cancel' TO MAP-MSG
1350 MOVE (AD=P) TO #ATTR
1360 DOEND
1370 ELSE DO
1380 MOVE 'Press ENTER to confirm data or PF-3 to Cancel' TO MAP-MSG
1390 MOVE (AD=D CD=NE) TO #ATTR
1400 DOEND
1410 REPEAT
1420 INPUT (AD=TMIL'_' CD=NE)
1430 MAP-MSG (IP=OFF AD=O)
```

```
1440 / 9T 'API Maintenance of Trigger Definitions' (YEI)
1450 // 10T PAGE-TITLE (AD=O IP=OFF)
1460 // 'File Name.........' FILE-NAME (AD=O)
1470 / 'Field Name........' FIELD-NAME (CV=#ATTR)
1480 / 'Command Type......' CMD-TYPE (CV=#ATTR)
1490 // 'Trigger Program...' TRIGGER-PROGRAM (CV=#ATTR)
1500 / 'Priority..........' PRIORITY (CV=#ATTR)
1510 / 'Pre-Command.......' PRE-POST-FLAG (CV=#ATTR)
1520 / 'Trigger Type......' PART-NON-FLAG (CV=#ATTR)
1530 / 'CALLNAT Params....' CALLNAT-FLAG (CV=#ATTR)
1540 / 'RecBuffer Access..' RECBUFF-OPTION (CV=#ATTR)
1550 RESET MAP-MSG
1560 IF *PF-KEY = 'PF3'
1570 DO
1580 MOVE 'Function cancelled per user request' TO MAP-MSG
1590 ESCAPE BOTTOM
1600 DOEND
1610 **
1620 ** Perform the update of data (add, del, or mod)
1630 ** and handle the response
1640 **
1650 CALLNAT 'TRGMAIN' FUNCTION CALLING-PARMS
1660 DECIDE ON FIRST VALUE OF RESP
1670 VALUE 0 MOVE RETURN-MSG TO MAP-MSG
1680 EXAMINE MAP-MSG FOR 'confirmed' REPLACE 'successful'
1690 ESCAPE BOTTOM
1700 VALUE 20 REINPUT WITH TEXT RETURN-MSG MARK *FIELD-NAME
1710 VALUE 23 REINPUT WITH TEXT RETURN-MSG MARK *FIELD-NAME
1720 VALUE 25 REINPUT WITH TEXT RETURN-MSG MARK *CMD-TYPE
1730 * VALUE 37 REINPUT WITH TEXT RETURN-MSG MARK *PRIORITY
1740 * VALUE 38 REINPUT WITH TEXT RETURN-MSG MARK *PRIORITY
1750 VALUE 39 REINPUT WITH TEXT RETURN-MSG MARK *TRIGGER-PROGRAM
1760 VALUE 40 REINPUT WITH TEXT RETURN-MSG MARK *PRE-POST-FLAG
1770 VALUE 41 REINPUT WITH TEXT RETURN-MSG MARK *PART-NON-FLAG
1780 VALUE 42 REINPUT WITH TEXT RETURN-MSG MARK *CALLNAT-FLAG
1790 VALUE 43 REINPUT WITH TEXT RETURN-MSG MARK *RECBUFF-OPTION
1800 NONE MOVE RETURN-MSG TO MAP-MSG
1810 END-DECIDE
1820 LOOP
1830 RESET FUNCTION
1840 DOEND
1850 ** Loop back up to display the starting screen
1860 LOOP
1870 ******************************
1880 ** Subroutine to retrieve the *
1890 ** trigger information *
1900 ******************************
1910 DEFINE SUBROUTINE GET-TRIGGER
1920 MOVE FUNCTION TO HOLD-FUNCTION /* Go get the existing
1930 MOVE 'DISP' TO FUNCTION /* Trigger Information
1940 MOVE PRE-POST-FLAG TO HOLD-PRE-POST-FLAG
1950 CALLNAT 'TRGMAIN' FUNCTION CALLING-PARMS
```

```
1960 MOVE HOLD-FUNCTION TO FUNCTION
1970 IF RESP NE 0
1980 REINPUT WITH TEXT RETURN-MSG
1990 IF RESP = 0
2000 DO
2010 IF PRE-POST-FLAG NE HOLD-PRE-POST-FLAG
2020 AND HOLD-PRE-POST-FLAG NE ' '
2030 DO
2040 MOVE HOLD-PRE-POST-FLAG TO PRE-POST-FLAG
2050 MOVE 'Trigger does not exist' TO MAP-MSG
2060 IF HOLD-PRE-POST-FLAG = 'Y'
2070 COMPRESS 'Pre-' MAP-MSG INTO MAP-MSG
2080 ELSE COMPRESS 'Post-' MAP-MSG INTO MAP-MSG
2090 MOVE HOLD-FUNCTION TO FUNCTION
2100 ESCAPE TOP
2110 DOEND
2120 IF CMD-TYPE = ' '
2130 MOVE '*' TO CMD-TYPE
2140 DOEND
2150 RETURN
2160 *** End of Subroutine ***
2170 END
2180 ** ↩
```

Response Codes

Code	Meaning
000	Function completed successfully.
013	Invalid file-field entry specified. The triggers facility requires access to a file-field table that maps long file names to file numbers and field names to Adabas two-character field identifiers. See section *File-Field Tables* for more information.
016	No trigger definition found with this criteria.
020	Field name must be blank for delete command class.
023	Field name not found for this file. The field name specified does not exist in the file-field table for this file. See section *File-Field Tables* for more information.
025	Invalid command type. See list above for valid values.
027	No field found for this file and command type.
029	No trigger found for this file, command, and field criteria.
037	Priority must be between 1 and 900.
038	Priority cannot be set if Adabas field name is **.
039	Natural subprogram name is invalid.
040	Specify "Y" for pre-trigger and "N" for post-trigger.
041	Trigger type may be "A", "N", or "P" only.

Code	Meaning
042	CALLNAT type may be "C", "E", or "N" only.
043	Record buffer usage may be "A", "N", or "U" only.
044	Invalid parameter combination. No record buffer available with asyncronous triggers.
045	Invalid parameter combination. No record buffer available for pre-triggers for read or find commands.
046	Invalid parameter combination. No record buffer available for delete commands.
047	A trigger already exists for this parameter combination.
048	Invalid request. Trigger is already in the specified state.
052	Change in trigger state not possible now.
103	File name must not be blank.
111	Invalid function code.
112	TYPE must be temp or perm. The type of activation or deactivation of a trigger must be either temporary (lasting for this nucleus session only) or permanent.
1xxx	Function resulted in an Adabas response code 22 (ADARSP022) where *xxx* represents the subcode.
3xxx	Non-zero Adabas response code was returned, where *xxx* is the actual Adabas response code.
9999	Function not successful. Verify parameters and existing trigger definitions. Contact your Software AG technical support representative with questions.

VIII TRGUNLD and TRGLOAD Utilities

10 TRGUNLD and TRGLOAD Utilities

The unload and load utilities for the Adabas triggers and stored procedures facility are part of the online Trigger Maintenance facility and are run in a Natural environment:

Utility	... is used to
TRGUNLD	unload trigger definitions from the Adabas trigger file and related file-field table entries and write them to a work file.
TRGLOAD	load trigger definitions and related file-field tables from the TRGUNLD work file into the trigger file.

The work file for TRGUNLD is a Natural work file (work file 1) that is defined either

- in the batch job; or
- to the Natural environment where the unload utility is running.

The TRGUNLD work file is used as input to the TRGLOAD utility.

A report is prepared at the end of the unload or load summarizing the triggers processed.

Starting a Utility

To invoke a utility:

- Issue the name of the utility (TRGUNLD or TRGLOAD) as a command, optionally followed by a parameter list.

  Parameters are used to limit the triggers to be processed.

  Individual parameter values must be separated by the input delimiter (ID): the default is a comma ",". The input-mode parameter `IM` should be set to delimiter mode `IM=D`.

  When executing a utility as a batch job, the batch job must log on to the SYSTRG library before executing the selected utility and its parameter list.

TRGUNLD

If the TRGULND utility is invoked from the command line in the online system without a parameter list, the following window is presented:

```
17:51:42          ***** A D A B A S TRIGGER MAINTENANCE *****       YYYY-MM-DD
User
     |-------------------------------------------------------------------------|
     | 17:51:46  **** A D A B A S TRIGGER MAINTENANCE ****   YYYY-MM-DD        |
     | -----------------  Trigger  Unload  Utility  -------------------        |
     |                                                                         |
     |       Specify the following information to identify                     |
     |       the Triggers to be unloaded to Work File 1:                       |
     |                                                                         |
     |        File Name ..... ____________________________________             |
     |        Field ......... ____________________________________             |
     |        Pre-triggers .. ____                                             |
     |        Command Type .. __                                               |
     |        Active State .. _                                                |
     |        Trigger Type .. _                                                |
     |                                                                         |
     |       Use 'PF3' to cancel or hit  Enter' when ready                     |
     |-------------------------------------------------------------------------|


Command ==> trgunld
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Field Trigr Admin Procs             FTRG  FDIC  Canc
```

Enter values for the parameters as needed to limit the trigger definitions being unloaded. Any parameters left blank assume the default values.

If no parameters are specified for TRGUNLD, *all* triggers on the trigger file are unloaded and the related file-field table entries as well.

TRGLOAD

If the TRGLOAD utility is invoked from the command line in the online system without a parameter list, the following window is presented:

```
18:13:43          ***** A D A B A S TRIGGER MAINTENANCE *****       YYYY-MM-DD
User DBAU |-------------------------------------------------------|nr 105
          |                                                       |
          |  ---------------  Trigger  Load  Utility  ------------ |
          |                                                       |
          |     Specify the following information to identify     |
          |     the Triggers to be loaded from Work File 1:       |
          |                                                       |
          |        File ...... ____________________________       |
          |        Field ..... ____________________________       |
          |        Replace ... ___                                |
          |        With FFT .. ___                                |
          |                                                       |
          |     Use 'PF3' to cancel or hit  Enter' when ready     |
          |                                                       |
          |-------------------------------------------------------|


Command ==> trgload
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Field Trigr Admin Procs             FTRG  FDIC  Canc
```

If no parameters are specified for TRGLOAD, *all* triggers and file-field table entries found on the TRGUNLD work file are loaded into the trigger file.

Utility Parameters

Wildcard Notation

The following "wildcard" or special character notation can be used in values specified for FILE and FIELD parameters discussed in the following sections for each utility:

Character	Example	Load triggers with file or field names...
*	PERS*	that start with "PERS"
>	PERS>	that have a value greater than "PERS"
<	PERS<	that have a value less than "PERS"

TRGUNLD Parameters

The following parameters are available to limit the triggers unloaded from the trigger file into the work file during TRGUNLD processing (the default value for each parameter is "*" to include all possibilities):

Parameter	Description
FILE	Name of a file found in the trigger definition. Valid values are a file name ("wildcard" notation allowed) * — all files (the default)
FIELD	Name of a field found in the trigger definition. Valid values are a field name ("wildcard" notation allowed) * — all fields (the default)
PRE	Specifies pre-trigger or post-trigger. Valid values are Y — pre-trigger N — post-trigger * — both
CMD	Command type that causes trigger to fire. Valid values are R — read F — find I — insert U — update D — delete * — all command types (the default)
ACT	State of the trigger to be processed. Valid values are A — active D — not active (deactivated) * — both
TYPE	Type of participation of trigger in user's ET logic. Valid values are A — asynchronous N — non-participating P — participating * — all participation types

TRGUNLD Parameter Examples

```
TRGUNLD FILE=EMPLOYEES
```

Specifying the FILE parameter limits the triggers unloaded to those defined for the specified file, in this case the EMPLOYEES file.

```
TRGUNLD FILE=VEHICLES,PRE=Y,ACT=A,TYPE=N
```

Combining parameters further limits the triggers unloaded. In this case, only active, non-participating pre-triggers defined for the VEHICLES file are unloaded. Note that the parameters must be separated by the input delimiter as specified in the ID parameter in the Natural environment.

TRGLOAD Parameters

The following parameters are available to limit the triggers loaded from the TRGUNLD work file to the trigger file during TRGLOAD processing:

Parameter	Description
FILE	Name of a file found in the trigger definition. Valid values are a file name ("wildcard" notation allowed) * : all files (the default)
FIELD	Name of a field found in the trigger definition. Valid values are a field name ("wildcard" notation allowed) * : all fields (the default)
FFT	Request that the file-field table entry found on the TRGUNLD input data set be loaded with the trigger. Valid values are Y : yes (the default) N : no
REPLACE	Request that a trigger definition being loaded that already exists on the database replace the old one. Valid values are Y : yes: replace the existing definition with the new one N : no (the default): do not replace the existing definition and return an error message

TRGLOAD Parameter Examples

```
TRGLOAD FILE=EMP*
```

Load into the trigger file all file-field table entries and triggers found on the TRGUNLD input work file for files starting with "EMP".

```
TRGLOAD FFT=N,REPLACE=Y
```

Ignore any file-field table entries read from the work file. If any trigger definition read from the work file has the same specifications as an existing trigger definition, replace the existing definition with the definition from the work file.

End of Processing Reports

Unload Report (TRGUNLD)

A report is written out at the end of the unload process stating the number of trigger definitions and the number of file-field table entries unloaded.

Example

The first page of the report indicates the source file and database for unloading the triggers and summarizes the selection criteria. Following this is a list of each record being unloaded into the work file:

```
18:12:00            *****  TRIGGERS UNLOAD UTILITY *****            YYYY-MM-DD
DBAUSER
Unloading from Trigger File     12 on Database    105
Rec  File  Details
---  ----- ------------------------------------------------------------
Unloading Pre and Post Triggers for file(s) *
and field name(s) * for all command types

FFT     45 ZB-FIELD (ZB,2,A)
FFT     45 ZF-FIELD (ZF,2,A)
FFT     45 ZZ-FIELD (ZZ,4,P)
TRG     45 CMD=R FLD=ZB PRTY=01 PGM=RBEGIMU PRE=S TYP=P PRM=C RB=U
GRP      4 Group Record
FFT      4 ADDRESS-LINE (AI,20,A)
FFT      4 AREA-CODE (AN,6,A)
FFT      4 BIRTH (AH,6,U)
.
.
.
```

The end of the report lists a summary count of records written to the work file. The physical work file written includes these records plus two additional records used for internal purposes.

```
18:13:04           *****  TRIGGERS UNLOAD UTILITY *****           YYYY-MM-DD
 DBAUSER
 Unloading from Trigger File     12 on Database    105
 Rec  File  Details
 ---  ----- ------------------------------------------------------------
 FFT      7 NA-FIELD (NA,40,A)
 FFT      7 NT-SUPER (NT,110,A)
 FFT      7 TI-FIELD (TI,70,A)
 FFT      7 TY-FIELD (TY,10,A)
 TRG      7 CMD=R FLD=** PRTY=90 PGM=NACNN200 PRE=S TYP=A PRM=C RB=N

 Total records written to Work File 1

 File-Field Table Entries: ...    141
 Group Control Records: ......      2
 Trigger Definitions: ........     13

 Total Records: ..............    156

 ***  TRGUNLD completed successfully.  ***
```

The final page of the report summarizes the triggers unloaded by categories. This report is useful for cross-checking with the TRGLOAD utility.

```
**** Trigger Unload Statistics ****

Number of Triggers Unloaded by Categories

Pre or Post :           Pre:      7
                       Post:      6

Trigger Type:  Asynchronous:      6
              Participating:      3
          Non-Participating:      4

Command Type:          READ:      6
                       FIND:      0
                     INSERT:      0
                     DELETE:      2
                     UPDATE:      1
                      *ANY*:      4

Field Criteria:       *ANY*:      4
                    Specific:     9
```

Load Report (TRGLOAD)

A report is written at the end of the unload process, stating the number of trigger definitions and the number of file-field table entries loaded.

Example

The first page of the report indicates the target database for loading the triggers, the source database where the work file was created, and a summary of the selection criteria. Following this is a list of each record being loaded into the work file.

```
20:05:05            *****  TRIGGERS LOAD UTILITY  *****           YYYY-MM-DD
DBAUSER
Rec  File  Details
---  ----- ------------------------------------------------------------

Loading Trigger Definitions into database    106 file     14
from data set unloaded from database    105 file     12
created on 1999-07-26 20:04 from version  711

FFT     50 AA-FIELD (AA,8,A)
FFT     50 AC-FIELD (AC,20,A)
FFT     50 AD-FIELD (AD,20,A)
FFT     50 AE-FIELD (AE,20,A)
FFT     50 AF-FIELD (AF,1,A)
FFT     50 AG-FIELD (AG,1,A)
FFT     50 AH-FIELD (AH,6,U)
FFT     50 AI-FIELD (AI,20,A)
FFT     50 AJ-FIELD (AJ,20,A)
FFT     50 AK-FIELD (AK,10,A)
.
.
.
```

The end of the report lists a summary count of records read from the work file and how they were loaded into the database.

```
20:08:43            *****  TRIGGERS LOAD UTILITY  *****           YYYY-MM-DD
 DBAUSER
 Rec  File  Details
 ---  ----- ------------------------------------------------------------
 TRG      5 CMD=R FLD=AA PGM=CAROL PRE=P TYP=A PRM=C RB=N

 Total records loaded from Work File 1

 File-Field Table Entries: ...    109
 Group Control Records: ......      1
 Trigger Definitions: ........     65

 Total Records Loaded: .......    175
 Records found with errors ...      0
 Total Records Read: .........    178

 ***  TRGLOAD completed successfully.  ***
```

The final page of the report summarizes the triggers loaded by categories. This report is useful for cross-checking with the TRGUNLD utility.

```
******* Trigger Load Statistics *******

Number of Triggers Loaded by Categories

Pre or Post :           Pre:     34
                       Post:     31

Trigger Type:  Asynchronous:     23
              Participating:     32
          Non-Participating:     10

Command Type:          READ:     25
                       FIND:      6
                     INSERT:     14
                     DELETE:      2
                     UPDATE:      3
                      *ANY*:     15

Field Criteria:       *ANY*:     19
                    Specific:     46
```

Utility Response Codes

Code	Meaning
000	Function completed successfully.
013	Invalid file-field table entry specified. Triggers requires access to a file-field table that maps long file names to file numbers and field names to Adabas two-character field identifiers. See section *File-Field Tables* for more information.
016	No trigger definition found with this criteria.
023	Field name not found for this file. The field name specified does not exist in the file-field table for this file. See section *File-Field Tables* for more information.
025	Invalid command type. See list above for valid values.
027	No field found for this file and command type.
029	No trigger found for this file, command, and field criteria.
039	Natural subprogram name is invalid.
041	Trigger type may be "A", "N", or "P" only.
042	CALLNAT type may be "C", "E", or "N" only.
043	Record buffer usage may be "A", "N", or "U" only.
047	Trigger already exists with this criteria.
103	File name must not be blank.
110	File-field table entry already exists for file.
1xxx	Function resulted in an Adabas response code 22 (ADARSP022) where "xxx" represents the subcode.
3xxx	Nonzero Adabas response code was returned, where "xxx" is the actual Adabas response code.
9999	Function not successful. Verify parameters and existing trigger definitions. For more information, contact your Software AG technical support representative.

A Examples

This document contains the example programs and data areas listed in the following table. Source code is provided during the installation procedure and is located in the library SYSSPT.

See **Examples** for stored procedure link routines STPLNK01, STPLNK02, and STPLNK03.

Name	Description
SAMPINT1	This program issues a command that results in a trigger being fired and performs additional processing whenever an update or store record is written for a specific file.
SAMPPRC1	A stored procedure version of SAMPINT1. Data is passed as a parameter and information extracted from this parameter is returned.
SAMPP001	A stored procedure invoked from SAMPPRC1. It uses information from the input parameter to extract and create additional information to be returned to the caller.
STPLCB	An Adabas ACB control block layout used as a local data area (LDA) in example routines such as STPLNK02 and STPLNK03.
STPLCBE	An Adabas ACBX control block layout used as a local data area (LDA)
STPLRBE	An example layout of the parameter data area (PDA) that must be used when the record buffer extraction routine (STPRBE) is called.
STPUTPRM	An example of the parameter data area passed to the subprogram STPUTRAK.
STPUTRAK	The routine invoked by the Natural trigger driver when a request is received to execute a procedure. The routine is called only if the log trigger activity option in the Trigger Maintenance profile is set to `active'. The name of this routine must not be changed.
STPAPARM	An example layout of the parameter data area (PDA) that is passed from the Natural trigger driver when a procedure is invoked.
STPAPRM1	An example layout of the parameter data area (PDA) that is passed from the Natural trigger driver when a procedure is being invoked and STPUTRAK has requested that extended information also be passed. The extended area length is 250 bytes. It is maintained as part of the trigger driver's global data area (GDA) and is kept intact across each invocation of a procedure by the trigger driver.
SAMP0001	The procedure that is invoked as the result of a trigger being fired for the commands originating from SAMPINT1.
SAMP0002	The procedure invoked for any command issued to the CONTACTS file, which is used as an example file in the SAMPINT1 program. The procedure audits all commands by writing a message to CMPRINT and creating a log record on the audit file.
SAMP0003	Like SAMP0002, this procedure audits commands issued to the CONTACTS file; however, it audits only participating commands.
SAMP0004	The procedure invoked through the execution of the SAMPREF1 routine for any deletes to the VEHICLES file. The example application is for referential logic type Restrict.
SAMP0005	The procedure invoked through the execution of the SAMPREF2 routine for any updates to the EMPLOYEES file. This is an example of referential integrity type Cascade.
SAMPREF1	This routine issues commands to invoke triggers that will perform some Referential integrity validation of the VEHICLES file. The associated procedure is SAMP0004, which performs Restrict checking on the records being deleted.

Name	Description
SAMPREF2	This routine issues commands to invoke triggers that will perform some Referential integrity checking for updates to the primary key on the EMPLOYEES file. The associated procedure is SAMP0005, which cascades any updates to the primary key to the foreign keys on the VEHICLES and MISCELLANEOUS files.

SAMPINT1

```
0010 ************************************************************************
0020 *  Application: Adabas Triggers
0030 *      Program: SAMPINT1
0040 *     Function: Routine to add names to file (CONTACTS). A trigger
0050 *               will be established to perform additional processing
0060 *               whenever the name is updated or added.
0070 * Trigger Defn: The definition on the trigger file (one for an update
0080 *               and one for the STORE/INSERT) is as follows:
0090 *                 File Number ....... 11
0100 *                 File Name ......... CONTACTS
0110 *                 Command Type ...... Update + Insert
0120 *                 Long Field Name ... CONTACT-NAME
0130 *                 Adabas Field ...... LE
0140 *                 Field Prty/Seq .... 10_
0150 *               Procedure Information
0160 *                 Name (Subpgm)...... SAMP0001
0170 *                 Pre Cmd Select .... Y  (Pre)
0180 *                 Trigger Type ...... P  (Participating)
0190 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0200 *                 RecBuffer Access .. U  (May be Updated)
0210 *
0220 *               NOTE: An additional trigger also exists for this file
0230 *                     whereby any commands to the file will result in a
0240 *                     trigger being fired. This definition is:
0250 *
0260 *                 File Number ....... 11
0270 *                 File Name ......... CONTACTS
0280 *                 Command Type ...... ** All Command **
0290 *                 Long Field Name ... ** Any Field **
0300 *                 Adabas Field ...... **
0310 *                 Field Prty/Seq .... ___
0320 *               Procedure Information
0330 *                 Name (Subpgm)...... SAMP0002
0340 *                 Pre Cmd Select .... Y  (Pre)
0350 *                 Trigger Type ...... A  (Asynchronous)
0360 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0370 *                 RecBuffer Access .. N  (No RecBuff Access)
0380 *
0390 *       Author: Adabas Development
```

```
0400 *         Date: December 1995
0410 ************************************************************************
0420 DEFINE DATA LOCAL
0430  01  #NAME     (A60)
0440  01  RESP       (N4)
0450  01  CONTACTS VIEW OF CONTACTS    /* file 11 for this example
0460    02  CONTACT-NAME                /* field LE,A,60,NU,DE
0470    02  CONTACT-UPPER               /* field LO,A,60,NU,DE
0480    02  KEYWORDS (20)               /* field LM,A,20,NU,MU
0490 END-DEFINE
0500 *
0510 REPEAT
0520 *
0530 INPUT (AD=WMIL'_' CD=NE)
0540      'Trigger Example for Data Consistancy' (YEI)
0550   // 'Name ...' (TU) #NAME
0560 *
0570 IF #NAME = MASK('.')               /* exit?
0580   STOP                             /* yes
0590 IF #NAME = ' '                     /* name must be specified
0600   REINPUT 'Invalid Name specified'
0610 *
0620 FIND CONTACTS WITH CONTACT-NAME = #NAME /* Find the name
0630   IF NO RECORDS FOUND              /* does it exist?
0640     DO                             /* no, so we should add it
0650 *
0660 *     Although only the CONTACT-NAME is being added, the other fields to
0670 *     be used in the Store are included. In this way, the format buffer
0680 *     and record buffer have reference to these files, since this example
0690 *     results in a pre-trigger that sets the values in these fields.
0700 *     Of course a post-trigger would also have worked; however, it would
0710 *     have been necessary for the procedure to do an additional read and
0720 *     update of the record. In this example, better performance is
0730 *     achieved with a pre-trigger.
0740 *
0750      RESET CONTACT-UPPER KEYWORDS(*)
0760      MOVE #NAME TO CONTACT-NAME    /* move value into view
0770 *
0780      STORE CONTACTS                /* insert the new record on the file
0790      END TRANSACTION               /* and commit the transaction
0800      IF *ISN(0780) = 0             /* we can check that a new ISN exists
0810        DO
0820          WRITE 'Store was unsuccessful' *ISN(0780)
0830          ESCAPE BOTTOM
0840        DOEND
0850      GET CONTACTS *ISN(0780)       /* now we refresh the record buffer
0860      INPUT (AD=O CD=TU)            /* and show the results of the Store
0870            '*** Results ***' (YEI)
0880          / 'Name ......' (GRI) CONTACT-NAME
0890          / 'Upper .....' (GR)  CONTACT-UPPER
0900          / 'Keywords ..' (GR) KEYWORDS (1:3)
0910          / '           ' KEYWORDS (4:6)
```

```
0920          / '           ' KEYWORDS (7:9)
0930          / '           ' KEYWORDS (10:12)
0940          / '           ' KEYWORDS (13:15)
0950          / '           ' KEYWORDS (16:18)
0960          / '           ' KEYWORDS (19:20)
0970      ESCAPE BOTTOM               /* and exit the Add Record logic
0980    DOEND
0990  SET KEY PF3 PF5                /* activate a couple of PF-keys
1000  INPUT (AD=O CD=TU)              /* allow the name to be changed
1010            '*** Make required Changes and PRESS PF5 to Update:' (YEI)
1020          / 'Name ......' (TU) CONTACT-NAME (AD=WMIL'_' CD=NE)
1030          / 'Upper .....' (TU) CONTACT-UPPER
1040          / 'Keywords ..' (TU) KEYWORDS (1:3)
1050          / '           ' KEYWORDS (4:6)
1060          / '           ' KEYWORDS (7:9)
1070          / '           ' KEYWORDS (10:12)
1080          / '           ' KEYWORDS (13:15)
1090          / '           ' KEYWORDS (16:18)
1100          / '           ' KEYWORDS (19:20)
1110    IF *PF-KEY = 'PF5'
1120      DO
1130        UPDATE(0620)              /* do the modification
1140        GET CONTACTS *ISN(0620)   /* now we refresh the record buffer
1150        INPUT (AD=O CD=TU)        /* and show the results of the update
1160              '*** Results of the Update ***' (YEI)
1170            / 'Name ......' (GRI) CONTACT-NAME
1180            / 'Upper .....' (GR)  CONTACT-UPPER
1190            / 'Keywords ..' (GR) KEYWORDS (1:3)
1200            / '           ' KEYWORDS (4:6)
1210            / '           ' KEYWORDS (7:9)
1220            / '           ' KEYWORDS (10:12)
1230            / '           ' KEYWORDS (13:15)
1240            / '           ' KEYWORDS (16:18)
1250            / '           ' KEYWORDS (19:20)
1260        MOVE CONTACT-NAME TO #NAME /* and reset the name
1270        ESCAPE BOTTOM             /* and exit the Update Record logic
1280      DOEND
1290  ESCAPE BOTTOM                   /* no update done
1300 CLOSE LOOP(0620)
1310 END TRANSACTION                  /* my job is to confirm and release
1320 CLOSE LOOP(0510)
1330 *
1340 END
```

SAMPPRC1

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *      Program: SAMPPRC1
0040 *     Function: Routine to input a name and then invoke a stored
0050 *               procedure to populate additional fields based on the
0060 *               name passed.
0070 *
0080 *       Author: Adabas Development
0090 *         Date: December 1995
0100 ************************************************************************
0110 DEFINE DATA LOCAL
0120  01  #NAME     (A60)
0130  01  RESP       (N4)
0140  01  CONTACTS-INFORMATION         /* could be a file
0150    02  CONTACT-NAME  (A60)
0160    02  REDEFINE CONTACT-NAME
0170      03  PARM1     (A1/60)
0180    02  CONTACT-UPPER (A60)
0190    02  REDEFINE CONTACT-UPPER
0200      03  PARM2     (A1/60)
0210    02  KEYWORDS (A20/1:20)
0220    02  REDEFINE KEYWORDS
0230      03  PARM3    (A1/400)
0240  01  LINK-ROUTINE-PARMS           /* parameters for the link routine
0250    02  P-FUNC         (A1)
0260    02  P-PROC         (A8)        /* SAMP0001
0270    02  P-OPTIONS      (A8)        /* 'PCU'
0280    02  P-LEN        (P3/5)        /* lengths for the parameters
0290    02  P-MSG         (A72)        /* response message
0300    02  P-RESP         (N4)        /* response code
0310 END-DEFINE
0320 *
0330 MOVE '2'        TO P-FUNC         /* function....not relevant for this
0340 MOVE 'SAMPP001' TO P-PROC         /* procedure name
0350 MOVE 'NCU'      TO P-OPTIONS      /* non-partic + ctrl parms + upd RB
0360 MOVE 60         TO P-LEN(1) P-LEN(2)
0370 MOVE 200        TO P-LEN(3)
0380 MOVE 100        TO P-LEN(4) P-LEN(5)
0390 RESET P-MSG P-RESP
0400 *
0410 REPEAT
0420 *
0430 * In this example, the routine prompts the end user for an organization
0440 * name and in response, extracts some keywords from the value.
0450 * This is similar to SAMPINT1 (except that no file is being used) and
0460 * is a possible alternative to it.
```

```
0470 *
0480 INPUT (AD=WMIL'_' CD=NE)
0490 'Stored Procedure Example for Data Consistency' (YEI)
0500 // 'Name ...' (TU) #NAME
0510 *
0520 IF #NAME = MASK('.') /* exit?
0530 STOP /* yes
0540 IF #NAME = ' ' /* name must be specified
0550 REINPUT 'Invalid Name specified'
0560 *
0570 RESET CONTACT-UPPER KEYWORDS(*)
0580 MOVE #NAME TO CONTACT-NAME /* move value into view
0590 *
0600 CALLNAT 'STPLNK03' P-FUNC P-PROC P-OPTIONS P-LEN(1) PARM1(1:60)
0610 P-LEN(2) PARM2(1:60) P-LEN(3) PARM3(1:200)
0620 P-LEN(4) PARM3(201:300) P-LEN(5) PARM3(301:400)
0630 P-MSG P-RESP
0640 *
0650 INPUT (AD=O CD=TU) /* and show the results of the Store
0660 '*** Results ***' (YEI)
0670 / 'Name ......' (GRI) CONTACT-NAME
0680 / 'Upper .....' (GR) CONTACT-UPPER
0690 / 'Keywords ..' (GR) KEYWORDS (1:3)
0700 / ' ' KEYWORDS (4:6)
0710 / ' ' KEYWORDS (7:9)
0720 / ' ' KEYWORDS (10:12)
0730 / ' ' KEYWORDS (13:15)
0740 / ' ' KEYWORDS (16:18)
0750 / ' ' KEYWORDS (19:20)
0760 *
0770 CLOSE LOOP(0410)
0780 *
0790 END
    ↩
```

SAMPP001

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *  Subprogram : SAMPP001
0040 *  Author     : Adabas Development
0050 *  Date       : August 1995
0060 *  Function   : Sample routine of processing by a procedure
0070 *  Remarks    : This routine converts a name into uppercase and extracts
0080 *               all the keywords associated with it. Once processing is
0090 *               completed, control is returned to the caller.
0100 *
0110 *               Parameter RESP must be set to zero if processing is
```

```
0120 *               successful.
0130 *
0140 *  Parameters : Name1        (A60)
0150 *               Name2        (A60)
0160 *               Keyword(A20/01:10)
0170 *               Keyword(A20/11:15)
0180 *               Keyword(A20/16:20)
0190 *
0200 *  Rec Buffer : The record buffer will be available for update via a
0210 *               CALL to the external routine STPRBE.
0220 *
0230 ************************************************************************
0240 DEFINE DATA PARAMETER USING STPAPARM
0250            LOCAL     USING STPLRBE      /* parms for the Call routine
0260            LOCAL
0270  01  REC-BUFFER(A20/1:26)               /* max rec buffer passed to STPRBE
0280  01  REDEFINE REC-BUFFER                /* redefine this to get the def.
0290    02  INPUT-NAME   (A60)
0300    02  OUTPUT-NAME  (A60)
0310    02  KEYWORDS(A20/1:20)
0320  01  FUNC            (A4)
0330  01  SUB             (I2)
0340  01  SUB1            (I2)
0350  01  SUB2            (I2)
0360  01  W-UPPER        (A61)
0370  01  REDEFINE W-UPPER
0380    02  #UPPER       (A60)
0390    02  REDEFINE #UPPER
0400      03  CHAR   (A1/1:60)
0410  01  #KEYS     (A40/1:20)
0420 END-DEFINE
0430 *
0440 *  Option below is to audit any procedure activity.
0450 *
0460 * CALLNAT 'SAMP0002' REQ-AREA RESP
0470 *
0480 * Since the record buffer information is available to us, we can
0490 * now call the record buffer extraction routine (STPRBE) to obtain
0500 * the contents of the buffer.
0510 *
0520 * Function 'GR' -- GET RB Value using RB offset + length
0530 *     This enables the caller to obtain information based on a
0540 *     certain location; hence, RBE-OFFSET specifies the start
0550 *     position, and RBE-LENGTH specifies the length.
0560 *
0570 MOVE 1   TO RBE-OFFSET               /* start at the beginning
0580 MOVE 520 TO RBE-LENGTH               /* for a max length of 520 bytes
0590 MOVE 'GR' TO FUNC
0600 CALL 'STPRBE' 'GR' RBE-AREA REC-BUFFER(1)
0610 IF RBE-RESP NE 0
0620   PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
0630         RBE-ERROR 'subcode' RBE-SUBCODE 'for func GR'
```

```
0640   MOVE RBE-RESP TO RESP
0650   ESCAPE ROUTINE
0660 END-IF
0670 * PERFORM PRINT-REC-BUFFER            /* option to print the parms
0680 *
0690 * Change all lowercase to UPPERcase
0700 *
0710 MOVE INPUT-NAME TO #UPPER
0720 *
0730 EXAMINE #UPPER AND TRANSLATE INTO UPPER CASE
0740 *
0750 MOVE #UPPER TO OUTPUT-NAME            /* save the uppercase name
0760 *
0770 FOR SUB 1 60                          /* loop to remove all special chars.
0780   IF CHAR(SUB) = MASK(S)
0790     MOVE ' ' TO CHAR(SUB)
0800     ESCAPE TOP
0810   END-IF
0820 END-FOR
0830 *
0840 * We are now ready to extract keywords from our name. This sample is
0850 * very basic and may be made as complex as required.
0860 * This routine assumes a max. length of 20 and a max. num. of 20 keywords
0870 *
0880 EXAMINE FULL W-UPPER FOR FULL ' A '   REPLACE ' '
0890 EXAMINE FULL W-UPPER FOR FULL ' AND ' REPLACE ' '
0900 EXAMINE FULL W-UPPER FOR FULL ' AS '  REPLACE ' '
0910 EXAMINE FULL W-UPPER FOR FULL ' AT '  REPLACE ' '
0920 EXAMINE FULL W-UPPER FOR FULL ' ARE ' REPLACE ' '
0930 EXAMINE FULL W-UPPER FOR FULL ' BE '  REPLACE ' '
0940 EXAMINE FULL W-UPPER FOR FULL ' DO '  REPLACE ' '
0950 EXAMINE FULL W-UPPER FOR FULL ' FOR ' REPLACE ' '
0960 EXAMINE FULL W-UPPER FOR FULL ' HERE ' REPLACE ' '
0970 EXAMINE FULL W-UPPER FOR FULL ' IF '  REPLACE ' '
0980 EXAMINE FULL W-UPPER FOR FULL ' IN '  REPLACE ' '
0990 EXAMINE FULL W-UPPER FOR FULL ' IS '  REPLACE ' '
1000 EXAMINE FULL W-UPPER FOR FULL ' IT '  REPLACE ' '
1010 EXAMINE FULL W-UPPER FOR FULL ' OF '  REPLACE ' '
1020 EXAMINE FULL W-UPPER FOR FULL ' ON '  REPLACE ' '
1030 EXAMINE FULL W-UPPER FOR FULL ' OR '  REPLACE ' '
1040 EXAMINE FULL W-UPPER FOR FULL ' TO '  REPLACE ' '
1050 EXAMINE FULL W-UPPER FOR FULL ' THE ' REPLACE ' '
1060 EXAMINE FULL W-UPPER FOR FULL ' TOO ' REPLACE ' '
1070 EXAMINE FULL W-UPPER FOR FULL ' WAS ' REPLACE ' '
1080 EXAMINE FULL W-UPPER FOR FULL ' WITH ' REPLACE ' '
1090 EXAMINE #UPPER FOR FULL ' ' REPLACE ',' /* put delimiters in the string
1100 *
1110 RESET KEYWORDS(*)
1120 STACK TOP DATA #UPPER                   /* now we will separate each word
1130 INPUT (AD=I IP=ON) #KEYS(01:03) / #KEYS(04:06) / #KEYS(07:09)
1140                  / #KEYS(10:12) /* #KEYS(13:15) / #KEYS(16:18)
1150                  / #KEYS(19:20)
```

```
1160 *
1170 MOVE 1 TO SUB2
1180 MOVE #KEYS(1) TO KEYWORDS(1)
1190 FOR SUB 2 20                              /* now we remove all duplicates
1200   FOR SUB1 1 SUB
1210     IF #KEYS(SUB) = KEYWORDS(SUB1)
1220       RESET #KEYS(SUB)
1230     END-IF
1240   END-FOR
1250   IF #KEYS(SUB) NE ' '
1260     ADD 1 TO SUB2
1270     MOVE #KEYS(SUB) TO KEYWORDS(SUB2)   /* and finally save the value
1280   END-IF
1290 END-FOR
1300 *
1310 * Function 'UR' -- Update RB value using RB offset + length
1320 *     This enables the caller to change information based on a
1330 *     certain location; hence, RBE-OFFSET specifies the start
1340 *     position and RBE-LENGTH specified the length.
1350 *
1360 * PERFORM PRINT-REC-BUFFER              /* print the final results
1370 MOVE 1   TO RBE-OFFSET                  /* start at the beginning
1380 MOVE 520 TO RBE-LENGTH                /* for a max. length of 520 bytes
1390 MOVE 'UR' TO FUNC                     /* req to update all changes
1400 CALL 'STPRBE' 'UR' RBE-AREA REC-BUFFER(1)
1410 IF RBE-RESP NE 0
1420   PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
1430        RBE-ERROR 'subcode' RBE-SUBCODE 'for func UR'
1440   MOVE RBE-RESP TO RESP
1450   ESCAPE ROUTINE
1460 END-IF
1470 *
1480 * Return to the caller: everything went okay
1490 *
1500 ESCAPE ROUTINE
1510 *
1520 DEFINE SUBROUTINE PRINT-REC-BUFFER
1530 *-----------------------------------------------------------------------*
1540 *
1550 *  For testing purposes, display the information returned from STPRBE
1560 *  This routine assumes a maximum of three subsystems running.
1570 *
1580 *-----------------------------------------------------------------------*
1590   DECIDE ON FIRST VALUE OF RQ-TASK
1600     VALUE '01'
1610       WRITE (1) NOTITLE NOHDR (AD=L CD=TU)
1620           '**** RECORD BUFFER EXTRACTION: Function' FUNC '****'
1630           'Stored Procedure RBE' *PROGRAM '****'
1640         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1650         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1660         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1670         / '  Message .......' (TU) RBE-MSG(AL=60)
```

```
1680         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1690         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
1700     VALUE '02'
1710       WRITE (2) NOTITLE NOHDR (AD=L CD=TU)
1720           '**** RECORD BUFFER EXTRACTION: Function' FUNC '****'
1730           'Stored Procedure RBE' *PROGRAM '****'
1740         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1750         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1760         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1770         / '  Message .......' (TU) RBE-MSG(AL=60)
1780         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1790         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
1800     VALUE '03'
1810       WRITE (3) NOTITLE NOHDR (AD=L CD=TU)
1820           '**** RECORD BUFFER EXTRACTION: Function' FUNC '****'
1830           'Stored Procedure RBE' *PROGRAM '****'
1840         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1850         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1860         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1870         / '  Message .......' (TU) RBE-MSG(AL=60)
1880         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1890         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
1900     NONE
1910       WRITE NOTITLE NOHDR (AD=L CD=TU)
1920           '**** RECORD BUFFER EXTRACTION: Function' FUNC '****'
1930           'Stored Procedure RBE' *PROGRAM '****'
1940         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1950         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1960         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1970         / '  Message .......' (TU) RBE-MSG(AL=60)
1980         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1990         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
2000   END-DECIDE
2010 *
2020 END-SUBROUTINE
2030 *
2040 END
```

STPLCB

```
0010 ************************************************************************
0020 **
0030 **Local data area  'STPLCB'
0040 **describes Adabas control block
0050 **
0060 ************************************************************************
0070 DEFINE DATA LOCAL
0080 1 CB         (B80)
```

```
0090 1 REDEFINE CB
0100 2 CB-DSECT             /* ACB definition
0110 3 CB-CALL-TYPE(B1)
0120 3 CB-HOST-ID  (B1)
0130 2 CB-CMD      (A2)     /* command code
0140 2 CB-CID      (A4)     /* command ID
0150 2 CB-FILE     (B2)         /* file number
0160 2 REDEFINE CB-FILE
0170 3 CB-DBID     (B1)         /* one-byte DBNR
0180 3 CB-FNR      (B1)         /* one-byte FNR
0190 2 CB-RSP      (B2)         /* response code
0200 2 CB-ISN(B4)               /* ISN value
0210 2 CB-ISLL(B4)              /* ISN lower limit
0220 2 CB-ISQ(B4)               /* ISN quantity
0230 2 CB-FBL(B2)               /* format buffer length
0240 2 CB-RBL(B2)               /* record buffer length
0250 2 CB-SBL(B2)               /* search buffer length
0260 2 CB-VBL(B2)               /* value buffer length
0270 2 CB-IBL(B2)               /* ISN buffer length
0280 2 CB-CO1(A1)               /* command option 1
0290 2 CB-CO2(A1)               /* command option 2
0300 2 CB-ADD1(A8)              /* additions 1
0310 2 CB-ADD2(A4)              /* additions 2
0320 2 CB-ADD3(A8)              /* additions 3
0330 2 CB-ADD4(A8)              /* additions 4
0340 2 CB-ADD5(A8)              /* additions 5, reserved
0350 2 CB-CT(B4)                /* command time
0360 2 CB-UA(B4)                /* user area
0370 ************************************************************************
0380 ******* END OF LOCAL DATA AREA ***************************************
0390 ************************************************************************
```

STPLCBE

```
0010   * * ******************************** * **** ********************************
0020   * *                                                                      ↩

0030   * * LOCAL DATA AREA  'STPLCBE'                                          ↩

0040   * * DESCRIBES ADABAS CONTROL BLOCK                                      ↩

0050   * * EXTENDED                                                           ↩

0060   * *                                                                    ↩

0070   * * ******************************** * **** ********************************
0080     1 CB                               B  192                            ↩

0090   R 1 CB                                                                 ↩
```

```
0100     2 CB-DSECT                              /* ACBE DEFINITION            ↩
0110     3 CB-TYPE                          B    1                               ↩
0120     3 CB-HOST                          A    1                               ↩
0130     2 CB-VERSION                       A    2 /* ACBE VERSION              ↩
0140     2 CB-LENGTH                        B    2 /* ACBE LENGTH               ↩
0150     2 CB-CMD                           A    2 /* COMMAND                   ↩
0160     2 CB-NUCID                         B    2 /* NUCID                     ↩
0170     2 CB-RSP                           B    2 /* RESPONSE CODE             ↩
0180     2 CB-CID                           B    4 /* COMMAND ID                ↩
0190     2 CB-DBID                          B    4 /* DBID                      ↩
0200     2 CB-FNR                           B    4 /* FILE NUMBER               ↩
0210     2 CB-RESERVED-1                    B    4 /* UNUSED
0220     2 CB-ISN                           B    4 /* ISN VALUE                 ↩
0230     2 CB-RESERVED-2                     B    4 /* UNUSED
0240     2 CB-ISLL                           B    4 /* ISN LOWER LIMIT
0250     2 CB-RESERVED-3                     B    4 /* UNUSED
0260     2 CB-ISQ                            B    4 /* ISN QUANTITY
0270     2 CB-CO1                            A    1 /* COMMAND OPTION 1
0280     2 CB-CO2                            A    1 /* COMMAND OPTION 2
0290     2 CB-CO3                            A    1 /* COMMAND OPTION 3
0300     2 CB-CO4                            A    1 /* COMMAND OPTION 4
0310     2 CB-CO5                            A    1 /* COMMAND OPTION 5
0320     2 CB-CO6                            A    1 /* COMMAND OPTION 6
0330     2 CB-CO7                            A    1 /* COMMAND OPTION 7
0340     2 CB-CO8                            A    1 /* COMMAND OPTION 8
0350     2 CB-ADD1                           A    8 /* ADDITIONS 1
0360     2 CB-ADD2                           B    4 /* ADDITIONS 2
0370     2 CB-ADD3                           A    8 /* ADDITIONS 3
0380     2 CB-ADD4                           A    8 /* ADDITIONS 4
0390     2 CB-ADD5                           B    8 /* ADDITIONS 5
0400     2 CB-ADD6                           A    8 /* ADDITIONS 6
0410     2 CB-RESERVED-4                     B    4 /* RESERVED
0420     2 CB-ERROR                          B   16 /* SUPPLEMENTAL ERROR INFO
0430   R 2 CB-ERROR
0440     3 CB-ERROR-G                        B    4 /* OFFSET IN BUFFER 64-BIT
0450     3 CB-ERROR-A                       B    4 /* OFFSET IN BUFFER 32-BIT    ↩
0460     3 CB-ERROR-B                       A    2 /* ERROR CHARACTER FIELD      ↩
```

```
0470     3 CB-ERROR-C                        B    2 /* SUBCODE                    ↩
0480     3 CB-ERROR-D                        A    1 /* ERROR BUFFER ID            ↩
0490     3 CB-ERROR-E                        B    3 /* BUFFER SEQ. NUMBER         ↩
0500     2 CB-SUB                            B    8 /* SUBCOMPONENT ERROR INFO    ↩
0510   R 2 CB-SUB                                                                  ↩
0520     3 CB-SUB-R                          B    2 /* SUBCOMPONENT RSP CODE      ↩
0530     3 CB-SUB-S                          B    2 /* SUBCOMPONENT REASON CD     ↩
0540     3 CB-SUB-T                          A    4 /* SUBCOMPONENT ERROR TEXT    ↩
0550     2 CB-LCMP                           B    8 /* COMPRESSED RECORD LNGTH    ↩
0560     2 CB-LDEC                           B    8 /* DECOMPRESSED LENGTH        ↩
0570     2 CB-TIME                           B    8 /* COMAND TIME                ↩
0580     2 CB-USER                           B   16 /* USER FIELD                 ↩
0590     2 CB-ROUTER                         B    1 /* ROUTER FLAGS               ↩
0600     2 CB-RESERVED-5                     B   23 /* RESERVED                   ↩
0610   * * ********************************* * **** ********************************
0620   * * ***** END OF LOCAL DATA AREA *** * **** ********************************
0630   * * ********************************* * **** ********************************↩
```

STPLRBE

```
***** DEFINE DATA LOCAL
0010 1  RBE-AREA(A154)      /* record buffer extraction area
0020 1  REDEFINE RBE-AREA
0030 2  RBE-MSG(A72)            /* error text for errors
0040 2  RBE-RESP(B4)            /* error number
0050 2  REDEFINE RBE-RESP
0060 3  RBE-SUBCODE(B2)         /* error subcode
0070 3  RBE-ERROR(B2)           /* actual error code
0080 2  RBE-VERNO(A4)           /* structure version
0090 2  RBE-FIELD-NAME(A32)     /* long name of field
0100 2  RBE-FORMAT(A1)      /* field format
0110 2  RBE-OPTS(A3)            /* special options
0120 2  RBE-LENGTH(B4)      /* field/RB length
0130 2  RBE-ADA-FIELD(A2)       /* Adabas short name
```

```
0140 2  RBE-RESRV2(A2)      /* reserved
0150 2  RBE-FIELD-OCC(B4)       /* field occurrence for MU or PE
0160 2  RBE-GROUP-OCC(B4)       /* PE occurrence for MU within PE
0170 2  RBE-OFFSET(B4)      /* offset into RB
0180 2  RBE-UNUSED(A18)         /* not used
***** END-DEFINE
```

STPUTPRM

```
***** DEFINE DATA PARAMETER
0010 1  CALL-TYPE(A1)       /* type of call
0020 **                     /* 'B' before invoking
0030 **                     /* 'A' after  invoking
0040 **                     /* 'E' error  incurred
0050 1  REQ-AREA(A200)      /* request area
0060 1  REDEFINE REQ-AREA
0070 2  RQ-VERNO(A4)        /* structure version
0080 2  RQ-TASK(A2)         /* subsystem number
0090 2  RQ-PROC(A8)         /* procedure name
0100 2  RQ-USER(A32)        /* user identification
0110 2  RQ-CMD(A2)          /* trigger command
0120 2  RQ-DBID(B2)         /* Trigger DBID
0130 2  RQ-FNR(B2)          /* trigger target file number
0140 2  RQ-FIELD(A2)        /* trigger field (short name)
0150 2  RQ-SYNC(A1)         /* sync/async request
0160 2  RQ-PARTIC(A1)       /* participating/non-participating request
0170 2  RQ-LENGTH(B2)       /* record buffer length
0180 2  RQ-UPD(A1)          /* RB update indicator
0190 2  RQ-TTYP(A1)         /* trigger type "P"re or Po"S"t
0200 2  RQ-RESP(B4)
0210 2  REDEFINE RQ-RESP
0220 3  RESP-CODE(B2)
0230 3  SUB-CODE(B2)
0240 2  RQ-PDA-TYPE(A1)     /* calling type
0250 2  RQ-RESRVED2(A55)
0260 2  RQ-CB(A80)          /* trigger control block
0270 1  ERR-INFO(A72)       /* error information for CALL-TYPE 'E'
0280 1  REDEFINE ERR-INFO
0290 2  ERR-NR(N4)          /* error number
0300 2  ERR-LINE(N4)        /* line number of error
0310 2  ERR-STAT(A1)        /* error status indicator
0320 2  ERR-PROG(A8)        /* error program
0330 2  ERR-LEVEL(N2)       /* not used
0340 2  ERR-TYPE(A24)       /* error identification
0350 1  REQ-GLOBAL-WS(A250) /* global WS area
0360 1  RESP(B4)            /* procedure response
***** END-DEFINE
```

STPUTRAK

```
0010 ************************************************************************
0020 *  Application: Adabas Stored Procedures
0030 *  Program    : STPUTRAK
0040 *  Function   : Routine that is invoked if triggers is running with
0050 *               'Trigger Logging' set to Active.
0060 *               Invoked (CALL-TYPE):
0070 *                  'I' - when the subsystem is initialized
0080 *                  'T' - when the subsystem is being terminated
0090 *                  'B' - before a procedure is invoked
0100 *                  'A' - after a procedure has completed
0110 *                  'E' - whenever an error occurs
0120 *  NOTE       : If logging is active, then the module (cataloged object)
0130 *               must exist; otherwise, a NAT0082 occurs.
0140 *  Author     : Adabas Development
0150 *  Date       : June 1994
0160 ************************************************************************
0170 DEFINE DATA PARAMETER USING STPUTPRM
0180            LOCAL
0190 01  EVENT      (A14)              /* event criteria
0200 01  REDEFINE EVENT
0210   02  E-FNR     (N5)
0220   02  E-F1      (A2)
0230   02  E-CMD     (A2)
0240   02  E-F2      (A2)
0250   02  E-FIELD   (A2)
0260 01  PARM-TYPE   (A7)
0270 01  TRIG-TYPE  (A10)
0280 01  REDEFINE TRIG-TYPE
0290   02  PRE-POST  (A4)
0300   02  FILL      (A1)
0310   02  PROC-TYPE (A5)
0320 01  UQE-ID     (A28)
0330 01  RB-TYPE     (A6)
0340 01  RBLEN       (N5)
0350 01  RESP-BIN    (B4)
0360 01  REDEFINE RESP-BIN
0370   02  RESP-SUBC (B2)
0380   02  RESP-CDE  (B2)
0390 END-DEFINE
0400 *
0410 FORMAT PS=0 LS=133                /* set report attributes
0420 *
0430 IF CALL-TYPE = 'I'                /* subsystem initialization
0440   WRITE NOTITLE (CD=TU)
0450         '***** Triggers and Stored Procedures *****' (GRI)
0460     / '      - Natural Subsystem Initialization -' (YEI)
```

```
0470    // 'Program + Library Location ...' (TU) *PROGRAM *LIBRARY-ID
0480     / 'Task Initialization Time .....' (TU) *DATX *TIMX
0490     / 'Task Identification Number ...' (TU) RQ-TASK
0500     / 'Task User Identification .....' (TU) *INIT-USER *INIT-ID
0510     / '********************** INIT **********************' (GRI)
0520  NEWPAGE                            /* setup for proper headings
0530  ESCAPE ROUTINE                     /* return control
0540 END-IF
0550 *
0560 IF CALL-TYPE = 'T'                  /* subsystem termination
0570  WRITE NOTITLE (CD=TU)
0580       '***** Triggers and Stored Procedures *****' (GRI)
0590     / '        - Natural Subsystem Termination -' (YEI)
0600    // 'Task Termination Time ........' (TU) *DATX *TIMX
0610     / 'Task Identification Number ...' (TU) RQ-TASK
0620     / 'Task User Identification .....' (TU) *INIT-USER *INIT-ID
0630     / '********************** EXIT **********************' (GRI)
0640  ESCAPE ROUTINE                     /* return control
0650 END-IF
0660 *
0670 IF CALL-TYPE = 'A'                  /*  after invoking procedure
0680   MOVE RESP TO RESP-BIN
0690   WRITE RQ-TASK 2X *DATX *TIMX 'complete' 16X RQ-PROC 6X 'Resp:'
0700         RESP-CDE RESP-SUBC
0710 END-IF
0720 *
0730 IF CALL-TYPE = 'B'                  /*  before invoking procedure
0740  MOVE RQ-RESRVED2 TO UQE-ID
0750* MOVE RB-RBL   TO RBLEN
0760  MOVE RQ-LENGTH TO RBLEN
0770  IF RQ-TTYP = 'P'
0780    MOVE 'Pre ' TO PRE-POST
0790  ELSE
0800    MOVE 'Post' TO PRE-POST
0810  END-IF
0820  IF RQ-SYNC = 'A'
0830    MOVE 'ASync' TO PROC-TYPE
0840  END-IF
0850  IF RQ-SYNC = 'S'
0860    MOVE 'Sync' TO PROC-TYPE
0870    IF RQ-PARTIC = 'P'
0880      MOVE 'Part' TO PROC-TYPE
0890    END-IF
0900    IF RQ-PARTIC = 'N'
0910      MOVE 'Non-P' TO PROC-TYPE
0920    END-IF
0930  END-IF
0940  DECIDE ON FIRST VALUE OF RQ-PDA-TYPE
0950    VALUE 'C' MOVE 'Control' TO PARM-TYPE
0960    VALUE 'N' MOVE 'No Parm' TO PARM-TYPE
0970    VALUE 'R' MOVE 'Resp'    TO PARM-TYPE
0980    NONE      MOVE 'Unknown' TO PARM-TYPE
```

```
0990  END-DECIDE
1000  DECIDE ON FIRST VALUE OF RQ-UPD
1010    VALUE 'A' MOVE 'Access' TO RB-TYPE
1020    VALUE 'N' MOVE 'No Rec' TO RB-TYPE
1030    VALUE 'U' MOVE 'Update' TO RB-TYPE
1040    NONE      MOVE '??????' TO RB-TYPE
1050  END-DECIDE
1060  IF RQ-CMD = 'PC'
1070    MOVE '*Stored Proc.*' TO EVENT
1080  ELSE
1090    MOVE RQ-FNR TO E-FNR
1100    MOVE RQ-CMD TO E-CMD
1110    MOVE RQ-FIELD TO E-FIELD
1120  END-IF
1130  DISPLAY NOTITLE (CD=TU)
1140      'Tsk' RQ-TASK 'Date' *DATX 'Time' *TIMX 'User' RQ-USER(AL=8)
1150      'Fnr    Cmd Fld' (TU) EVENT 'Proc' (TU) RQ-PROC
1160      'Type' (TU) TRIG-TYPE 'Parms' (TU) PARM-TYPE
1170      'RecBuf' RB-TYPE
1180 * PRINT 5X 'UserID ...' UQE-ID(EM=H(28)) /* UQE-ID
1190 *
1200 * A special overwrite option allows the user to have the procedure
1210 * called with the additional parameter of the RQ-GLOBAL-WS.
1220 * This is valid only if RQ-PDA-TYPE is set to 'C'.
1230 * The procedure should expect parameters to be passed as specified in
1240 * STPAPRM1 i.e. CALLNAT 'procname' REQ-AREA REQ-GLOBAL-WS RESP
1250 *
1260 * MOVE 'W' TO CALL-TYPE
1270 *
1280 * RQ-GLOBAL-WS is an area that is not changed between each call to
1290 * the procedures. It may be used for keeping statistics or whatever.
1300 *
1310 END-IF
1320 *
1330 IF CALL-TYPE = 'E'                    /* subsystem error notification
1340   WRITE NOTITLE (CD=TU)
1350        '***** Triggers and Stored Procedures *****' (GRI)
1360     / '        - Natural Subsystem Error Info -' (YEI)
1370    // 'Task Termination Time .......' (TU) *DATX *TIMX
1380     / 'Task Identification Number ..' (TU) RQ-TASK
1390       '<<< Interrupted with an ERROR <<<<<'
1400    // '* Subsystem Error Number ....' ERR-NR
1410   41T '* Stored Proc ....' RQ-PROC
1420     / '*           Active Module ...' ERR-PROG
1430   41T '* UserID .........' RQ-USER
1440     / '*           Line Number .....' ERR-LINE (EM=9999)
1450   41T '* Trigger Cmd ....' RQ-CMD
1460     / '*           Error Level .....' ERR-LEVEL
1470   41T '* Trigger Fnr ....' RQ-FNR (AD=L)
1480     / '*           Error Status ....' ERR-STAT
1490   41T '* Trigger Type ...' RQ-TTYP RQ-PARTIC 'opt' RQ-PDA-TYPE RQ-UPD
1500     / '*           Error Type ......' ERR-TYPE (AL=14)
```

```
1510*  41T '* Field Name .....' RQ-TRG-FIELD '+' RB-RBL
1520*    / '*           UQE Ident. ......' CA-USER
1530     /  '*** Processing for this request ABNORMALLY terminated ***'
1540    // '********************* ERROR *********************' (GRI)
1550  ESCAPE ROUTINE                     /* return control
1560 END-IF
1570 *
1580 END
```

STPAPARM

```
***** DEFINE DATA PARAMETER
0010 1  REQ-AREA(A200)      /* request area
0020 1  REDEFINE REQ-AREA
0030 2  RQ-VERNO(A4)        /* structure version
0040 2  RQ-TASK(A2)         /* subsystem number
0050 2  RQ-PROC(A8)         /* procedure name
0060 2  RQ-USER(A32)        /* user identification
0070 2  RQ-CMD(A2)          /* trigger command
0080 2  RQ-DBID(B2)         /* Trigger DBID
0090 2  RQ-FNR(B2)          /* trigger target file number
0100 2  RQ-FIELD(A2)        /* trigger field (short name)
0110 2  RQ-SYNC(A1)         /* sync/async request
0120 2  RQ-PARTIC(A1)       /* participating/non-participating request
0130 2  RQ-LENGTH(B2)       /* record buffer length
0140 2  RQ-UPD(A1)          /* RB update indicator
0150 2  RQ-TTYPE(A1)        /* pre- or post-trigger
0160 2  RQ-RESP(B4)         /* subcode(B2) + resp code(B2)
0170 2  RQ-PDA-TYPE(A1)     /* calling parameters type
0180 2  RQ-USERID(A28)      /* user ID from Adabas CQE
0190 2  RQ-RESRVED2(A27)
0200 2  RQ-CB(A80)          /* trigger control block
0210 1  RESP(B4)            /* procedure response
***** END-DEFINE
```

STPAPRM1

```
***** DEFINE DATA PARAMETER STPAPRM1 LIBRARY SYSSPT
0010 1  REQ-AREA(A200)           /* Request area
0020 1  REDEFINE REQ-AREA
0030 2  RQ-VERNO(A4)             /* structure version
0040 2  RQ-TASK(A2)              /* subsystem number
0050 2  RQ-PROC(A8)              /* procedure name
0060 2  RQ-USER(A32)             /* user identification
0070 2  RQ-CMD(A2)               /* trigger command
0080 2  RQ-DBID(B2)              /* Trigger DBID
0090 2  RQ-FNR(B2)               /* trigger target file number
0100 2  RQ-FIELD(A2)             /* trigger field (short name)
0110 2  RQ-SYNC(A1)              /* sync/async request
0120 2  RQ-PARTIC(A1)            /* participating/non-participating request
0130 2  RQ-LENGTH(B2)            /* record buffer length
0140 2  RQ-UPD(A1)               /* RB update indicator
0150 2  RQ-RESRVED1(A1)          /* not used
0160 2  RQ-RESP(B4)
0170 2  RQ-PDA-TYPE(A1)
0180 2  RQ-RESRVED2(A55)
0190 2  RQ-CB(A80)               /* trigger control block
0200 1  REQ-GLOBAL-WS(A250)      /* global WS area
0210 1  RESP(B4)                 /* procedure response
***** END-DEFINE
```

STPXPARM

```
0010     1 REQ-AREA                          A  200 /* Request Area                ↩
0020   R 1 REQ-AREA                                 /* Redef. begin : REQ-AREA     ↩
0030     2 RQ-VERNO                          A    4 /* Structure Version          ↩
0040     2 RQ-TASK                           A    2 /* Subsystem Number           ↩
0050     2 RQ-PROC                           A    8 /* Procedure Name             ↩
0060     2 RQ-USER                           A   32 /* User Identification        ↩
0070     2 RQ-CMD                            A    2 /* Trigger Cmd                ↩
0080     2 RQ-DBID                           B    2 /* Trigger DBID               ↩
0090     2 RQ-FNR                            B    2 /* Trigger Target File Nr     ↩
```

```
0100     2 RQ-FIELD                          A    2 /* Trigger Field (Short Name)  ↩
0110     2 RQ-SYNC                           A    1 /* Sync/Async Request          ↩
0120     2 RQ-PARTIC                         A    1 /* Part/Non-participating Req  ↩
0130     2 RQ-LENGTH                         B    2 /* Record Buffer Length       ↩
0140     2 RQ-UPD                            A    1 /* RB Update Indicator        ↩
0150     2 RQ-TTYPE                          A    1 /* Pre or Post Trigger        ↩
0160     2 RQ-RESP                           B    4 /* Subcode(B2) + Resp Code(B2) ↩
0170     2 RQ-PDA-TYPE                       A    1 /* Calling Parameters Type    ↩
0180     2 RQ-USERID                         A   28 /* UserID from ADABAS UQE     ↩
0190     2 RQ-RESRVED2                       A   27 /*                             ↩
0200     2 RQ-CB                             A   80 /* Trigger Control Block      ↩
0210     1 RQ-CBX                            A  192 /* X Verion of CB             ↩
0220     1 RESP                               B    4 /* Procedure Response
```

SAMP0001

```
0010 ************************************************************************
0020 *  Application: ADASTP
0030 *  Subprogram : SAMP0001
0040 *  Author     : Adabas Development
0050 *  Date       : August 1995
0060 *  Function   : Sample routine of processing by a procedure
0070 *  Remarks    : This routine converts the CONTACT-NAME into uppercase
0080 *               and extracts all the keywords associated with it.
0090 *               Once processing is completed, control is returned to
0100 *               the caller.
0110 *               Parameter RESP must be set to zero if processing is
0120 *               successful.
0130 *
0140 *  Parameters : REQ-AREA (A200)
0150 *               RESP       (B4)
0160 *
0170 *  Trigger Typ: The type of trigger will be PARTICIPATING; i.e.,
0180 *               synchronous.
0190 *  Rec Buffer : The record buffer will be available for update via a
```

```
0200 *               call to the external routine STPRBE.
0210 * Trigger Defn: Definition on the trigger file (note that there is a
0220 *               trigger for the insert and update) is as follows:
0230 *                 File Number ....... 11
0240 *                 File Name ......... CONTACTS
0250 *                 Command Type ...... Update + Insert
0260 *                 Long Field Name ... CONTACT-NAME
0270 *                 Adabas Field ...... LE
0280 *                 Field Prty/Seq .... 10_
0290 *               Procedure Information
0300 *                 Name (Subpgm)...... SAMP0001
0310 *                 Pre Cmd Select .... Y  (Pre)
0320 *                 Trigger Type ...... P  (Participating)
0330 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0340 *                 RecBuffer Access .. U  (May be updated)
0350 *
0360 ************************************************************************
0370 DEFINE DATA PARAMETER USING STPAPARM
0380            LOCAL     USING STPLRBE /* parms for the call routine
0390            LOCAL
0400   01  REC-BUFFER(A20/1:26)         /* max, record buffer passed to STPRBE
0410   01  REDEFINE REC-BUFFER          /* redefine this to get the definition
0420     02  INPUT-NAME   (A60)
0430     02  OUTPUT-NAME  (A60)
0440     02  KEYWORDS(A20/1:20)
0450   01  FUNC            (A4)
0460   01  SUB             (I2)
0470   01  SUB1            (I2)
0480   01  SUB2            (I2)
0490   01  W-UPPER        (A61)
0500   01  REDEFINE W-UPPER
0510     02  #UPPER       (A60)
0520     02  REDEFINE #UPPER
0530       03  CHAR   (A1/1:60)
0540   01  #KEYS     (A40/1:20)
0550 END-DEFINE
0560 *
0570 *  First, all procedures for this file must go through the audit procedure
0580 *  because our example requires a trace of all commands to this file.
0590 *
0600 CALLNAT 'SAMP0003' REQ-AREA RESP
0610 *
0620 * Since the record buffer information is available to us, we can now call
0630 * the record buffer extraction routine (STPRBE) to obtain the contents of
0640 * the buffer.
0650 *
0660 * Function 'GR' -- GET RB value using RB offset + length
0670 *     This enables the caller to obtain information based on a
0680 *     certain location; hence, RBE-OFFSET specifies the start
0690 *     position and RBE-LENGTH specifies the length.
0700 *
0710 MOVE 1   TO RBE-OFFSET            /* start at the beginning
```

```
0720 MOVE 520 TO RBE-LENGTH             /* for a max. length of 520 bytes
0730 MOVE 'GR' TO FUNC
0740 CALL 'STPRBE' 'GR' RBE-AREA REC-BUFFER(1)
0750 IF RBE-RESP NE 0
0760   PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
0770        RBE-ERROR 'subcode' RBE-SUBCODE 'for func GR'
0780   MOVE RBE-RESP TO RESP
0790   ESCAPE ROUTINE
0800 END-IF
0810 * PERFORM PRINT-REC-BUFFER         /* option to print the parameters
0820 *
0830 * Change all lowercase to UPPERcase
0840 *
0850 MOVE INPUT-NAME TO #UPPER
0860 *
0870 EXAMINE #UPPER AND TRANSLATE INTO UPPER CASE
0880 *
0890 MOVE #UPPER TO OUTPUT-NAME         /* save the uppercase name
0900 *
0910 FOR SUB 1 60                       /* loop to remove all special chars.
0920   IF CHAR(SUB) = MASK(S)
0930     MOVE ' ' TO CHAR(SUB)
0940     ESCAPE TOP
0950   END-IF
0960 END-FOR
0970 *
0980 * We are now ready to extract keywords from our name. This sample is
0990 * very basic and may be made as complex as required.
1000 * This routine assumes a max. length of 20 and a max. num. of 20 keywords
1010 *
1020 EXAMINE FULL W-UPPER FOR FULL ' A '   REPLACE ' '
1030 EXAMINE FULL W-UPPER FOR FULL ' AND ' REPLACE ' '
1040 EXAMINE FULL W-UPPER FOR FULL ' AS '  REPLACE ' '
1050 EXAMINE FULL W-UPPER FOR FULL ' AT '  REPLACE ' '
1060 EXAMINE FULL W-UPPER FOR FULL ' ARE ' REPLACE ' '
1070 EXAMINE FULL W-UPPER FOR FULL ' BE '  REPLACE ' '
1080 EXAMINE FULL W-UPPER FOR FULL ' DO '  REPLACE ' '
1090 EXAMINE FULL W-UPPER FOR FULL ' FOR ' REPLACE ' '
1100 EXAMINE FULL W-UPPER FOR FULL ' HERE ' REPLACE ' '
1110 EXAMINE FULL W-UPPER FOR FULL ' IF '  REPLACE ' '
1120 EXAMINE FULL W-UPPER FOR FULL ' IN '  REPLACE ' '
1130 EXAMINE FULL W-UPPER FOR FULL ' IS '  REPLACE ' '
1140 EXAMINE FULL W-UPPER FOR FULL ' IT '  REPLACE ' '
1150 EXAMINE FULL W-UPPER FOR FULL ' OF '  REPLACE ' '
1160 EXAMINE FULL W-UPPER FOR FULL ' ON '  REPLACE ' '
1170 EXAMINE FULL W-UPPER FOR FULL ' OR '  REPLACE ' '
1180 EXAMINE FULL W-UPPER FOR FULL ' TO '  REPLACE ' '
1190 EXAMINE FULL W-UPPER FOR FULL ' THE ' REPLACE ' '
1200 EXAMINE FULL W-UPPER FOR FULL ' TOO ' REPLACE ' '
1210 EXAMINE FULL W-UPPER FOR FULL ' WAS ' REPLACE ' '
1220 EXAMINE FULL W-UPPER FOR FULL ' WITH ' REPLACE ' '
1230 EXAMINE #UPPER FOR FULL ' ' REPLACE ',' /* put delimiters in the string
```

```
1240 *
1250 RESET KEYWORDS(*)
1260 STACK TOP DATA #UPPER                    /* now we will separate each word
1270 INPUT (AD=I IP=ON) #KEYS(01:03) / #KEYS(04:06) / #KEYS(07:09)
1280                  / #KEYS(10:12) /* #KEYS(13:15) / #KEYS(16:18)
1290                  / #KEYS(19:20)
1300 *
1310 MOVE 1 TO SUB2
1320 MOVE #KEYS(1) TO KEYWORDS(1)
1330 FOR SUB 2 20                             /* now we remove all duplicates
1340   FOR SUB1 1 SUB
1350     IF #KEYS(SUB) = KEYWORDS(SUB1)
1360       RESET #KEYS(SUB)
1370     END-IF
1380   END-FOR
1390   IF #KEYS(SUB) NE ' '
1400     ADD 1 TO SUB2
1410     MOVE #KEYS(SUB) TO KEYWORDS(SUB2)  /* and finally save the value
1420   END-IF
1430 END-FOR
1440 *
1450 * Function 'UR' -- Update RB value using RB offset + length
1460 *     This enables the caller to change information based on a
1470 *     certain location; hence, RBE-OFFSET specifies the start
1480 *     position and RBE-LENGTH specified the length.
1490 *
1500 * PERFORM PRINT-REC-BUFFER             /* print the final results
1510 MOVE 1   TO RBE-OFFSET                 /* start at the beginning
1520 MOVE 520 TO RBE-LENGTH                 /* for a max. length of 520 bytes
1530 MOVE 'UR' TO FUNC                      /* request to update all changes
1540 CALL 'STPRBE' 'UR' RBE-AREA REC-BUFFER(1)
1550 IF RBE-RESP NE 0
1560   PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
1570        RBE-ERROR 'subcode' RBE-SUBCODE 'for func UR'
1580   MOVE RBE-RESP TO RESP
1590   ESCAPE ROUTINE
1600 END-IF
1610 *
1620 * Return to the caller: everything went okay
1630 *
1640 ESCAPE ROUTINE
1650 *
1660 DEFINE SUBROUTINE PRINT-REC-BUFFER
1670 *-----------------------------------------------------------------------*
1680 *
1690 *  For testing purposes, display the information returned from STPRBE
1700 *  This routine assumes a maximum of three subsystems running.
1710 *
1720 *-----------------------------------------------------------------------*
1730   DECIDE ON FIRST VALUE OF RQ-TASK
1740     VALUE '01'
1750       WRITE (1) NOTITLE NOHDR (AD=L CD=TU)
```

```
1760           '>>>> RECORD BUFFER EXTRACTION: Function' FUNC '<<<<'
1770         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1780         / '             ....' (TU) RBE-ADA-FIELD RBE-FIELD-OCC
1790         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1800         / '  Message .......' (TU) RBE-MSG(AL=60)
1810         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1820         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
1830     VALUE '02'
1840       WRITE (2) NOTITLE NOHDR (AD=L CD=TU)
1850           '>>>> RECORD BUFFER EXTRACTION: Function' FUNC '<<<<'
1860         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1870         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1880         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1890         / '  Message .......' (TU) RBE-MSG(AL=60)
1900         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
1910         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
1920     VALUE '03'
1930       WRITE (3) NOTITLE NOHDR (AD=L CD=TU)
1940           '>>>> RECORD BUFFER EXTRACTION: Function' FUNC '<<<<'
1950         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
1960         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
1970         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR '<<<<<'
1980         / '  Message .......' (TU) RBE-MSG(AL=60)
1990         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
2000         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
2010     NONE
2020       WRITE NOTITLE NOHDR (AD=L CD=TU)
2030           '>>>> RECORD BUFFER EXTRACTION: function' FUNC '<<<<'
2040         / '  Field Info ....' (TU) RBE-FIELD-NAME RBE-FORMAT RBE-LENGTH
2050         / '             ....' (TU) RBE-ADA-FIELD  RBE-FIELD-OCC
2060         / '  Resp + Error ..' (TU) RBE-RESP RBE-ERROR'<<<<<'
2070         / '  Message .......' (TU) RBE-MSG(AL=60)
2080         / '  Rec Buffer ....' (TU) / REC-BUFFER(1)(AL=79)
2090         / '* * * * * * * * * * * * * * * * * * * * * * * * * * * * * '
2100   END-DECIDE
2110 *
2120 END-SUBROUTINE
2130 *
2140 END
```

SAMP0002

```
0010 ************************************************************************
0020 *  Application: Adabas Triggers
0030 *   Subprogram: SAMP0002
0040 *     Function: Sample routine of processing by a stored procedure
0050 *               The requirement is to audit all commands for a file
0060 *               by writing out an audit record to a file/printer.
0070 *               For this example, the audit is the Natural system file.
0080 * Trigger Defn: Trigger Information
0090 *                 File Number ....... 11
0100 *                 File Name ......... CONTACTS
0110 *                 Command Type ...... ** All Command **
0120 *                 Long Field Name ... ** Any Field **
0130 *                 Adabas Field ...... **
0140 *                 Field Prty/Seq .... ___
0150 *               Procedure Information
0160 *                 Name (Subpgm)...... SAMP0002
0170 *                 Pre Cmd Select .... Y  (Pre)
0180 *                 Trigger Type ...... A  (Asynchronous)
0190 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0200 *                 RecBuffer Access .. N  (No RecBuff Access)
0210 *
0220 *       AUTHOR: Adabas Development
0230 *         DATE: December 1995
0240 ************************************************************************
0250 DEFINE DATA PARAMETER USING STPAPARM
0260            LOCAL     USING STPLCB /* DSECT of the Adabas control block
0270            LOCAL
0280  01  #SRCID     (A18)             /* key of the audit record
0290  01  REDEFINE #SRCID
0300    02  SRC-LIB   (A8)             /* to be placed on a Natural system file
0310    02  SRC-PGM   (A8)
0320    02  SRC-SEQ   (B2)
0330  01  #DATE       (A8)
0340  01  #TIME       (A8)
0350  01  LOG-AREA VIEW OF SYSTEM2     /* write information to the FNAT file
0360    02  SRCID
0370    02  SRCTX      (1)
0380  01  W-USERID   (A28)             /* user ID from originating command
0390  01  REDEFINE W-USERID
0400    02  W-F1     (A20)
0410    02  W-USER    (A8)             /* TP USID of the user ID
0420  01  #TEXT (A72)                  /* text message to be written
0430  01  REDEFINE #TEXT
0440    02  TX-LNO    (B2)
0450    02  TX-F1     (A1)
0460    02  TX-DATE   (A8)
```

```
0470    02  TX-F2     (A1)
0480    02  TX-TIME   (A5)
0490    02  TX-F3     (A1)
0500    02  TX-USER   (A8)
0510    02  TX-F4     (A1)
0520    02  TX-CMD    (A2)
0530    02  TX-F5     (A1)
0540    02  TX-PRE    (A3)
0550    02  TX-F6     (A1)
0560    02  TX-FNR    (N3)
0570    02  TX-F7     (A1)
0580    02  TX-RBL    (N4)
0590    02  TX-F8     (A1)
0600    02  TX-SYNC   (A5)
0610    02  TX-F9     (A1)
0620    02  TX-TASK   (A2)
0630    02  TX-F10    (A1)
0640    02  TX-FIELD  (A2)
0650    02  TX-F11    (A1)
0660    02  TX-PROC   (A8)
0670    02  TX-F12    (A1)
0680    02  TX-USR2   (A8)
0690 END-DEFINE
0700 *
0710 ASSIGN #SRCID =  'AUDIT   LOGINFO' /* set the target lib and pgm name
0720 MOVE H'0000' TO SRC-SEQ
0730 *
0740 MOVE RQ-CB   TO CB                 /* move ACB into our CB layout
0750 MOVE H'0010' TO TX-LNO             /* line number of Natural source
0760 MOVE *DATE   TO TX-DATE
0770 MOVE *TIMX   TO TX-TIME
0780 MOVE RQ-USERID TO W-USERID         /* user ID of the command
0790 MOVE W-USER  TO TX-USER            /* may be a batch user
0800 IF NOT TX-USER = MASK(PPPPPPPP)    /* printable user ID?
0810  MOVE RQ-USER TO TX-USER           /* no, so use the jobname or TPname
0820 END-IF
0830 MOVE RQ-CMD  TO TX-CMD             /* information from the CB layout
0840 MOVE RQ-FNR  TO TX-FNR
0850 MOVE RQ-TASK TO TX-TASK            /* subsystem number
0860 IF RQ-LENGTH > 9999                /* exceed max. size in audit message?
0870  MOVE 9999 TO TX-RBL
0880 ELSE
0890  MOVE RQ-LENGTH TO TX-RBL          /* the real record buffer length
0900 END-IF
0910 MOVE *PROGRAM  TO TX-PROC          /* originating procedure. This subpgm
0920 IF RQ-TTYPE = 'P'                  /* trigger type
0930  MOVE 'Pre' TO TX-PRE
0940 ELSE
0950  MOVE 'Pos' TO TX-PRE
0960 END-IF
0970 IF RQ-SYNC = 'A'                   /* processing type
0980  MOVE 'ASync' TO TX-SYNC
```

```
0990 ELSE
1000  IF RQ-PARTIC = 'P'                /* trigger logic type
1010    MOVE 'Part'  TO TX-SYNC
1020  ELSE
1030    MOVE 'Non-P' TO TX-SYNC
1040  END-IF
1050 END-IF
1060 *
1070 * Now we do some logic to write the information out to a report
1080 * Here we support up to five subsystems
1090 * Contents are a one-line display to minimize output
1100 *
1110 DECIDE ON FIRST VALUE OF RQ-TASK
1120    VALUE '01'
1130      DISPLAY (1) NOTITLE (AD=L CD=TU)
1140            'Procedure' *PROGRAM
1150            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1160            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1170            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1180            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1190    VALUE '02'
1200      DISPLAY (2) NOTITLE (AD=L CD=TU)
1210            'Procedure' *PROGRAM
1220            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1230            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1240            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1250            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1260    VALUE '03'
1270      DISPLAY (3) NOTITLE (AD=L CD=TU)
1280            'Procedure' *PROGRAM
1290            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1300            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1310            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1320            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1330    VALUE '04'
1340      DISPLAY (4) NOTITLE (AD=L CD=TU)
1350            'Procedure' *PROGRAM
1360            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1370            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1380            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1390            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1400    NONE
1410      DISPLAY (5) NOTITLE (AD=L CD=TU)
1420            'Procedure' *PROGRAM
1430            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1440            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1450            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1460            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1470  END-DECIDE
1480 *
1490 * Finally, we write this information to a 'audit' File. In this case, we
1500 * use the Natural FNAT file for simplicity. Realistically, a separate
```

```
1510 * 'audit' file should be used.
1520 *
1530  MOVE #TEXT TO LOG-AREA.SRCTX (1.1)
1540  MOVE H'0001' TO SRC-SEQ
1550  ASSIGN LOG-AREA.SRCID = #SRCID
1560  STORE LOG-AREA
1570  END TRANSACTION                    /* required for non-participating
1580 *                                   /* and asynchronous triggers
1590 *
1600 END
```

SAMP0003

```
0010 ************************************************************************
0020 *  Application: Adabas Triggers
0030 *   Subprogram: SAMP0003
0040 *     Function: Sample routine of processing by a stored procedure
0050 *               The requirement is to audit  all commands for a file
0060 *               by writing out an audit record to a file/printer.
0070 *               For this example, the audit is the Natural system file.
0080 *
0090 *               This routine is called by participating triggers and
0100 *               will contain no ET logic; hence, it must have been
0110 *               invoked as a result of a update/delete/store command
0120 * Trigger Defn: None because it will be invoked directly from another
0130 *               procedure. In this case SAMP0001.
0140 *
0150 *       Author: Adabas Development
0160 *         Date: December 1995
0170 ************************************************************************
0180 DEFINE DATA PARAMETER USING STPAPARM
0190            LOCAL     USING STPLCB /* DSECT of the Adabas control block
0200            LOCAL
0210  01  #SRCID     (A18)             /* key of the audit record
0220  01  REDEFINE #SRCID
0230    02  SRC-LIB   (A8)             /* to be placed on a Natural system file
0240    02  SRC-PGM   (A8)
0250    02  SRC-SEQ   (B2)
0260  01  #DATE      (A8)
0270  01  #TIME      (A8)
0280  01  LOG-AREA VIEW OF SYSTEM2     /* write information to the FNAT file
0290    02  SRCID
0300    02  SRCTX      (1)
0310  01  W-USERID   (A28)             /* user ID from originating command
0320  01  REDEFINE W-USERID
0330    02  W-F1     (A20)
0340    02  W-USER    (A8)             /* TP USID of the user ID
0350  01  #TEXT (A72)                  /* text message to be written
```

```
0360  01  REDEFINE #TEXT
0370    02  TX-LNO    (B2)
0380    02  TX-F1     (A1)
0390    02  TX-DATE   (A8)
0400    02  TX-F2     (A1)
0410    02  TX-TIME   (A5)
0420    02  TX-F3     (A1)
0430    02  TX-USER   (A8)
0440    02  TX-F4     (A1)
0450    02  TX-CMD    (A2)
0460    02  TX-F5     (A1)
0470    02  TX-PRE    (A3)
0480    02  TX-F6     (A1)
0490    02  TX-FNR    (N3)
0500    02  TX-F7     (A1)
0510    02  TX-RBL    (N4)
0520    02  TX-F8     (A1)
0530    02  TX-SYNC   (A5)
0540    02  TX-F9     (A1)
0550    02  TX-TASK   (A2)
0560    02  TX-F10    (A1)
0570    02  TX-FIELD  (A2)
0580    02  TX-F11    (A1)
0590    02  TX-PROC   (A8)
0600    02  TX-F12    (A1)
0610    02  TX-USR2   (A8)
0620 END-DEFINE
0630 *
0640 ASSIGN #SRCID =  'AUDIT   LOGINFO' /* set the target lib and program name
0650 MOVE H'0000' TO SRC-SEQ
0660 *
0670 MOVE RQ-CB   TO CB                /* move ACB into the CB layout
0680 MOVE H'0010' TO TX-LNO            /* line number of Natural source
0690 MOVE *DATE   TO TX-DATE
0700 MOVE *TIMX   TO TX-TIME
0710 MOVE RQ-USERID TO W-USERID        /* user ID of the command
0720 MOVE W-USER  TO TX-USER           /* may be a batch user
0730 MOVE RQ-USER TO TX-USR2           /* jobname or TPname
0740 MOVE RQ-CMD  TO TX-CMD            /* information from the CB layout
0750 MOVE RQ-FNR  TO TX-FNR
0760 MOVE RQ-TASK TO TX-TASK           /* subsystem number
0770 IF RQ-LENGTH > 9999               /* exceed max. size in audit message?
0780   MOVE 9999 TO TX-RBL
0790 ELSE
0800   MOVE RQ-LENGTH TO TX-RBL        /* the real record buffer length
0810 END-IF
0820 MOVE *PROGRAM  TO TX-PROC         /* originating procedure. This subpgm
0830 IF RQ-TTYPE = 'P'                 /* trigger type
0840   MOVE 'Pre' TO TX-PRE
0850 ELSE
0860   MOVE 'Pos' TO TX-PRE
0870 END-IF
```

```
0880 IF RQ-SYNC = 'A'                    /* processing type
0890   MOVE 'ASync' TO TX-SYNC
0900 ELSE
0910   IF RQ-PARTIC = 'P'                /* trigger logic type
0920     MOVE 'Part'  TO TX-SYNC
0930   ELSE
0940     MOVE 'Non-P' TO TX-SYNC
0950   END-IF
0960 END-IF
0970 *
0980 * Now we do some logic to write the information out to a report
0990 * Here we support up to five subsystems
1000 * Contents are a one-line display to minimize output
1010 *
1020 DECIDE ON FIRST VALUE OF RQ-TASK
1030    VALUE 'O1'
1040      DISPLAY (1) NOTITLE (AD=L CD=TU)
1050            'Procedure' *PROGRAM
1060            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1070            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1080            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1090            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1100            'Usr2' (TU) TX-USR2
1110    VALUE 'O2'
1120      DISPLAY (2) NOTITLE (AD=L CD=TU)
1130            'Procedure' *PROGRAM
1140            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1150            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1160            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1170            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1180            'Usr2' (TU) TX-USR2
1190    VALUE 'O3'
1200      DISPLAY (3) NOTITLE (AD=L CD=TU)
1210            'Procedure' *PROGRAM
1220            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1230            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1240            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1250            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1260            'Usr2' (TU) TX-USR2
1270    VALUE 'O4'
1280      DISPLAY (4) NOTITLE (AD=L CD=TU)
1290            'Procedure' *PROGRAM
1300            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1310            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
1320            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1330            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1340            'Usr2' (TU) TX-USR2
1350    NONE
1360      DISPLAY (5) NOTITLE (AD=L CD=TU)
1370            'Procedure' *PROGRAM
1380            'Date' (TU) TX-DATE 'Time'  (TU) TX-TIME
1390            'Task' (TU) RQ-TASK 'UserID'(TU) TX-USER
```

```
1400            'Cmd'  (TU) TX-CMD  'Fld'   (TU) RQ-FIELD
1410            'PreP' (TU) TX-PRE  'Fnr'   (TU) TX-FNR
1420            'Usr2' (TU) TX-USR2
1430  END-DECIDE
1440 *
1450 * Finally we write this info to a 'audit' file. In this case, we use the
1460 * Natural FNAT file for simplicity. Realistically, a separate 'audit' file
1470 * should be used. End Transaction (ET) must  not be issued because this
1480 * will be controlled by the application and not the trigger procedure.
1490 *
1500  MOVE #TEXT TO LOG-AREA.SRCTX (1.1)
1510  MOVE H'0001' TO SRC-SEQ
1520  ASSIGN LOG-AREA.SRCID = #SRCID
1530  STORE LOG-AREA
1540 *
1550 END
```

SAMP0004

```
0010 ************************************************************************
0020 *  Application: Adabas Triggers
0030 *   Subprogram: SAMP0004
0040 *     Function: Sample routine of processing by a stored procedure
0050 *               referential integrity - RESTRICT
0060 *               (assume that the primary key is on the EMPLOYEES file and
0070 *                the foreign key on the VEHICLES + MISCELLANEOUS files).
0080 * Trigger Defn: Definition on the trigger file is as follows:
0090 *                 File Number ....... 3
0100 *                 File Name ......... VEHICLES-FILE
0110 *                 Command Type ...... Delete
0120 *                 Long Field Name ... ** Any Field **
0130 *                 Adabas Field ...... **
0140 *                 Field Prty/Seq .... ___
0150 *               Procedure Information
0160 *                 Name (Subpgm)...... SAMP0004
0170 *                 Pre Cmd Select .... Y  (Pre)
0180 *                 Trigger Type ...... N  (Non-Participating)
0190 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0200 *                 RecBuffer Access .. N  (No RecBuff Access)
0210 *
0220 *      Invoked: Invoked with deletes from VEHICLES/MISCELLANEOUS files
0230 *               Sample Routine: SAMPREF1
0240 *       Author: Adabas Development
0250 *         Date: December 1995
0260 ************************************************************************
0270 DEFINE DATA PARAMETER USING STPAPARM
0280           LOCAL
0290   01  VEHICLES  VIEW OF VEHICLES
```

```
0300     02  PERSONNEL-ID               /* foreign key: field AC
0310   01  MISCELLANEOUS VIEW OF MISCELLANEOUS
0320     02  PERSONNEL-ID               /* foreign key: field CA
0330   01  EMPLOYEES VIEW OF EMPLOYEES
0340     02  PERSONNEL-ID               /* primary key: field AA
0350   01  #FILE        (P5)
0360   01  #ISN        (P10)
0370   01  #PERS-NUM    (A8)
0380   01  CONTRL-BLK  (A80)
0390   01  REDEFINE CONTRL-BLK
0400     02  CB-FIL1   (A12)
0410     02  CB-ISN     (B4)
0420 END-DEFINE
0430 *
0440 *  First we extract the foreign key information
0450 *  i.e., get the ISN of the record in the ACB and read this record
0460 *    to extract the required information; i.e., the foreign key info.
0470 *    NOTE: With a delete, no data is passed in the record buffer.
0480 *
0490 MOVE RQ-CB  TO CONTRL-BLK         /* get the ACB of the originating cmd
0500 MOVE RQ-FNR TO #FILE              /* find out which file has the delete
0510 MOVE CB-ISN TO #ISN               /* ISN of the record to be deleted
0520 *
0530 IF #FILE = 3                      /* identify the file: Vehicles
0540   DO
0550     GET VEHICLES #ISN             /* get the value of the foreign key
0560     MOVE PERSONNEL-ID(0550) TO #PERS-NUM    /* get the key
0570   DOEND
0580 ELSE
0590   IF #FILE = 2                    /* or the Miscellaneous file
0600     DO
0610       GET MISCELLANEOUS #ISN      /* get the value of the foreign key
0620       MOVE PERSONNEL-ID(0610) TO #PERS-NUM  /* get the key
0630     DOEND
0640   ELSE                            /* a check for the unexpected...
0650     DO                            /* a trigger may have been defined wrong
0660       MOVE 913 TO RQ-RESP         /* either ignore or return an error
0670       ESCAPE ROUTINE              /* and exit
0680     DOEND
0690 *
0700 RESET RQ-RESP
0710 *
0720 * Now we check the primary file to see if the value exists. If yes
0730 * then we cannot allow this deletion; hence, we prevent any deletions
0740 * of the foreign key files if a record with the same key exists on the
0750 * primary file.
0760 *
0770 * NOTE: With the setting of RESP, consideration should be given to
0780 *       ambiguities. While the command will receive a response 155
0790 *       (pre-trigger) or 156 (post-trigger), the additions field will
0800 *       contain the error returned from this procedure. The value
0810 *       could be in the form of an Adabas response (1-255) or a
```

```
0820 *       Natural error (e.g., 954 or 935 or 3009); therefore, a
0830 *       user-specified error from the procedure should be something
0840 *       outside these ranges........for simplicity.
0850 *
0860 FIND EMPLOYEES WITH PERSONNEL-ID = #PERS-NUM
0870   MOVE 901 TO RESP                /* it does: delete may not be done
0880   ESCAPE ROUTINE
0890 CLOSE LOOP(0860)
0900 *
0910 END
```

SAMP0005

```
0010 ************************************************************************
0020 *  Application: ADASTP
0030 *   Subprogram: SAMP0005
0040 *     Function: Sample routine of processing by a stored procedure
0050 *               referential integrity - CASCADE
0060 *               (assume that the primary key is on the EMPLOYEES file
0070 *               and foreign keys on the VEHICLES + MISCELLANEOUS files).
0080 * Trigger Defn: Definition on the trigger file is as follows:
0090 *                 File Number ....... 4
0100 *                 File Name ......... EMPLOYEES
0110 *                 Command Type ...... Update
0120 *                 Long Field Name ... PERSONNEL-ID
0130 *                 Adabas Field ...... AA
0140 *                 Field Prty/Seq .... 010
0150 *               Procedure Information
0160 *                 Name (Subpgm)...... SAMP0005
0170 *                 Pre Cmd Select .... Y  (Pre)
0180 *                 Trigger Type ...... P  (Participating)
0190 *                 CALLNAT Params .... C  (Cntl Info + Resp)
0200 *                 RecBuffer Access .. A  (May be Accessed)
0210 *
0220 *      Invoked: Invoked with updates to the EMPLOYEES PERSONNEL-ID
0230 *               Sample routine: SAMPREF2
0240 *       Author: Adabas Development
0250 *         Date: December 1995
0260 ************************************************************************
0270 DEFINE DATA PARAMETER USING STPAPARM
0280            LOCAL     USING STPLRBE      /* parameters for call to STPRBE
0290            LOCAL
0300  01  EMPLOYEES VIEW OF EMPLOYEES
0310    02  PERSONNEL-ID                    /* primary key: field AC
0320  01  MISCELLANEOUS VIEW OF MISCELLANEOUS
0330    02  PERSONNEL-ID                    /* foreign key: field CA
0340  01  VEHICLES  VIEW OF VEHICLES
0350    02  PERSONNEL-ID                    /* foreign key: field AA
```

```
0360  01  FUNC         (A4)
0370  01  #ISN        (P10)
0380  01  #PERS-NUM    (A8)
0390  01  CONTRL-BLK  (A80)
0400  01  REDEFINE CONTRL-BLK
0410    02  CB-FIL1   (A12)
0420    02  CB-ISN     (B4)
0430 END-DEFINE
0440 *
0450 *  First we extract the foreign key information
0460 *  There are two ways to pick this up:
0470 *
0480 *      1) Since the value is in the record buffer, we can use STPRBE to
0490 *         extract the required information; i.e., the primary key
0500 *         information. There are three ways to do this...in this case:
0510 *
0520 *             A) identify the field by its long name; i.e., PERSONNEL-ID
0530 *             B) identify the field by its short name; i.e., AA
0540 *             C) identify the location and length in the record buffer
0550 *
0560 *      2) Get the ISN of the record in the ACB and read this record to
0570 *         extract the required information; i.e., the primary key
0580 *         information. However, this is the old value and cannot be used
0590 *         in this example.
0600 *
0610 *
0620 * OPTION 1A
0630 *
0640 * Function 'GV' -- GET field value using the long field name
0650 *     This enables the caller to obtain information about a specific
0660 *     field which is determined according to the long field name
0670 *     passed in the parameters to STPRBE.
0680 *
0690 RESET #PERS-NUM
0700 MOVE 'GV'           TO FUNC
0710 MOVE 'PERSONNEL-ID' TO RBE-FIELD-NAME /* and identify the corresponding
0720 *                                        field for this file
0730 MOVE 8              TO RBE-LENGTH     /* default or give override length
0740 *                                     /* length in FB could have been used
0750 CALL 'STPRBE' FUNC RBE-AREA #PERS-NUM
0760 PRINT *PROGRAM 'Option 1A returned ..' #PERS-NUM 'resp' RBE-RESP
0770 IF RBE-RESP NE 0                      /* successful ?
0780  DO
0790    PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
0800          RBE-ERROR 'subcode' RBE-SUBCODE 'for func GV'
0810    MOVE RBE-RESP TO RESP              /* indicate this
0820    ESCAPE ROUTINE                     /* and exit
0830  DOEND
0840 *
0850 * OPTION 1B
0860 *
0870 * Function 'GV' -- GET field value using short field name
```

```
0880 *     This enables the caller to obtain information about a specific
0890 *     field which is determined according to the short field name
0900 *     passed in the parameters to STPRBE.
0910 *     NOTE: '**' in field name means user-supplied details in short name
0920 *
0930 RESET #PERS-NUM
0940 MOVE 'GV'     TO FUNC
0950 MOVE '**'     TO RBE-FIELD-NAME    /* special notation for this request
0960 MOVE RQ-FIELD TO RBE-ADA-FIELD     /* get field name that fired the
0970 *                                     trigger from the parm area...OR.....
0980 IF NOT (RQ-FIELD = 'AA')           /* if we know the field....
0990   MOVE 'AA'   TO RBE-ADA-FIELD     /* identify the specific field name
1000 MOVE 8        TO RBE-LENGTH        /* for a maximum length of 8 bytes
1010 CALL 'STPRBE' FUNC RBE-AREA #PERS-NUM
1020 PRINT *PROGRAM 'Option 1B returned ..' #PERS-NUM 'resp' RBE-RESP
1030 IF RBE-RESP NE 0                   /* successful
1040  DO
1050    PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
1060          RBE-ERROR 'subcode' RBE-SUBCODE 'for func GV'
1070    MOVE RBE-RESP TO RESP           /* indicate this
1080    ESCAPE ROUTINE                  /* and exit
1090  DOEND
1100 *
1110 * OPTION 1C
1120 *
1130 * Function 'GR' -- GET RB value using RB offset + length
1140 *     This enables the caller to obtain information based on a
1150 *     certain location; hence, RBE-OFFSET specifies the start
1160 *     position and RBE-LENGTH specifies the length.
1170 *
1180 RESET #PERS-NUM
1190 MOVE 'GR' TO FUNC
1200 MOVE 1    TO RBE-OFFSET            /* start at the beginning
1210 MOVE 8    TO RBE-LENGTH            /* for a max. length of 50 bytes
1220 CALL 'STPRBE' FUNC RBE-AREA #PERS-NUM
1230 PRINT *PROGRAM 'Option 1C returned ..' #PERS-NUM 'resp' RBE-RESP
1240 IF RBE-RESP NE 0
1250  DO
1260    PRINT *PROGRAM 'received an error from the STPRBE routine. Error:'
1270          RBE-ERROR 'subcode' RBE-SUBCODE 'for func GR'
1280    MOVE RBE-RESP TO RESP
1290    ESCAPE ROUTINE
1300  DOEND
1310 *
1320 *  NOTE: Only one of the options need be used to extract the value
1330 *
1340 RESET RQ-RESP
1350 *
1360 * Now, we read the original record, which is not yet changed; hence the
1370 * reason for setting this up as a pre-trigger, to see if the value
1380 * (PERSONNEL-ID in this case) has changed.
1390 *
```

```
1400 MOVE RQ-CB  TO CONTRL-BLK          /* get the original ACB of the A1/4
1410 MOVE CB-ISN TO #ISN                /* extract the ISN of the record
1420 GET EMPLOYEES #ISN                 /* read the, so far, unchanged data
1430 IF PERSONNEL-ID(1420) = #PERS-NUM  /* have the numbers changed?
1440   ESCAPE ROUTINE                   /* no, then exit
1450 *
1460 * Now that we have observed that the primary key has changed, we must
1470 * read all the files with a foreign key and CASCADE the update.
1480 *
1490 FIND VEHICLES WITH PERSONNEL-ID = PERSONNEL-ID(1420) /* Vehicles file
1500  ASSIGN PERSONNEL-ID(1490) = #PERS-NUM
1510  UPDATE (1490)
1520 CLOSE LOOP(1490)
1530 *
1540 FIND MISCELLANEOUS WITH PERSONNEL-ID = PERSONNEL-ID(1420) /* Misc file
1550  ASSIGN PERSONNEL-ID(1540) = #PERS-NUM
1560  UPDATE (1540)
1570 CLOSE LOOP(1540)
1580 *
1590 * Issuing an ET now, is not valid with a participating trigger because
1600 * the originating command (A1/Update) has not yet been executed and
1610 * the originating user expects to do the ET once the update is complete.
1620 * If this ET were done here, the A1/4 (pre-trigger) would receive a
1630 * response 144 because the ISN would be released. If the originating
1640 * user had to do other updates, then a misplaced ET (End Transaction)
1650 * could cause a loss of data integrity across the files.
1660 *
1670 END
```

SAMPREF1

```
0010 ************************************************************************
0020 * Application: Adabas Triggers
0030 *     Program: SAMPREF1 - Example of referential integrity (restrict)
0040 *    Function: SPT routine to delete records from the Vehicles file
0050 *              Invoked with a delete trigger as shown below:
0060 *
0070 *              Trigger Information
0080 *                File Number ....... 3
0090 *                File Name ......... VEHICLES-FILE
0100 *                Command Type ...... Delete
0110 *                Long Field Name ... ** Any Field **
0120 *                Adabas Field ...... **
0130 *                Field Prty/Seq .... ___
0140 *              Procedure Information
0150 *                Name (Subpgm)...... SAMP0004
0160 *                Pre Cmd Select .... Y  (Pre)
0170 *                Trigger Type ...... N  (Non-Participating)
```

```
0180 *                CALLNAT Params .... C  (Cntl Info + Resp)
0190 *                RecBuffer Access .. N  (No RecBuff Access)
0200 *
0210 ************************************************************************
0220 DEFINE DATA LOCAL
0230   01  #NUMBER    (A8)
0240   01  VEHICLES VIEW OF VEHICLES
0250     02  PERSONNEL-ID
0260 END-DEFINE
0270 *
0280 INPUT (AD=TMIL'_' CD=NE)
0290      'Trigger Example for Referential Integrity - RESTRICT' (YEI)
0300   // 'Enter Personnel Number ..' (TU) #NUMBER
0310 *
0320 IF #NUMBER = MASK('.')                    /* exit?
0330   STOP                                    /* yes
0340 IF #NUMBER = ' '                          /* a number must be specified
0350   REINPUT 'Invalid Number specified'
0360 *
0370 FIND VEHICLES WITH PERSONNEL-ID = #NUMBER /* find the record to be deleted
0380   DELETE(0370)                            /* issue the Delete request
0390   END TRANSACTION                         /* finalize the delete
0400   REINPUT 'Record has now been deleted'   /* confirm and restart
0410 CLOSE LOOP
0420 IF *NUMBER(0370) = 0                      /* validate existence of number
0430   REINPUT 'Invalid Personnel Number specified'
0440 *
0450 * Below, any error handling may be done. With a trigger, a procedure
0460 * could return a non-zero response. This would result in the trigger
0470 * command (the Delete in this case) receiving a response 155. Pre-triggers
0480 * receive a response 155 and post-triggers receive a response 156.
0490 *
0500 ON ERROR                            /* handle any errors from the trigger
0510   DO
0520     BACKOUT TRANSACTION             /* release the held record/ISN
0530     INPUT (AD=O CD=YE) 8X '*** Warning ***' (REI)
0540       // 'Personnel Number' (YE) #NUMBER 'NOT deleted' (YEI)
0550        / 'Response' (YE) *ERROR-NR 'received for this request' (YE)
0560       // 4X 'Press Enter to continue' (REI)
0570     STACK TOP COMMAND *PROGRAM     /* return to start of this routine
0580     STOP
0590   DOEND
0600 END
```

SAMPREF2

```
0010 ************************************************************************
0020 * Application: Adabas Triggers
0030 *     Program: SAMPREF2 - Example of referential integrity (Cascade)
0040 *    Function: SPT routine to update records on the Employees file
0050 *              Invoked with an update trigger as shown below:
0060 *
0070 *              Trigger Information
0080 *                File Number ....... 4
0090 *                File Name ......... EMPLOYEES
0100 *                Command Type ...... Update
0110 *                Long Field Name ... PERSONNEL-ID
0120 *                Adabas Field ...... AA
0130 *                Field Prty/Seq .... 10_
0140 *              Procedure Information
0150 *                Name (Subpgm)...... SAMP0005
0160 *                Pre Cmd Select .... Y  (Pre)
0170 *                Trigger Type ...... P  (Participating)
0180 *                CALLNAT Params .... C  (Cntl Info + Resp)
0190 *                RecBuffer Access .. A  (May be Accessed)
0200 *
0210 ************************************************************************
0220 DEFINE DATA LOCAL
0230   01  #NUMBER    (A8)
0240   01  EMPLOYEES VIEW OF EMPLOYEES
0250     02  PERSONNEL-ID
0260     02  FIRST-NAME
0270     02  NAME
0280     02  MIDDLE-NAME
0290 END-DEFINE
0300 *
0310 REPEAT
0320 *
0330 INPUT (AD=TMIL'_' CD=NE)
0340       'Trigger Example for Referential Integrity - CASCADE' (YEI)
0350    // 'Enter Personnel Number ..' (TU) #NUMBER
0360 *
0370 IF #NUMBER = MASK('.')                    /* exit ?
0380   STOP                                    /* yes
0390 *
0400 IF #NUMBER = ' '                          /* a number must be specified
0410   REINPUT 'Invalid Number specified'
0420 *
0430 FIND EMPLOYEES WITH PERSONNEL-ID = #NUMBER /* read the record
0440   INPUT (AD=MIL CD=NE)                    /* show data for doing updates
0450         'Enter Employee Details Below for Update:-' (YEI)
0460     // 'Personnel Number ...' (TU) PERSONNEL-ID
```

```
0470      / 'Last Name ..........' (TU) NAME
0480      / 'First Name .........' (TU) FIRST-NAME
0490      / 'Middle Name ........' (TU) MIDDLE-NAME
0500 *
0510 * Validation of the changes may now be done as required
0520 *
0530   UPDATE(0430)                     /* make the database changes
0540   END TRANSACTION                  /* and finalize them
0550   ESCAPE BOTTOM
0560 CLOSE LOOP
0570 IF *NUMBER(0430) = 0
0580   REINPUT 'Invalid Personnel Number specified'
0590 ELSE
0600   INPUT NO ERASE ////// 4X 'Record has now been updated' (YEI)
0610 *
0620 CLOSE LOOP(0310)                   /* repeat loop
0630 *
0640 * Below, any error handling may be done. With a trigger, a procedure
0650 * could return a non-zero response. This would result in the trigger
0660 * command (the update in this case) receiving a response 155. Pre-triggers
0670 * receive a response 155 and post-triggers receive a response 156.
0680 *
0690 ON ERROR                           /* handle any errors from the trigger
0700   DO
0710     BACKOUT TRANSACTION            /* release the held record/ISN
0720     INPUT (AD=O CD=YE) 8X '*** Warning ***' (REI)
0730       // 'Personnel Number' (YE) #NUMBER 'NOT Updated' (YEI)
0740        / 'Response' (YE) *ERROR-NR 'received for this request' (YE)
0750       // 4X 'Press Enter to continue' (REI)
0760     STACK TOP COMMAND *PROGRAM     /* return to start of this routine
0770     STOP
0780   DOEND
0790 END
```