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.

This chapter covers the following topics:

- Writing Procedures
- Natural Syntax Limitations
- Using the Format and Record Buffers

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 queueing 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 (indication of pre-command triggers) or 156 (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 parameter data area (PDA), 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). 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	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.

Code	Meaning
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.
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).

Code	Meaning
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.