

# Natural under UTM - Part 4

This part of the Natural UTM Interface documentation covers the following topics:

- Accounting for Natural UTM Applications
- Utility Programs for Use with Natural/UTM
- Software Exchange
- UTM TACCLASS Concept - Priority Control
- Generating a Natural UTM Application
- Optimizing Natural UTM Applications
- Several Applications with one Common Natural
- Entering and Defining Dynamic Natural Parameters
- UTM User Restart
- Adabas Priority Control

**Installation** - refer to Installing the Natural UTM Interface in the Natural Installation Guide for Mainframes.

**Notation\_vrs\_or\_vr">Notation vrs or vr:** If used in the following document, the notation *vrs* or *vr* stands for the relevant *version, release, system maintenance level* numbers.

---

## Accounting for Natural/UTM Applications

To better control the use of resources by Natural/UTM applications, accounting records are made available by the user exits ACCINIT and ACCEXIT.

The user exit ACCINIT is activated by the Natural UTM Interface at the beginning of each dialogue step.

The user exit ACCEXIT is activated by the Natural UTM Interface depending on the keyword parameter ACCNT in macro NURENT:

<b>ACCNT=DIAL</b>	The user exit ACCEXIT is activated at the end of each dialogue step.
<b>ACCNT=APPL</b>	The user exit ACCEXIT is activated at each change of application (new Natural logon ID).

In both cases, an accounting record is also provided at the end of the session (FIN system command or TERMINATE statement).

## Structure of the Accounting Record

0 - 7	Logical UTM terminal name	DS	CL8
8 - 15	User ID	DS	CL8
16 - 23	Current Natural application name	DS	CL8
24 - 27	Number of Adabas calls	DS	F
28 - 31	Accumulated message length	DS	F
32 - 35	Elapsed time in Natural including subroutines (milliseconds)	DS	F
36 - 37	Number of pages printed	DS	H
38 - 39	Number of terminal I/O transfers	DS	H
40 - 49	(user area)	DS	CL10
50 - 51	unused	DS	CL2
52 - 55	Adabas command time (milliseconds)	DS	F
56 - 63	Name of last transaction program	DS	CL8

The user area of the accounting record can (if required) be used for additional application-specific accounting information. The accounting area is in the user-specific UTM communication area (Kommunikationsbereich, KB).

The current address of the UTM KBs can be found with the entry "CMKBADR" of macro NATUTM as necessary; otherwise, the operand of the keyword parameter KB of macro NATUTM must be set to "YES". In this case, Natural passes the address of the communication area as the first parameter of every subroutine call.

The user exit routine ACCEXIT can store the accounting records in an Adabas file, in a shared sequential PAM dataset or in a task-specific SAM dataset. The program ACCEXIT shows an example of the method for storing accounting records; see Software Exchange.

## Utility Programs for Use with Natural/UTM

Several utility programs are provided for use with Natural under UTM. The following rules apply to their usage:

- The Natural and UTM macro libraries must be used when assembling these utilities.
- When a particular program is to be used:
  - its name must be specified with the keyword parameter LINK or LINK2 of macro NATUTM
  - and the program itself must be linked with the front-end part of the Natural/UTM application.

A detailed description, including the interface, valid parameter values and a summary of the logic, can be found in each program's maintenance log.

### Utility Program NATDUE

The program NATDUE can be used to find out within a Natural program whether the user has entered data in the current dialogue step or whether merely EM/DÜ or DÜ was entered.

The utility program INPTEX must be used if NATDUE is to be called. The program INPTEX satisfies the user exit INPTEX in the format exit module FREXIT and checks at each dialogue step whether data were entered. According to the result of this test, a flag that is subsequently interrogated by the program NATDUE is set in the communication area (Kommunikationsbereich, KB).

#### Example of a Natural Program that Calls NATDUE:

```
* PROG1 - EXAMPLE FOR CALLING THE SUBROUTINE 'NATDUE'
RESET P1(A1) ...
...
INPUT USING MAP ...
CALL 'NATDUE' P1
IF P1 = 'Y' DO ... /* INPUT FROM USER
IF P1 = 'N' DO ... /* NO INPUT FROM USER
IF P1 = 'E' DO ... /* ERROR
...
END
```

## Utility Program INPTEX

The utility program INPTEX satisfies the user exit of the same name in the format exit FREXIT.

**Important:** INPTEX must be linked with the front-end part of the Natural/UTM application.

**Warning:** Any modifications that can be made to this program, for example, ignoring data entered in a particular line on the terminal screen, are made at the user's risk.

Function: This program checks each input message for the presence of input from the terminal, or whether merely EM/DÜ or DÜ was pressed.

It is not necessary to define the program name INPTEX with the keyword parameter LINK or LINK2 of macro NATUTM.

## Utility Program NATPRNT

The program NATPRNT provides the following special service functions for operating local printers:

- accepting the logical name of the target printer;
- verifying the printer name against a list of valid printer names;
- setting a marker for building variable length print records.

## Utility Program UTMTAC

The program UTMTAC, which can be called from a Natural program, yields the current UTM TAC. This makes it possible for a central Natural program to perform UTM TAC-controlled "navigation" within a Natural/UTM application.

## Utility Program TACSWTCH

The utility program TACSWTCH is a macro which can be used to dynamically assign a UTM TAC for a PEND PR(OGRAM) from within a Natural program. The specified UTM TAC is checked against the generated UTM table and saved accordingly. Also, information can be passed to the PEND PR(OGRAM). To use this utility, proceed as follows:

1. Define the valid UTM TACs and assemble the TACSWTCH macro:

For Example: TACSWTCH TAC=(tac1,tac2,tac3,...tacn)

These TACs have to be defined in KDCDEF as well, and for the generation of KDCROOT they have to be assigned to the corresponding UTM partial programs.

1. Define the program TACSWTCH with the keyword parameters LINK to LINK4 in macro NATUTM.
2. Link program TACSWTCH to the front-end part of the Natural UTM Interface.
3. Interface description: CALL 'TACSWTCH' P1 [P2] P3

P1 (A8)	Contains the UTM TAC to be used for a PEND PR.
P2 (An)	Is optional and contains the length and data of a message to the PEND PR  The structure of P2 is: LLLDDD.....  LLL = Message length (3 digits, no length field); minimum length: 000, maximum length: 160.  DDD = Message area.
P3 (A1)	Has two functions:  On call and if P3 contains the value "G" (Go), the PEND PR(OGRAM) is executed at the next Natural output (INPUT, WRITE, DISPLAY). After calling the Natural UTM Interface with PEND PR, the Natural session is continued where it had been suspended, which means that the last output is displayed to the user.  On return, P3 contains the return code from TACSWTCH.  Possible return codes are:  0 = The operation has been executed without error. 1 = TAC has not been found in the TAC table. 2 = Message length was less than "000". 3 = Message length was over "160".  Once TACSWTCH has been called without error, a PEND PR(OGRAM) can be executed by either issuing a FIN command or with a TERMINATE statement or by activating the function key for PEND PR; see the keyword parameter PRKEY.

## Special TACSWTCH Functions

You can use the first TACSWTCH parameter with the following values:

<b>RESET</b>	The UTM TAC currently available will be cleared, that is, the session will be terminated with PEND FI.
<b>GETP</b>	Data will be moved from the print buffer to the adequate data area of the calling Natural program.
<b>GETU</b>	Data will be moved from the KB user extension to the adequate data area of the calling Natural program.

The first two bytes (format: binary) in the print buffer or in the KB user extension must contain the data length (including these first two bytes).

<b>PUTP</b>	Data will be moved from the adequate data area of the calling Natural program to the print buffer.
<b>PUTU</b>	Data will be moved from the adequate data area of the calling Natural program to the KB user extension.

The first two bytes (format: binary) in the data area of the Natural program must contain the data length (including these first two bytes). The data will be moved including the first two bytes.

**Example for PUTP and GETP:**

```

DEFINE DATA LOCAL
01 P1(A8) /* FUNCTION CODE/UTM TAC
01 P2(A252) /* FIRST PART OF DATA AREA
01 REDEFINE P2
02 P21(B2) /* DATA LENGTH INCLUDING FIRST TWO BYTES
02 P22(A250)
01 A1(A250) /* SECOND PART OF DATA AREA
01 P3(N1) /* RETURN CODE
END-DEFINE
... /* PROGRAM LOGIC
MOVE 'PUTP' TO P1 /* MOVE FUNCTION CODE FOR TACSWTCH
MOVE 502 TO P21 /* MOVE TOTAL LENGTH OF DATA
CALL 'TACSWTCH' P1 P2 P3 /* PUT DATA INTO PRINT BUFFER
IF P2 NE 0 /* RETURN CODE CONTROLLING
DO... /* ERROR LOGIC
MOVE 'NAT1' TO P1 /* MOVE ADEQUATE UTM TAC
MOVE 'G' TO P3 /* EXECUTE PEND PR WITH TAC NAT1
CALL 'TACSWTCH' P1 P3
IF P3 NE 0 /* RETURN CODE CONTROLLING
DO... /* ERROR LOGIC
INPUT ' ' /* DUMMY MESSAGE FOR DRIVER CONTROL

```

Now the Natural/UTM driver gets control and runs with the following logic:

1. It ignores the dummy message (INPUT ' ').
2. MPUT with LENGTH=0 and PEND PR with TAC 'NAT1' for the UTM partial program.
3. The UTM partial program gets the Natural program data through the print buffer. The print buffer is located in the UTM SPAB and the address of the print buffer is defined in the field 'KBAPBUFF', which is located in the UTM KB:
  - It moves data for the Natural program into the print buffer (the first two bytes must contain the data length in binary format, including the two-byte length field).
  - It executes an MPUT with LENGTH=0 and a PEND PR with the TAC defined for the Natural/UTM driver.
4. The Natural/UTM driver gets control (INIT/MGET).
5. It simulates ONLY ENTER for Natural.
6. It resumes with Natural as follows:

```

MOVE 'RESET' TO P1 /* MOVE FUNCTION CODE FOR TACSWTCH
CALL 'TACSWTCH' P1 P3 /* RESET PEND PR TAC (NAT1)
IF P3 NE 0 /* RETURN CODE CONTROLLING
DO... /* ERROR LOGIC
MOVE 'GETP' TO P1 /* MOVE FUNCTION CODE FOR TACSWTCH
CALL 'TACSWTCH' P1 P2 P3 /* GET DATA FROM PRINT BUFFER
IF P3 NE 0 /* RETURN CODE CONTROLLING
DO... /* ERROR LOGIC
... /* PROGRAM LOGIC
END

```

If the keyword parameter **KBSAVE** of macro **NATUTM** is set to **YES**, the called UTM partial program may use the UTM KB (from the end of the header plus first twelve bytes). In this case, the UTM KB will be saved (beginning from KB header plus first twelve bytes) with **SPUT** and will be refreshed with **SGET**.

When defining UTM transaction codes for the transaction logic between Natural and other UTM partial programs, the following rule applies:

For a **PEND PR(ogram)** from another UTM partial program to the Natural/UTM driver, the preceding start TAC may never be used. The fact that the Natural/UTM driver was called by a **PEND PR(ogram)** can only be recognized if the contents of the preceding start TAC in field **KCTACVG** are different from the current TAC in field **KCTACAL**. (Normally, field **KCTACVG** contains the TAC with which the user has entered the application.)

## Software Exchange

Software AG's customers have developed programs that meet certain specific needs found in their Natural/UTM applications. These programs are made available to all interested users via the "Software Exchange". This also applies to programs developed by Software AG that demonstrate example solutions to particular problems.

These programs, which are available free of charge, are not maintained by Software AG. The documentation of each program is usually included in the maintenance log of the source listing.

### Program XAMDUSA

This program saves and restores the current user-specific WORKING-STORAGE SECTION of the calling COBOL program.

This enables user-specific data areas, for example tables, to be accessible over many dialogue steps and without regard to the UTM task in which the user is currently running. The data are saved in a PAM file using logical/physical chained PAM-I/O.

### Program UTMCOB

Program UTMCOB is an example of a user-specific UTM partial program within a Natural/UTM application. It shows the fundamental logical structure of a program that, as a UTM partial program:

- Can be activated by the user by associated UTM TACs.
- Activates the Natural UTM Interface and hence the Natural application by means of PEND PR(OGRAM) with dynamic Natural parameters.
- Can be activated from the Natural UTM Interface by means of PEND PR(OGRAM).

See also Calling UTM Chained Partial Programs.

### Program UTMNAV

Program UTMNAV is another example of a user-specific UTM partial program within a Natural/UTM application:

- It can be activated by the user or with PEND PR(OGRAM) by the associated UTM TAC.
- It interprets passed messages as dynamic Natural parameters.
- It provides screen output of information on the program logic.
- Previously received screen input (Natural dynamic parameters) is sent with MPUT and passed to the Natural UTM Interface with PEND PR(OGRAM).

Program UTMNAV contains an example of how the UTM KB can be used as a "common" user area.

## Program NUEXAMPL

Program NUEXAMPL is an example of a user-specific UTM partial program which can exchange data with a Natural program. The program logic of NUEXAMPL and of the calling Natural program is described in the maintenance log of NUEXAMPL.

## Program ACCEXIT

Program ACCEXIT is an example of a program that saves accounting data on a shared ISAM dataset. The user exits ACCEXIT and SHUTEX2 of the Natural UTM Interface are used. See also Accounting for Natural/UTM Applications.

## Program TABMOD

The program TABMOD, which can be called from a Natural program, performs the following functions:

- load data records, for example a table, into a common memory pool using a unique key when an application is started and whilst an application is running;
- transfer data records according to the requirements of the calling Natural program.

This makes it possible to load frequently-needed data into storage once only and then keep them resident.

TABMOD is available as a macro in the library NUT $nnn$ .MAC. It contains all information necessary for its installation and usage.

## UTM TACCLASS Concept - Priority Control

Natural programs can allocate UTM TAC classes to optimize resource control using the UTM TACCLASS concept in a Natural/UTM application.

The following procedure should be followed when generating the Natural/UTM application and creating the Natural program:

### **Step 1: Specify UTM TACs and TAC Classes in the KDCDEF and KDCROOT Definitions**

```

Example:

OPTION GEN=ALL,ROOTSRC=INPUT.KDCROOT.KDCNATP
ROOT KDCNATP
MAX APPLINAME=NATUTM,APPLIMODE=S,KDCFILE=(NATUTM,S)
MAX KB=400,SPAB=8192,NB=5120,TRMSGLTH=5120
MAX TASKS=10
MAX ASYNTASKS=3
...
EXIT PROGRAM=NUSTART,USAGE=START
EXIT PROGRAM=NUSTART,USAGE=SHUT
EXIT PROGRAM=FREXIT,USAGE=FORMAT
...
DEFAULT PROGRAM COMP=ASSEMB
PROGRAM NUSTART
PROGRAM FREXIT
PROGRAM NUERROR
PROGRAM KDCADM,COMP=SPL4
...
DEFAULT TAC TYPE=D,PROGRAM=NUSTART,EXIT=NUERROR,CALL=BOTH,...
TAC NAT,TIME=(3600000,5400),TACCLASS=1,...
TAC NAT1,TIME=(3600000,5400),TACCLASS=2,...
...
DEFAULT TAC TYPE=A,PROGRAM=NUSTART,EXIT=NUERROR,CALL=FIRST,...
TAC NATAS,TACCLASS=9
TAC NATAS1,TACCLASS=10
...
TACCLASS 1,TASKS=3
TACCLASS 2,TASKS=1
TACCLASS 9,TASKS=2
TACCLASS 10,TASKS=1
...
END
    
```

See also the Siemens documentation "UTM Generierung und Administration" (UTM Generation and Administration).

**Notes on the UTM TACs Defined**

<b>NAT</b>	This is the UTM TAC for less resource-intensive synchronous transactions; that is, transactions of short duration.
<b>NAT1</b>	This is the UTM TAC for more resource-intensive synchronous transactions; that is, transactions of longer duration.
<b>NATAS</b>	This is the UTM TAC for less resource-intensive asynchronous transactions.
<b>NATAS1</b>	This is the UTM TAC for more resource-intensive asynchronous transactions.

## Step 2: The Structure of the UTM Start Job

The name of the job is "EN.NATUTM".

**Example:**

```
/.NATUTM LOGON Natural,E,,TIME=10000
/SYSFILE SYSOUT=PROT.UTMSTAT
/FILE NATUTM.KDCA,LINK=KDCFILE
/ERASE NATUTM.PRINTCONTROL
/STEP
/FILE LOG.NATUTM,LINK=SYSLOG
/FILE NATUTM.SWAPFILE,LINK=PAMNAT,SHARUPD=Y
/SYSFILE TASKLIB=NAT210.MOD
/.REPEAT EXEC NATUTM.E
.UTM START FILEBASE=NATUTM
START TASKS=7
START ASYNTASKS=3
START STARTNAME=EN.NATUTM
.UTM END
/SKIP .REPEAT
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/STEP
/SYSFILE SYSLST=(PRIMARY)
/CAT NATUTM.PRINTCONTROL,SHARE=YES
/PRINT LST.NATUTM.,SPACE=E
/ERASE LST.NATUTM.
/STEP
/LOGOFF NOSPOOL
```

### Step 3: Change the TAC Class of Synchr. Transactions by a Natural Program

The TAC-class of synchronous UTM transactions can be changed by a Natural program with the statements:

```
CALL 'NATTAC' operand1 [operand2] [operand3]
INPUT 'TACCLASS'
```

<i>operand1</i>	Must contain the value "S=n", where "S" denotes "synchronous" and "n" is an integer value (0 - 4) that denotes the priority level of the transaction in subroutine NATTAC's table of transaction codes for synchronous TACs.
	If "n" is 0, the table of transaction codes is not used. The TAC to be used is passed explicitly in <i>operand2</i> when NATTAC is called.
	If "n" is a value in the range 1 - 4, the priority level of the desired TAC is taken from the appropriate keyword parameter TCLS1 - TCLS4 (for synchronous transactions) or TCLA1 - TCLA4 (for asynchronous transactions).
	If the subroutine NATTAC detects an error in <i>operand1</i> , it returns immediately to the calling program with an error code in <i>operand1</i> :
	E01: The first two characters of <i>operand1</i> were neither "S=" nor "A=".
	E02: The third character of <i>operand1</i> was <0 or >4.
	E03: No UTM TAC was defined for the specified priority level when the Natural/UTM application was generated, which means that the corresponding keyword parameter (TCLSn or TCLA <sub>n</sub> ) has the value "-".
<i>operand2</i>	Optional. Must contain the UTM TAC for the desired TAC class if the third character of <i>operand1</i> is "0".
<i>operand3</i>	Optional. Must contain the value "Y" if the current user's subsequent dialogue is to be executed with the UTM TAC defined in <i>operand1</i> or <i>operand2</i> . If <i>operand3</i> is omitted when NATTAC is called, or if <i>operand3</i> has some value other than "Y", the START transaction code for the current user is used again with the first terminal output (standard function). If <i>operand3</i> has the value "Y" when NATTAC is called, further processing for the current user takes place with the UTM TAC specified in <i>operand1</i> (implicit) or <i>operand2</i> (explicit).

The statement INPUT 'TACCLASS' does not perform any terminal I/O; its function is merely to control the TACCLASS allocation.

Alternatively, a Natural program can call the Natural subprogram "NATTAC" with a CALLNAT statement. For this, the INPUT 'NATTAC' statement is omitted; the operands are the same as for the CALL statement (see above):

```
CALLNAT 'NATTAC' operand1 [operand2] [operand3]
```

This procedure can be used with synchronous as well as asynchronous transactions. NATTAC is contained in the library SYSTEM.

**Example 1:**

A Natural program that allocates a UTM TAC explicitly to assign a new TAC class and then changes over to the START UTM TAC.

```

* TACCLASS - EXAMPLE FOR A TACCLASS SWITCH
RESET CONTROL(A3) NEWTAC(A8) NR(N3)
REDEFINE CONTROL (ERRFLD(A1))
INPUT 'TEST FOR A TACCLASS SWITCH - NEW TAC: NAT1' IFELD(A1)
MOVE 'S=0' TO CONTROL /* SYNCHR. TAC, EXPLICIT --> Note 1
MOVE 'NAT1' TO NEWTAC /* SET NEW TAC --> Note 2
CALL 'NATTAC' CONTROL NEWTAC /* INVOKE TAC SWITCH --> Note 3
IF ERRFLD = 'E' DO /* ERROR CHECK --> Note 4
DISPLAY 'ERROR' CONTROL 'FROM NATTAC'
TERMINATE
DOEND
INPUT 'TACCLASS' /* ACTIVATE NEW TAC --> Note 5
READ (50) AUTOMOBILES BY MAKE /* NOW IN NEW TACCLASS --> Note 6
ADD 1 TO NR
WRITE NOTITLE NOHDR NR MAKE MODEL /* START TAC IS USED --> Note 7
LOOP
ON ERROR DISPLAY 'ERROR IN PROGRAM TACCLASS'
END
    
```

Note	
1	The value "S=0" indicates that it is a synchronous transaction and that the TAC is passed explicitly in the second parameter of the CALL 'NATTAC', which means that the TAC table is not used.
2	The new TAC (NAT1) is set up for the call to NATTAC.
3	The change of TAC class is initialized by calling NATTAC.
4	An error check is performed after returning from subroutine NATTAC.
5	A "pseudo"-MPUT and a "PEND PA" are executed with the new TAC.
6	The program is now running in the TAC class for NAT1.
7	When the first terminal output starts, the START UTM TAC takes effect again.

In this example, the AUTOMOBILE file is read using the UTM TAC NAT1. When the first terminal output begins, the START UTM TAC (NAT) takes effect again.

**Internal Processing Logic:** When NATTAC is called, a flag is set in the UTM communication area (Kommunikationsbereich, KB) indicating that a change of TACCLASS is pending.

The UTM TAC passed by the program is also stored in the user-specific communication area. The operation INPUT 'TACCLASS' causes terminal output from Natural, which causes the UTM interface to issue an MPUT and a PEND 'PA' with the new UTM TAC (the message is received by the Natural UTM Interface itself). When the message is received (in the new TAC class), the presence of the TACCLASS change flag causes the interface to simulate an ETX/DÜ in its input area. Further processing runs in the new TAC class.

Depending upon the value of the operand in the previous call of NATTAC, the first message sent to the terminal can cause an MPUT and a PEND 'PR' with the user's START UTM TAC; that is, a further TACCLASS change may take place.

**Example 2:**

A Natural program that allocates a UTM TAC explicitly to assign a new TAC class without changing over to the START UTM TAC.

```
* TACCLAS1 - EXAMPLE FOR A TACCLASS SWITCH
RESET CONTROL(A3) NEWTAC(A8) SWOFF(A1)
INPUT 'TEST FOR A TACCLASS SWITCH - NEW TAC: NAT1' IFELD(A1)
MOVE 'S=0' TO CONTROL /* SYNCHR. TAC, EXPLICIT
MOVE 'NAT1' TO NEWTAC /* SET NEW TAC
MOVE 'Y' TO SWOFF /* NO RESET TO START TAC
CALL 'NATTAC' CONTROL NEWTAC SWOFF /* INVOKE TAC SWITCH
INPUT 'TACCLASS' /* ACTIVATE NEW TAC
FETCH 'TACCLAS2' /* NOW IN NEW TACCLASS
END
* TACCLAS2 - THIS PROGRAM IS FETCHED FROM PROGRAM TACCLAS1
RESET NR(N3)
READ (25) AUTOMOBILES BY MAKE /* TACCLASS IS NAT1
ADD 1 TO NR
WRITE NOTITLE NOHDR NR MAKE MODEL HORSEPOWER YEAR
LOOP
FETCH 'MAINMENU' /* TACCLASS = NAT1
END
```

In this example, processing is assigned to a new TAC class with TAC NAT1. Switching to the user's START UTM TAC is avoided by the presence of the third parameter (SWOFF) in the call to NATTAC with value "Y".

It is also possible to perform several TACCLASS changes within one Natural program.

**Example 3:**

A Natural program that performs two explicit and one implicit TACCLASS changes.

```
*TACMULT - EXAMPLE FOR TWO TACCLASS SWITCHES IN ONE PROGRAM
RESET CONTROL(A3) NEWTAC(A8) SWOFF(A1) NR(N4)
INPUT 'TEST FOR 2 TACCLASS SWITCHES' IFELD(A1)
MOVE 'S=0' TO CONTROL /* SYNCHR. TAC, EXPLICIT
MOVE 'NAT1' TO NEWTAC /* SET NEW TAC
MOVE 'Y' TO SWOFF /* NO RESET TO START TAC
CALL 'NATTAC' CONTROL NEWTAC SWOFF /* INVOKE TAC SWITCH
INPUT 'TACCLASS' /* ACTIVATE NEW TAC
READ (50) AUTOMOBILES BY MAKE /* NOW IN NEW TACCLASS
ADD 1 TO NR
WRITE NR MAKE MODEL YEAR
LOOP
EJECT /* ACTIVATE NEW OUTPUT *****
MOVE 'S=0' TO CONTROL /* SYNCHR. TAC, EXPLICIT
MOVE 'NAT2' TO NEWTAC /* SET NEW TAC
CALL 'NATTAC' CONTROL NEWTAC /* INVOKE TAC SWITCH
INPUT 'TACCLASS' /* ACTIVATE NEW TAC
READ (100) AUTOMOBILES BY MAKE /* NOW IN NEW TACCLASS
WRITE MAKE MODEL YEAR /* NOW START TAC IS USED
LOOP
ON ERROR DISPLAY 'ERROR IN PROGRAM TACMULT'
END
```

The UTM TAC NAT2 has not been considered in the preceding examples; it must be defined in KDCROOT and KDCDEF.

If an explicit TACCLASS change is to take place after a WRITE, PRINT or DISPLAY statement, an EJECT must be issued before assigning the new TAC. This operation performs an unconditional output to the terminal before executing the INPUT 'TACCLASS'. Instead of the EJECT, the following statements can be used:

```
STACK TOP DATA 'A'
INPUT A(A1)
```

This sequence also performs an unconditional output to the terminal before executing the INPUT 'TACCLASS'.

#### Example 4:

A Natural program that allocates a UTM TAC implicitly to assign a new TAC class and then changes over to the START UTM TAC. This example uses the TAC table for synchronous transactions in the subroutine NATTAC.

```
* TACIMP1 - EXAMPLE FOR AN IMPLICIT TACCLASS SWITCH
RESET CONTROL(A3) NR(N3)
REDEFINE CONTROL (ERRFLD(A1))
INPUT 'TEST FOR AN IMPLICIT TACCLASS SWITCH' IFELD(A1)
MOVE 'S=1' TO CONTROL /* USE 1ST TAC IN TABLE --> NOTE
CALL 'NATTAC' CONTROL /* INVOKE TAC SWITCH
IF ERRFLD = 'E' DO /* ERROR CHECK
DISPLAY 'ERROR' CONTROL 'FROM NATTAC'
TERMINATE
DOEND
INPUT 'TACCLASS' /* ACTIVATE NEW TAC
READ (100) AUTOMOBILES BY MAKE /* NOW IN NEW TACCLASS
ADD 1 TO NR
WRITE NOTITLE NOHDR NR MAKE MODEL /* START TAC IS USED
LOOP
ON ERROR DISPLAY 'ERROR IN PROGRAM TACIMP1'
END
```

The value "S=1" indicates that it is a synchronous transaction and that the TAC is to be taken from the first entry in the TAC table. This is the TAC that was defined as the value of the operand of the keyword parameter TCLS1 (default value: NAT1).

The third character of the first parameter in the CALL 'NATTAC' indicates which of the four keyword parameters TCLS1 to TCLS4 applies.

**Example 5:**

A Natural program that allocates a UTM TAC implicitly to assign a new TAC class but does not change over to the START UTM TAC. This example uses the TAC table for synchronous transactions in the subroutine NATTAC, and processing continues with this TAC.

```
* TACIMP2 - EXAMPLE FOR AN IMPLICIT TACCLASS SWITCH
RESET CONTROL (A3) SWOFF(A1) NR(N3)
REDEFINE CONTROL (ERRFLD(A1))
MOVE 'S=4' TO CONTROL /* USE 4TH TAC IN TABLE --> NOTE
MOVE 'Y' TO SWOFF /* NO RESET TO START TAC
CALL 'NATTAC' CONTROL SWOFF /* INVOKE TAC SWITCH
IF ERRFLD = 'E' DO /* ERROR CHECK
DISPLAY 'ERROR' CONTROL 'FROM NATTAC'
TERMINATE
DOEND
INPUT 'TACCLASS' /* ACTIVATE NEW TAC
READ (100) AUTOMOBILES BY MAKE /* NOW IN NEW TACCLASS
ADD 1 TO NR
WRITE NR MAKE MODEL YEAR
LOOP
ON ERROR DISPLAY 'ERROR IN PROGRAM TACIMP2'
END
```

The value "S=4" indicates that it is a synchronous transaction and that the TAC is to be taken from the fourth entry in the TAC table. This is the TAC that was defined as the value of the operand of the keyword parameter TCLS4 (default value: NAT4).

The TAC NAT4 is not defined in the examples of KDCROOT and KDCDEF; in practice, the user must supply suitable definitions.

Using the TAC table has the advantage that the UTM TAC does not have to be coded explicitly in the Natural program. The Natural programs contain merely the relative priority "weights" of the transactions to be executed. The system administrator can allocate and change the names of the UTM TACs without having to change the Natural programs.

For testing Natural programs with TACCLASS change for synchronous transactions, please note the following: To verify correct operation of the TACCLASS change, the Natural program can be tested without the statement(s) "CALL 'NATTAC' *operand1* (*operand2*) (*operand3*)". If the INPUT 'TACCLASS' statement produces only the output 'TACCLASS' on the terminal, the program is correct. The operand(s) for the call to NATTAC must be set correctly. The UTM processing terminates with error code KM01 whenever a UTM TAC that is not defined in KDCROOT and KDCDEF is used.

## Step 4: Allocation of TAC Classes for Asynchronous Transactions within one Natural/UTM Application

The TAC class for asynchronous transactions within a Natural/UTM application can be changed with the statement:

```
CALL 'NATTAC' operand1 [operand2]
```

<i>operand1</i>	Must contain the value "A= <i>n</i> ", where "A" denotes "asynchronous" and " <i>n</i> " is an integer in the range from 0 to 4 that denotes the priority level of the transaction in subroutine NATTAC's table of transaction codes for asynchronous TACs. The form of the operand is analogous to the form of the operand for synchronous transactions.
<i>operand2</i>	Optional. Contains the UTM TAC for the required TAC class if <i>operand1</i> has the value "A=0".

All UTM TACs for asynchronous transactions must begin with the character string which is defined as unique identifier for asynchronous TACs in parameter ASYNTAC of macro NATUTM. Conversely, the UTM TACs for synchronous transactions must not begin with this string.

### Example 6:

A Natural program that performs initialization for asynchronous transaction processing, using the UTM TAC NATAS. This is the standard TAC for asynchronous transactions. See also the description of the keyword parameter ASYNTAC of macro NATUTM.

```
* STARTAS - EXAMPLE FOR ASYNCHRONOUS TRANSACTION WORKING
* WITHIN ONE APPLICATION - USING THE STANDARD TAC FORMAT LS=145
RESET PARM1(A144) PRDEST(A8) LTDEST(A8)
MOVE 'PRINTER1' TO PRDEST
MOVE *INITID TO LTDEST
COMPRESS 'SENDER=' PRDEST ',OUTDEST=' LTDEST ','
'MENU=F,STACK=(LOGON APPL1;READAUTO)' INTO PARM1
LEAVING NO
CALL 'NATASYN'
SET CONTROL 'H'
WRITE NOTITLE NOHDR PARM1
INPUT 'ASYNTASK INVOKED - HOPEFULLY' IFELD(A1)
END
```

**Example 7:**

A Natural program that initializes asynchronous transaction processing and allocates the UTM TAC NATAS1 for assignment to another TAC class.

```
* STASTAC - EXAMPLE FOR ASYNCHRONOUS TRANSACTION WORKING
* WITHIN ONE APPLICATION
* AND SWITCH TO A NEW TACCLASS
FORMAT LS=145
RESET PARM1(A144) PRDEST(A8) LTDEST(A8) CONTROL(A3) NEWTAC(A8)
REDEFINE CONTROL (ERRFLD(A1))
MOVE 'PRINTER1' TO PRDEST
MOVE *INIT-ID TO LTDEST
COMPRESS 'SENDER=' PRDEST ',OUTDEST=' LTDEST ', '
'MENU=F,STACK=(LOGON APPL1;READAUTO)' INTO PARM1
LEAVING NO
MOVE 'A=0' TO CONTROL /* ASYNCHR. TAC, EXPLICIT --> NOTE
MOVE 'NATAS1' TO NEWTAC /* SET NEW TAC
CALL 'NATTAC' CONTROL NEWTAC /* INVOKE TAC SWITCH
IF ERRFLD = 'E' DO /* ERROR CHECK
DISPLAY 'ERROR' CONTROL 'FROM NATTAC'
TERMINATE
DOEND
CALL 'NATASYN' /* INVOKE ASYNCHRONOUS TAC
SET CONTROL 'H'
WRITE NOTITLE NOHDR PARM1
INPUT 'ASYNAC INVOKED - HOPEFULLY' IFELD(A1)
END
```

The value "A=0" indicates that it is an asynchronous transaction and that the TAC is passed explicitly in the second parameter of the CALL 'NATTAC', which means that the TAC table is not used.

```
MOVE 'A=1' TO CONTROL
CALL 'NATTAC' CONTROL
```

The procedure for using the TAC table (see keyword parameters TCLA1 - TCLA4 in the section Keyword Parameters of Macro NATUTM) corresponds to the procedure for synchronous transactions.

An example of the program that is to be executed asynchronously (READAUTO):

```
* READAUTO - ASYNCHRONOUS Natural PROGRAM
READ (75) AUTOMOBILES BY MAKE
WRITE MAKE MODEL HORSEPOWER BODY-TYPE YEAR
LOOP
ON ERROR TERMINATE
TERMINATE
END
```

**Example 7 (continued):**

```

*+-----+
*I PTERM 9750 DEFINITION I
*+-----+
DEFAULT PTERM PRONAM=VR , PTYPE=T9750 , TERMN=FE , CONNECT=N
PTERM DFDSS001 , LTERM=DF97501
PTERM DFDSS002 , LTERM=DF97502
PTERM DFDSS003 , LTERM=DF97503
*+-----+
*I LTERM DEFINITION I
*+-----+
DEFAULT LTERM USAGE=D , STATUS=ON , ANNOAMSG=Y , RESTART=YES
LTERM=DF97501
LTERM=DF97502
LTERM=DF97503
*+-----+
*I SFUNC DEFINITION I
*+-----+
SFUNC F1 , RET=21Z
SFUNC F2 , RET=22Z
SFUNC F3 , RET=23Z
SFUNC F4 , RET=24Z
SFUNC F5 , RET=25Z
SFUNC K1 , RET=26Z
SFUNC K2 , RET=27Z
SFUNC K3 , RET=28Z
SFUNC K4 , RET=29Z
END

```

The desired UTM TAC must always be allocated in the Natural program that initializes the asynchronous transaction processing (the use of the standard TAC for asynchronous transaction processing is an exception; see the description of the keyword parameter ASYNTAC in the macro NATUTM). The program that is to be executed asynchronously then runs in the desired TAC class. Since each asynchronous Natural program must be ended with the TERMINATE statement, the UTM DC transaction is also ended (PEND 'FI') when the program ends.

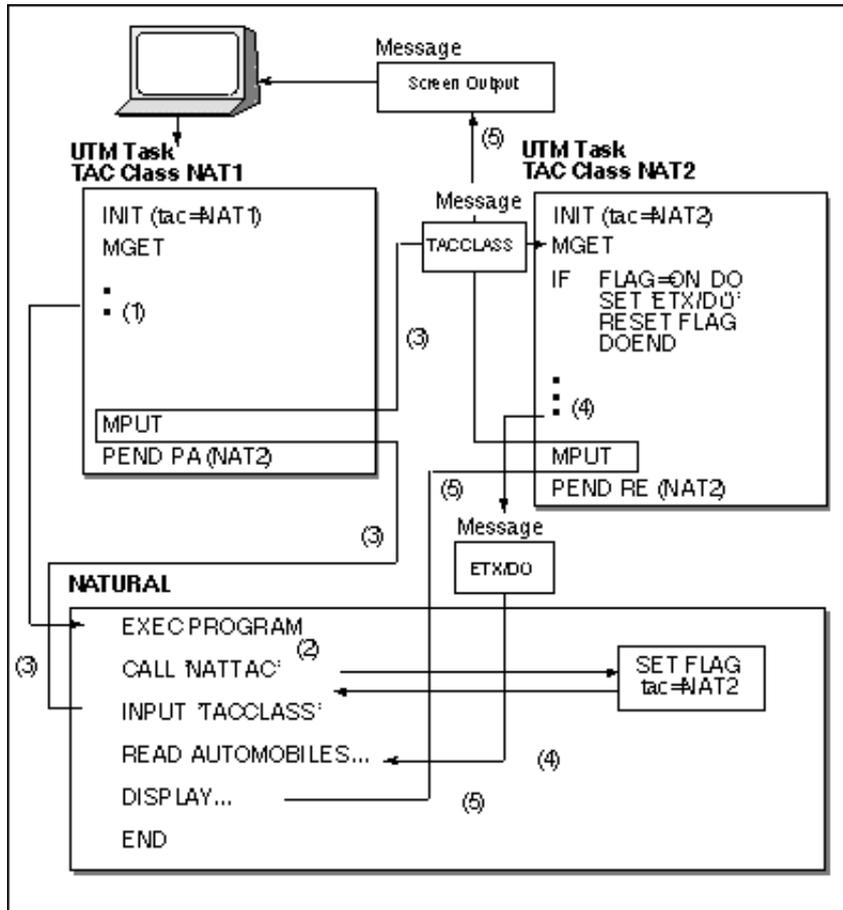
The program that initializes the asynchronous transaction processing always runs in a synchronous transaction. Thus it is feasible to perform a change of TACCLASS using the procedure for synchronous transactions. This change can take place before allocating the asynchronous TACs or after initializing the asynchronous transaction (INPUT statement).

### **Step 5: Assign the TAC Class for Asynchronous Transactions between two Natural/UTM Applications**

It is not necessary to call NATTAC for asynchronous transaction processing between two Natural/UTM applications. The necessary UTM TAC is allocated explicitly in the Natural program; see also Asynchronous Transaction Processing.

## **UTM TACCLASS Switch**

The following figure illustrates the logic of a UTM TACCLASS switch for synchronous transactions:



## Generating a Natural/UTM Application

The following programs and macros must be assembled to generate a Natural/UTM application:

<b>KDCROOT</b>	UTM interface module.
<b>NATUTM</b>	Front-end part of the Natural UTM Interface.
<b>BS2STUB</b>	Common memory pool definition.
<b>FREXIT</b>	Format exit module (only if the default parameter is to be changed).
<b>NURENT</b>	Reentrant part of the Natural UTM Interface.
<b>NTPRM</b>	Natural parameter module.
<b>NTSWPRM</b>	Swap pool parameter module.

This list does not include the utility programs of the Natural UTM Interface.

The following example shows how to generate an application.

```

OPTION GEN=ALL,ROOTSRC=INPUT.KDCROOT.KDCNATP
ROOT KDCNATP
MAX APPLINAME=NATUTM,APPLIMODE=S,KDCFILE=(NATUTM,S)
MAX KB=400,SPAB=8192,NB=5120,TRMSGLTH=5120
MAX TASKS=10,ASYNTASKS=3
MAX GSSBS=0,KSSBS=1
MAX LOGACKWAIT=600,RESWAIT=(600,1200),TERMWAIT=(1800,0)
MAX PGPOOL=(88,80,95),CONRTIME=2,RECBUF=(400,2048)
MAX DPUTLIMIT1=(001,23,59,59),DPUTLIMIT2=(001,23,59,59)
MAX LPUTLTH=0
*+-----+
*I EXIT DEFINITIONS: STARTUP (CSECT NAME OF NATUTM) I
*I SHUTDOWN (CSECT NAME OF NATUTM) I
*I FORMAT (FREXIT) I
*+-----+
EXIT PROGRAM=NUSTART,USAGE=START
EXIT PROGRAM=NUSTART,USAGE=SHUT
EXIT PROGRAM=FREXIT,USAGE=FORMAT
*+-----+
*I P R O G R A M D E F I N I T I O N S I
*+-----+
DEFAULT PROGRAM COMP=ASSEMB
PROGRAM NUSTART
PROGRAM FREXIT
PROGRAM NUERROR
PROGRAM AUTOTAC
PROGRAM KDCADM,COMP=SPL4
*+-----+
*I SYNCHRONOUS TACS FOR Natural/UTM I
*I THE ERROR EXIT 'NUERROR' MUST BE DEFINED FOR EACH TAC I
*+-----+
DEFAULT TAC TYPE=D,PROGRAM=NUSTART,EXIT=NUERROR,CALL=BOTH
TAC NAT,ADMIN=NO,TIME=0
TAC AUTOCONN
*+-----+
*I BADTACS DEFINITION FOR Natural/UTM I
*I THE ERROR EXIT 'NUERROR' MUST BE DEFINED FOR EACH TAC I
*+-----+
TAC KDCBADTC,CALL=FIRST,PROGRAM=AUTOTAC,EXIT=NUERROR,TYPE=D
*+-----+

```

```

*I ASYNCHRONOUS TACS FOR Natural/UTM I
*I THE ERROR EXIT 'NUERROR' MUST BE DEFINED FOR EACH TAC I
*+-----+
DEFAULT TAC TYPE=A,PROGRAM=NUSTART,EXIT=NUERROR,CALL=FIRST
TAC NATAS
TAC NATSY
*+-----+
*I UTM ADMINISTRATOR TACS I
*+-----+
DEFAULT TAC PROGRAM=KDCADM,ADMIN=Y,TYPE=D,CALL=BOTH
TAC KDCTAC
TAC KDCLOG
TAC KDCSHUT
TAC KDCAPPL
TAC KDCINF
TAC KDCUSER
TAC KDCSEND
TAC KDCDIAG
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCHELP

```

See also the Siemens documentation "UTM Generierung und Administration" (UTM Generation and Administration).

## Generating the Natural UTM Interface

1. The operands of the keyword parameters of macro NATUTM must be set to the correct values as required; the macro NATUTM must then be assembled.

### Example of NATUTM Macro Call:

```

NUSTART NATUTM APPLNAM=NATUTM, --> Note 1 -
NUCNAME=NATvrs, --> Note 2 -
LINK=TACSWTCH --> Note 3 -
PARMOD=24, --> Note 4 -
ROLLACC=UPAM-AS, --> Note 5 -
ROLLTSZ=180, --> Note 6 -
TERMTAB=(SWP,TERMNAME), --> Note 7 -
UMODE=(S,G) --> Note 8

```

Note	
1	The CSECT name of the front-end part of the Natural UTM Interface is specified as NUSTART (default value). The name of the Natural/UTM application is specified as NATUTM.
2	The name of the link-edited reentrant part of the Natural/UTM application is specified as NATvrs; this is also the name of the common memory pool into which the reentrant part will be loaded.
3	A TABLE macro call is to be executed for program TACSWTCH. This means that this program must be linked in the front-end part of the Natural/UTM application.
4	The Natural/UTM application runs in 24-bit addressing mode.
5	The access method to the Natural roll file is specified as UPAM with P1-Eventing for asynchronous writes.
6	The maximum thread size of the Natural roll file is specified as 180 KB.
7	The internal terminal control table is allocated in the Natural swap pool; the logical terminal name will be used for identifying the entries in the terminal control table.
8	The user dialogue with Natural is to take place in "single" mode; that is, one terminal can initiate one Natural session. Messages at restart, logoff and also free-running messages (asynchronous processing) are to be output in German.

The operands of the other keyword parameters of macro NATUTM are not specified since the default values apply.

2. Assemble the macro NURENT (the reentrant part of the Natural UTM Interface). In this example, no changes are required to the keyword parameters. The CSECT name of the assembled macro NURENT is "NURENT".
3. Assemble the macro BS2STUB with the common memory pool definitions specified in macro ADDON.
4. Assemble the Natural parameter module. The sample NTPRM macro call must be adapted to suit the local environment.
5. Assemble the swap pool parameter module (macro NTSWPRM).

## Linking the Non-Reentrant Front-End Part and the Reentrant Part

The front-end part and the reentrant part of the Natural UTM application can be linked using the JCL supplied. This JCL should be checked and modified as required to suit the local environment (library names, etc.) before being used. Special features in the JCL are indicated by REMARK statements.

## Setting Up the Natural Roll File

The size of the Natural swap file must be calculated and the file must be allocated with link name PAMNAT.

## The Start Job for a Natural UTM Application

JCL examples for starting the Natural UTM application are supplied. Before use, the JCL should be checked and modified as required (UTM startup parameters, dataset names, etc.).

## Optimizing Natural UTM Applications

The following points should be considered if the performance of a Natural UTM application is unsatisfactory:

- Can poor performance be localized to one or more particular Natural programs? If so, optimize the program(s) by redesigning. These programs can be identified by using the Natural monitor in library SYSTP.
- Is the swap I/O rate too high? By using the program MENU in library SYSTP you can check how efficiently the Natural swap pool is being used. The statistical information provided about the swap pool also helps to answer the following questions:
  - Is the number of logical swap pools and their slot lengths appropriate? Function SW in the main menu of SYSTP offers various possibilities for controlling the Natural swap pool optimization.
  - Has the Natural swap pool been defined large enough? Increasing the size of the swap pool reduces the swap I/O rate considerably.
- Is the Natural buffer pool too small? Information about the size and occupancy of the Natural buffer pool can be obtained with the Natural utility SYSBPM, which is described in the section Debugging and Monitoring.
- Has the number of UTM tasks been chosen correctly? This is strongly dependent upon the path lengths of the individual transactions and the number of terminals connected.
- Is it possible that particular transactions (so-called "long jobs") are loading the available UTM tasks so heavily that the shorter transactions are suffering from poor throughput as a result? If this is the case, the UTM TACCLASS concept and/or the asynchronous transaction processing facilities should be used.
- Does the Natural Roll File consist of too many extents on one disk drive (physical chained I/O is not possible over extent boundaries), or is the Natural Roll File on a very heavily used disk drive? If possible, allocate the Natural Roll File to one or more lightly-used disk drives, with only one extent on each.

These suggestions should be considered in the light of the total system environment, including such factors as available storage, storage paging rates, disk and channel I/O traffic loads, etc.

## Several Applications with one Common Natural

See also: Natural Shared Nucleus under BS2000/OSD (in the Natural Operations for Mainframes documentation).

To save storage space, it can be desirable for several Natural UTM applications to share a common Natural reentrant part in a common memory pool in the class 6 storage. The following steps must be taken when generating the Natural UTM application:

- The global Natural load pool must be defined with the keyword parameters of module CMPSTART, for example:

```
NAME=NATSHARE, POSI=ABOVE, ADDR=250, PFIX=YES, SIZE=2MB
LIBR=NATvrs.USER.MOD
```

For more information, see CMPSTART Program (in the Natural Operations for Mainframes documentation).

Notes:

- NATSHARE is the name of the linked Natural reentrant part. It is also the name of the common memory pool.
- The operand of parameter PFIX must be YES.
- The operand of parameter ADDR must be defined.
- The operand of parameter LIBR must contain the name of the module library from which the Natural reentrant part is to be loaded.
- The reentrant part of the Natural UTM driver (the assembled module of macro NURENT) must be linked to the front-end part of several applications.
- The operand of keyword parameter NUCNAME must be defined for each assembly of macro NATUTM as the same (in this example: NUCNAME=NATSHARE).
- The definition of the Natural load pool in the ADDON macro for the assembly of macro BS2STUB must be the same for all applications, for example:

```
STUBSHAR BS2STUB PARMOD=31, PROGMOD=ANY
          ADDON NAME=NATSHARE, STAT=GLOBAL
```

For more information, see ADDON Macro in the Natural Operations for Mainframes documentation.

## Lists of Shared and Application-Specific Parameter Modules

If application-specific Natural parameter modules are to be used, they must be linked to the front-end parts of the Natural UTM applications, which means that there is a parameter module in each front-end part. This also applies to the swap pool parameter module.

Only the addresses defined in the CSTATIC list of the parameter module of the front-end part are considered; if any of these addresses cannot be resolved in the front-end part (because they refer to the reentrant part), Natural tries to resolve these addresses with the CSTATIC list in the parameter module of the reentrant part. Thus it is allowed to have unresolved CSTATIC addresses when linking the front-end part, provided they can be resolved by the reentrant part.

As the CSTATIC list of the reentrant part is only used for those addresses which cannot be resolved by the front-end part, *all* CSTATIC entries to be used (whether they are in the front-end part or in the reentrant part) must be defined in the CSTATIC list of the parameter module of the front-end part.

## Entering and Defining Dynamic Natural Parameters

The following possibilities exist for entering and defining the Natural dynamic parameters:

- entering the dynamic parameters together with the UTM TAC when logging on to the application;
- passing the dynamic parameters from another UTM partial program using "MPUT" and "PEND PR(OGRAM)";
- defining the dynamic parameters in the operand of the keyword parameter MSPAR1. They then apply to all users of this application and cannot be changed.

## UTM User Restart

When a Natural session is started, any Natural dynamic parameters defined are saved up to a length which is defined in the operand of keyword parameter SVDYPRM in macro NATUTM. In case of a user restart situation, these saved data are automatically reused when the Natural session is started again. This also applies when the start of the Natural session results from a PEND PR(OGRAM) of another UTM partial program.

See also Global (Restartable) Swap Pool in the Natural Operations for Mainframes documentation.

## Adabas Priority Control

Adabas priority control has no connection with the priority control of BS2000/OSD. Unlike with BS2000/OSD priority control, for Adabas a higher priority value means higher priority. If several requests are in the Adabas command queue at the same time, the request with the highest priority is processed first by Adabas and "1" is added to the priority of the other requests that are in the command queue at this time.

Under certain conditions, it may be useful to assign to the Adabas task a lower BS2000/OSD priority than to the UTM tasks.

The following keyword parameters in macro NATUTM can be used to control Adabas priority control for UTM transactions:

<b>ADAPRI</b>	Activation of Adabas priority control for UTM transactions.
<b>APRISTD</b>	Assignment of standard Adabas priority for all UTM transactions to which no priority is assigned individually.
<b>TCLSn</b>	Assignment of Adabas priority for individual synchronous UTM transactions.
<b>TCLAn</b>	Assignment of Adabas priority for individual asynchronous UTM transactions.

If Adabas priority control is activated for UTM transactions (parameter ADAPRI=YES), it is also in effect for non-Natural programs which access Adabas via the subroutine ADACALL; see the keyword parameter ADACALL in the macro NATUTM.

By defining different Adabas priorities for different transactions with the above parameters, and at the same time using the UTM TACCLASS concept, it is possible to set up a very sophisticated system of priority control. However, when you explicitly assign Adabas priorities to UTM transaction, you should take into consideration the standard priorities Adabas assigns to other processes (for example, TIAM or batch processing).