

Natural

Operations

Version 8.2.7

October 2017

This document applies to Natural Version 8.2.7 and all subsequent releases.

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

Copyright © 1979-2017 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: NATMF-OPERATIONS-827-20180201

Table of Contents

Preface	xi
I Configuring Natural	1
1 Linking Natural Objects to the Natural Nucleus	3
Benefits	4
ULDOBJ Utility	5
Using ULDOBJ to Generate an Object Module	5
Additional Considerations for Linking Subroutines	7
Operating System Dependency of Object Module Generation	7
Example of Linking a Natural Object to the Natural Nucleus	8
2 Natural User Exits	11
NATUEX1 - User Exit for Authorization Control	12
NATSREX2 and NATSREX3 - User Exits for Sort Processing	13
NATUSKnn - User Exit for Computation of Sort Keys	13
NATPM - User Exit for Inverted Output	15
NREXPG - User Exit for NATRJE	16
USR0070P - User Exit for Editor Profiles	16
USR2002P - User Exit for Help Window Text Strings	17
USR2003P - User Exit for Main Menu	17
3 Natural User Access Method for Print and Work Files	19
NATAMUSR Module Description	20
NATAMUSR Module Installation	20
Invoking the Third Party Product	20
4 Natural System Files	21
Natural Scratch-Pad File	22
5 Natural Text Modules and Macros	25
Function and Usage of Text Modules	26
NATTEXT - Natural Keyword Definitions	26
NATTXT2 - Output Text, Keywords and User Termination Messages (Mixed Case)	27
NATTXT2U - Output Text, Keywords and User Termination Messages (Uppercase)	29
NATTXT3 - Text Fragments for Placeholders in Natural Error Messages	30
NTERMSG - Natural Termination Messages and Return Codes	31
6 Natural Configuration Tables	33
NATCONFIG - Natural Configuration Tables	34
General Overview of Macros Used by NATCONFIG	34
NTDVCE - Terminal-Device Specification Table	35
NTMSG - Message Log Table Definitions	36
NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus	36
NTCPAGE - Code Page Definitions	37
Code Page Support	39
Output Devices Supported	39

Example of NTDVCE Macro	40
Translation Tables	41
Upper-/Lower-Case Translation	44
CMULT Entry	45
Output Translation	45
Input Translation	46
Code Translation of DBCS Data	46
NTTZ - Time Zone Definitions	46
7 Natural Storage Management	51
Thread and Non-Thread Environments	52
Buffer Types	52
Fixed Buffers	53
Variable Buffers	53
Customization of Buffer Characteristics	53
II Profile Parameter Usage	55
8 Natural Parameter Hierarchy	57
Natural Parameter Hierarchy Overview	58
General Rules for Parameter Usage	58
Natural Parameter Module	59
Predefined Dynamic Parameter Sets	60
Predefined User Parameter Profiles	60
Dynamic Parameter Entry	60
Natural Security Definitions	61
Session Settings for Profile Parameters	61
Program/Statement Level Settings	61
Development Environment Settings	62
Examples of Parameter Evaluation	62
9 Assignment of Parameter Values	65
Sources for Parameter Value Assignment	66
Static Assignment of Parameter Values	67
Dynamic Assignment of Parameter Values	68
Session Parameters for Runtime Assignment of Parameter Values	70
10 Building a Natural Parameter Module	71
NTPRM Parameter Macro	72
Additional Macros in the Natural Parameter Module	73
Example of Macros in the Natural Parameter Module	75
III z/OS Environment	79
11 Natural under z/OS	81
Natural Subsystem	82
TP Monitor Interfaces	82
Interfaces to Database Management Systems	83
Natural in Batch Mode under z/OS	83
Natural as a Server under z/OS	83
12 Authorized Services Manager under z/OS	85
ASM Overview	86

ASM System Requirements	87
ASM Operation	88
13 Natural Roll Server Functionality	91
Natural Roll-Server Overview	92
Roll Server in a Single z/OS System	93
Roll Server in a z/OS Parallel Sysplex Environment	94
Roll File and LRB	96
14 Natural Roll Server Operation	99
Roll Server System Requirements	100
Formatting the Roll File	101
Starting the Roll Server	104
Roll Server Messages, Condition Codes and Abend Codes	107
Return Codes and Reason Codes of the Roll Server Request	108
Operating the Roll Server	109
Roll Server Performance Tuning	110
Roll Server User Exits	111
IV z/VSE Environment	113
15 z/VSE Environment	115
Natural Subsystem	116
TP Monitor Interfaces	116
Interfaces to Database Management Systems	116
Natural in Batch Mode under z/VSE	117
V BS2000 Environment	119
16 Refresh of Natural Load Pool	121
Prerequisites and Restrictions	122
Procedure	122
Keyword Parameters for the Program PREFRESH	123
17 Optimization of Message Handling	127
Screen Output Handling	128
Restoring the Screen Content	128
18 Terminal Types Supported under BS2000	129
Type 9748	130
975n Series	130
Type 9763M	131
19 Function Keys Supported under BS2000	133
Key Assignment	134
Modes for Key Assignment	134
20 Common Memory Pools	137
Global Common Memory Pools	138
Local Common Memory Pools	142
21 Calling Dynamically Reloadable 3GL Programs in a Natural Application	149
Storage Allocation Rule	150
Thread-Creation Rule	150
Address-Mode Dependencies	150
22 Print File/Work File Server NATPWSV2	153

Setup	154
Operation	155
23 Using Network Attached Storage Files as Natural Print and Work Files	157
24 RPC Server Front-End	159
Setup	160
VI Natural in Batch Mode	165
25 Natural in Batch Mode under z/OS	167
Natural z/OS Batch Interface	168
Driver Parameters for z/OS Batch	168
Data Sets Used by Natural in z/OS Batch Mode	168
26 Natural in Batch Mode under z/VSE	175
Natural z/VSE Batch Interface	176
Driver Parameters for z/VSE Batch	176
Natural Data Sets Used under a z/VSE Batch Mode Session	176
NATVSE Print and Work File Support for z/VSE Library Members	182
NATVSE Print File Support for Direct POWER SPOOL Access	183
NATVSE Dynamic Work File Allocation (DYNALLOC) Support	184
Debugging Facilities for Natural under z/VSE	188
NATVSE Attention Interrupts	191
27 Natural in Batch Mode under BS2000	193
Files and System Files Used by Natural in BS2000 Batch Mode	194
Keyword Parameters	196
BS2000 Job Variables	205
28 Natural in Batch Mode (All Environments)	207
Adabas Data Sets	208
Sort Data Sets	208
Subtasking Session Support for Batch Mode Environments	208
VII Natural Buffer Pools	213
29 Natural Buffer Pool - General	215
Natural Buffer Pool Principle of Operation	216
Buffer-Pool Monitoring and Maintenance	221
Natural Global Buffer Pool	224
30 Natural Global Buffer Pool under z/OS	227
Using a Natural Global Buffer Pool	228
Prerequisites	228
Operating the Natural Global Buffer Pool	229
Sample NATGBPvr Execution Jobs	231
Localization	232
Messages	233
31 Natural Global Buffer Pool under z/VSE	235
Using a Natural Global Buffer Pool	236
Prerequisites	236
Operating the Natural Global Buffer Pool	237
Sample NATGBPvr Execution Jobs	238
Localization	240

Messages	240
32 Common Natural GBP Operating Functions under z/OS and z/VSE	241
Global Buffer Pool Manager Parameter Module	242
Global Buffer Pool Operating Functions	242
Global Buffer Pool Function Parameters	244
Examples of NATBUFFER Specifications	250
33 Natural Global Buffer Pool under BS2000	253
Using a Natural Global Buffer Pool under BS2000	254
Establishing the Global Buffer Pool under BS2000	254
Administering the Global Buffer Pool under BS2000	255
VIII Message Buffer Pool	257
34 Message Buffer Pool	259
Purpose	260
Prerequisites	260
Operating the Message Buffer Pool	260
Sample NATMBPvr Execution Jobs	262
Message Buffer Pool Operating Functions	263
Function Parameters	264
Messages	265
IX Optimize Monitor Buffer Pool	267
35 Optimize Monitor Buffer Pool	269
Purpose	270
Prerequisites	270
Starting the Optimize Monitor Buffer Pool	271
Operator Commands	271
Messages	272
X Natural Swap Pool	273
36 Purpose of a Natural Swap Pool	275
Purpose of a Natural Swap Pool	276
Benefits of Using a Natural Swap Pool	276
Swap Pool Structure	276
37 Natural Swap Pool Operation	279
Users are On their Way to Natural - No Session Start	280
Users are Returning from Natural	280
38 Natural Swap Pool Initialization	281
Swap Pool Initialization Control	282
Swap Pool Initialization Parameters	283
39 Dynamic Swap-Pool Reorganization	285
Requirements for Dynamic Swap-Pool Reorganization	286
Statistics Tables	286
Swap-Pool-Reorganization Plus Table	286
Swap-Pool-Reorganization Minus Table	287
Parameters for Swap-Pool Reorganization	287
Checking for the Necessity of Swap-Pool Reorganization	288
Flow of Dynamic Swap-Pool Reorganization	288

Start of Dynamic Swap-Pool Reorganization	289
40 Defining the Natural Swap Pool	291
Environment-Specific Requirements	292
Keyword Parameters of Macro NTSWPRM	292
41 Natural User Area Size Considerations	301
Using the MAXSIZE Parameter	302
Defining the Size of the Individual Natural Buffers	302
Possible Error Messages	302
Displaying the Aggregate Size of All Buffers	303
Calculating the Maximum Size	303
42 Swap Pool Data Space	305
Using ESA Data Space in Addition	306
ESA Data Space Slot Size Adjustment	306
43 Global Restartable Swap Pool under openUTM	307
Purpose of a Natural Global Swap Pool under openUTM	308
Installing a Natural Global Swap Pool under openUTM	308
Starting a Natural Global Swap Pool under openUTM	309
Displaying Information about the Global Swap Pool	309
44 Terminating the Global Swap Pool under openUTM	311
Termination Using Console Commands	312
Abnormal Termination with Dump	312
Termination by Program	312
XI System Spool Access	315
45 System Spool Access	317
Purpose	318
Prerequisite	318
Using the Write-to-Spool Feature	318
XII Natural 3GL CALLNAT Interface	325
46 Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions	327
Purpose of 3GL CALLNAT Interface	328
Prerequisites	328
Restrictions	330
47 Natural 3GL CALLNAT Interface - Usage, Examples	333
Usage	334
Sample Environments	338
XIII Operating the Software AG Editor	341
48 Editor Work File	343
Editor Work File Structure	344
Editor Work File under z/OS, z/VSE and BS2000	345
Using the Software AG Editor Work File Formatting Utility	346
Formatting during Initialization	346
Maintaining the Editor Work File under z/OS and z/VSE	346
Maintaining the Editor Work File under BS2000	347
Editor Work File under Complete/SMARTS	348
49 Editor Buffer Pool	349

Purpose of the Editor Buffer Pool	350
Obtaining Free Blocks	351
Initializing the Editor Buffer Pool	351
Restarting the Editor Buffer Pool	352
Editor Buffer Pool Parameters	352
Buffer Pool Initialization for Multi-User Environments	352
XIV Natural Net Data Interface NATNETTO	355
50 Natural Net Data Interface NATNETTO	357
Natural Net Data Driver Functional Description	358
General Message Layout	359
Layout of Header	359
Format Buffer Layout	363
Value Buffer Layout	366
Attribute Buffer	366
XV Selectable Units for New Natural Features	369
51 Selectable Units for New Natural Features	371
XVI Natural as a Server	373
52 Natural as a Server under z/OS	375
Functionality	376
Natural Nucleus Installation in a Server Environment	377
Print and Work File Handling with External Data Sets in a Server Environment	377
53 Natural as a Server under z/VSE	379
Functionality	380
Natural Nucleus Installation in a Server Environment	381
Print and Work File Handling with External Data Sets in a Server Environment	381
54 Natural as a Server under CICS	383
Functionality	384
Natural CICS Interface Installation in a Server Environment	384
Restrictions	385
XVII Natural Execution - Miscellaneous Topics	387
55 Natural 31-Bit Mode Support	389
56 Support and Use of Natural and Non-Natural Objects	391
Support for Natural Objects from Previous Natural Versions	392
Back-End Program Calling Conventions	392
LE Subprograms	394
External Sort Programs	397
57 Input/Output Devices	399
Terminal Support	400
Light Pen Support	400
Printer Support	401
58 Double-Byte Character Sets	405
Natural Profile Parameter SOSI	406
Output Format Specification	406

Parameter Definitions for DBCS Support	406
Editor Profile Options	407
Input Data Check	407
Output Data Adjustment	408
Natural Stack Data	408
Application Programming Interfaces for DBCS Handling	408
Alternate Text Module NATTXT2U	409
59 Asynchronous Processing	411
Identifying Asynchronous Natural Sessions	412
Handling Output of an Asynchronous Natural Session	412
Handling Unexpected or Unwanted Input	413
Other Profile Parameter Considerations	413

Preface

This documentation contains information for operating Natural in a mainframe environment under various operating systems.

This documentation is organized under the following headings:

Configuring Natural	Describes how to link Natural objects to the Natural nucleus. Provides information on Natural user exits, Natural user access method for print and work files, Natural system files, Natural text modules, Natural configuration tables, and Natural storage management.
Profile Parameter Usage	Provides an overview of the hierarchical structure of the different levels on which Natural parameters can be set. Explains how values can be assigned to profile parameters statically, dynamically and at runtime. Describes how to build a Natural parameter module.
z/OS Environment	Contains an overview of special considerations that apply when you are running Natural under z/OS online or in batch mode. Describes the functions and the operation of the Authorized Services Manager (ASM). Explains the functions of the Natural Roll Server. Provides information on the Roll Server system requirements, operation, performance tuning and restart capability.
z/VSE Environment	Contains special considerations that apply when you are running Natural under z/VSE online or in batch mode.
BS2000 Environment	Contains special considerations that apply when running Natural under BS2000.
Natural in Batch Mode	Contains considerations that apply when running Natural in batch (Adabas data sets, sort data sets, subtasking session support for batch environments), and specifically when running Natural in batch mode under z/OS, z/VSE and BS2000.
Natural Buffer Pools	Contains information about the various storage management functions that are available to a Natural administrator under z/OS, z/VSE and BS2000.
Message Buffer Pool	Provides information on how to start and operate the message text buffer pool.
Optimize Monitor Buffer Pool	Provides information on how to start and operate the Optimize Monitor Buffer Pool.
Natural Swap Pool	Provides information on the Natural swap pool which is available when you are using CICS or <i>openUTM</i> .
System Spool Access	The Write-to-Spool feature enables Natural users to write reports to the system spool directly. It can be used in any Natural environment (Com-plete, TSO, CICS, IMS TM, batch, etc.) and uses the Entire System Server view WRITE - SPOOL. This output may then be processed by any software which expects output in JES or POWER spool (for example, Entire Output Management).

	This feature applies to z/OS and z/VSE only.
Natural 3GL CALLNAT Interface	Contains information about the Natural 3GL CALLNAT Interface with which Natural enables 3GL programs to invoke and execute Natural subprograms.
Operating the Software AG Editor	Contains information on how to operate the Software AG Editor.
Natural Net Data Interface NATNETTO	Provides information on the Natural Net Data Interface and the net data protocol definition.
Selectable Units for New Natural Features	Provides information on new Natural features supplied as selectable units.
Natural as a Server	Describes the use of Natural as a Server and the Natural Server Monitor.
Natural Execution - Miscellaneous Topics	Provides information on Natural 31-bit mode support, support and use of Natural and non-Natural objects, input/output devices, double-byte character sets and asynchronous processing.

Notation *vrs* or *vr*

When used in this documentation, the notation *vrs* or *vr* represents the relevant product version (see also *Version* in the *Glossary*).

I **Configuring Natural**

This part provides information on Natural configuration.

Linking Natural Objects to the Natural Nucleus

Natural User Exits

Natural User Access Method for Print and Work Files

Natural System Files

Natural Text Modules and Macros

Natural Configuration Tables

Natural Storage Management

See also the following documents in the *Utilities* documentation:

- *SYSCP Utility - Code Page Administration*
- *SYSEXT - Natural Application Programming Interfaces*
- *SYSAPI - APIs of Natural Add-On Products*

1 Linking Natural Objects to the Natural Nucleus

▪ Benefits	4
▪ ULDOBJ Utility	5
▪ Using ULDOBJ to Generate an Object Module	5
▪ Additional Considerations for Linking Subroutines	7
▪ Operating System Dependency of Object Module Generation	7
▪ Example of Linking a Natural Object to the Natural Nucleus	8

The Natural nucleus is a collection of service programs such as memory administration, string handling, operating system interfaces, the compiler and the runtime environment which comprise the kernel of Natural. It is independent of the operating-system and the TP system.

The Natural nucleus consists of the environment-independent and the environment-dependent nucleus. The environment-independent nucleus can be used by several mainframe operating and TP systems. The environment-dependent nucleus contains components that depend on the operating and TP systems. For further information, see *Natural Nucleus Components* in the *Installation for z/OS*, *Installation for BS2000* and *Installation for z/VSE* documentation.

This document describes the advantages of linking Natural objects to the Natural nucleus and provides information on how to proceed.

Benefits

Linking Natural objects to the Natural nucleus provides the following benefits:

■ Better Performance

The objects are executed from the nucleus and not from the Natural buffer pool. This saves space in the buffer pool and also results in fewer database calls. (If Natural cataloged objects are not linked to the Natural nucleus, they are stored in a database file, for example Adabas, and the actual code must be loaded from this file into the buffer pool before it can be executed.)

■ Consistency

As an object which is linked to the Natural nucleus is always executed from the nucleus, there is no effect if the cataloged object from which it was derived is deleted or changed in the Natural system file. Thus, during each TP-monitor session, the status of the object remains unchanged. A new version of an object which is linked to the nucleus can be obtained by unloading it with `ULDOBJ` (see below), relinking the new version to the Natural nucleus and refreshing the Natural module. (Refreshing implies that a new copy of a module is loaded into the TP monitor region.)

■ Global Error Handling

If a cataloged object fetches another program to handle errors (for example, by using the Natural system variable `*ERROR-TA`), and the error-handling program cannot be loaded into the buffer pool, the original error might be missed and any subsequent error may mask the first error and lead to confusion. To prevent this situation, you can link a user-written global error-handling program to the nucleus.

ULDOBJ Utility

You can use the `ULDOBJ` utility to link Natural cataloged objects to the Natural nucleus. With the `ULDOBJ` utility, you generate an object module from a Natural cataloged object and write it to a Natural work file. The generated object module is then processed by the linkage editor and linked to the Natural nucleus.

Under z/OS and z/VSE: When an environment-independent Natural nucleus is used, the generated object module has to be linked to the environment-*in*dependent part of the nucleus.

Using ULDOBJ to Generate an Object Module

➤ To invoke the `ULDOBJ` utility

- 1 Log on to the library `SYSMISC` and issue the command `ULDOBJ`.

```

15:49:39          ***** NATURAL OBJECT MAINTENANCE *****          2012-02-13
User: XYZ          - NATURAL ULDOBJ UTILITY -                          Library: SYSMISC
                                                                Opsys .. z/OS

Specify parameters below ....

Object ..... _____ (Enter '.' to exit)
Library ..... SYSMISC_
OP System ... z/OS_____

```

- 2 Specify and confirm the following parameters:

Object	The name of the cataloged object to be processed. The object can be a program, subprogram, subroutine, help routine or map.
Library	The name of the library containing the cataloged object.
OP System	The name of the operating system for which the object module is to be generated. (Different operating systems have different rules to which the object module must conform.) The name of the operating system must be one of the following: z/OS z/OS systems z/VSE z/VSE systems BS2000 BS2000 systems

For each object processed, the `ULDOBJ` utility displays a report containing the following information:

- the object type (Program, Subprogram, Subroutine, Helproutine, Map, Adapter); see *Objects for Natural Application Management* in the *Programming Guide*.
- the name of the cataloged object processed;
- the programming mode (S = structured mode, R = reporting mode);
- the name of the library containing the cataloged object;
- the name of the operating system for which the object deck was generated;
- the size of the cataloged object and optimized code (if applicable);
- the Natural version of the cataloged object (see *Version* in the *Glossary*);
- statistics about the last cataloging of the object, including user and terminal IDs.

`ULDOBJ` prompts for another object and library after the data from the initial input have been processed. The operating system is not requested, because it does not make sense to generate object modules for more than one operating system for the same Natural work file.

➤ **To terminate the `ULDOBJ` utility**

- After the last cataloged object has been processed, enter a “.” in the first input field (`Object`) and press `ENTER`.

The generated object module conforms to the format of the specified operating system. It is in relocatable format with non-executable code and consists of:

- an external symbol directory (ESD),
- a relocation dictionary (RLD),
- text with the instructions and data corresponding to the program,
- an `END` statement (end-of-module indicator for the load module).

The generated object module is written to a Natural work file, which is used as input to a linkage editor. (Depending on the operating system, it may be better to use `ULDOBJ` in batch mode.)

The generated object module must be processed by the linkage editor of the corresponding operating system before the code is executable as a load module (see the **example** given below). Each load module is valid once it is linked to the Natural nucleus and defined by an `NTSTAT` entry definition in the Natural configuration module `NATCONFIG` (see *Natural Configuration Tables*).

Additional Considerations for Linking Subroutines

Once a cataloged object has been unloaded by the `ULDOBJ` utility and linked to the Natural nucleus, the cataloged object can be deleted from the Natural system file.

However, this is *not* true for an object of type “subroutine”. A subroutine has two names:

- the name specified in the statements `PERFORM` and `DEFINE SUBROUTINE` and
- the name of the object that contains the `DEFINE SUBROUTINE` statement.

Natural internally associates these two names, but this is possible only if the cataloged object still exists on the Natural system file. If the cataloged object were deleted, this association would be lost and the subroutine linked to the nucleus would not be executable.

Operating System Dependency of Object Module Generation

The object module is generated in different ways, according to the operating system. These differences are listed below.

Platform	Requirement
z/OS	<p>A <code>NAME</code> control statement is generated as the last card of the object module. It specifies the replace function. For example:</p> <pre>NAME TEST (R)</pre> <p><code>TEST</code> is the name of the cataloged object.</p>
z/VSE	<p>The object module(s) will be in <code>LIBR</code> format. A <code>CATALOG</code> control statement is generated as the first card and a <code>/*</code> as the last card of the object module. For example:</p> <pre>CATALOG TEST.OBJ REPLACE=YES object module ... /+</pre> <p><code>TEST.OBJ</code> is the name of the cataloged object.</p> <p>When the <code>LIBR</code> utility is executed, assign <code>SYSIPT</code> to the work file written by the <code>ULDOBJ</code> utility (<code>ASSIGN SYSIPT=work-file-1</code>).</p>

Platform	Requirement
BS2000	<p>The object module(s) will be in LMS format. An ADD control statement is generated as the first card and an END statement as the last card of the object module. For example:</p> <pre>ADDR >TEST object module ... END</pre> <p>When the LMS utility is executed, assign SYSDTA to the work file written by the ULDOBJ utility (SYSFILE SYSDTA=<i>work-file-1</i>). The file name generated is <i>Nvr</i>.MOD, where <i>vr</i> represents the relevant product version.</p> <p>If multiple cataloged objects are unloaded during execution of the utility, the object decks are appended to each other.</p>

Example of Linking a Natural Object to the Natural Nucleus

If, for example, the objects LOGPROG and EDITPROG in the library SYSLIB are to be linked to the Natural nucleus, the following steps could be taken:

1. Identify the cataloged objects to be linked.

Object	Library
LOGPROG	SYSLIB
EDITPROG	SYSLIB

2. Set up the batch Natural job stream. Assuming a z/OS environment, include the following cards:

```
//CMWKF01 DD DSN=ULD.NAT.PGMS,UNIT=SYSDA,DISP=(,KEEP),
//          SPACE=(CYL,(3,1),,RLSE),VOL=SER=VVVVVV,
//          DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//CMSYNIN DD *
LOGON SYSMISC
ULDOBJ LOGPROG,SYSLIB,OS
EDITPROG,SYSLIB
.
FIN
/*
```

3. Set up the linkage editor job stream.

```
//JOB CARD JOB (ACCTING),CLASS=A,MSGCLASS=X
//*
/* GENERATE OS LOAD MODULE FROM ULDOBJ UTILITY
/*
//LINK1 EXEC PGM=IEWL,PARM='LIST,LET,XREF,NCAL,RENT,REUS'
//SYSLMOD DD DSN=NATURAL.USER.LOAD,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=X
//SYSLIN DD DSN=NAT.ULD.PGMS,DISP=OLD,UNIT=SYSDA,VOL=SER=VVVVVV
/*
```

This step places the load modules LOGPROG and EDITPROG in the NATURAL.USER.LOAD data set.

With an additional link-edit job, these modules can be linked together as a single load module before being linked to the nucleus in Step 5.

```
//JOB CARD JOB (ACCTING),CLASS=A,MSGCLASS=X
//*
/* OPTIONAL JOB TO LINK CATALOGED OBJECTS TOGETHER
/*
//LINK2 EXEC PGM=IEWL,PARM='LIST,LET,XREF,NCAL,RENT,REUS'
//SYSLMOD DD DSN=NATURAL.USER.LOAD,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=X
//SYSLIN DD *
INCLUDE SYSLMOD(LOGPROG) LOGON NATURAL PGM
INCLUDE SYSLMOD(EDITPROG) EDITOR NATURAL PGM
NAME XXXXXX(R)
/*
```

4. Define the statically linked Natural programs in source module NATCONFIG in the NSTATIC table for linked Natural programs:

```
NTSTAT INPL,TYPE=W
NTSTAT INPLLIB,TYPE=W
NTSTAT AERROR,TYPE=W
NTSTAT LOGPROG <==== your entries
NTSTAT EDITPROG <====
```

TYPE=W means that a “weak” external reference to the specified program is generated rather than a normal one.

5. Review the linkage editor job stream for the Natural nucleus and include the following:

```
/**
/** INCLUDE DDNAME AND DSN OF DATASET WHERE OBJECTS RESIDE
/**
//SYSLMOD DD DSN=NATURAL.USER.LOAD,DISP=SHR
//NATLIB DD DSN=NATURAL.USER.LOAD,DISP=SHR/**
//SYSLIN DD*
...
...           INCLUDE MODULES FOR NUCLEUS
...
INCLUDE NATLIB(nat-parm-module)    NATURAL PARAMETER MODULE
INCLUDE SYSLMOD(LOGPROG)           LOGON NATURAL PGM
INCLUDE SYSLMOD(EDITPROG)         EDITOR NATURAL PGM
...
...           INCLUDE ENTRY AND NAME CARDS
...
/**
```

nat-parm-module represents the name of the **Natural parameter module**.

If the cataloged objects were linked together (as done optionally in Step 3), include this load module instead of the individual load modules in the link of the nucleus.

2 Natural User Exits

▪ NATUEX1 - User Exit for Authorization Control	12
▪ NATSREX2 and NATSREX3 - User Exits for Sort Processing	13
▪ NATUSKnn - User Exit for Computation of Sort Keys	13
▪ NATPM - User Exit for Inverted Output	15
▪ NREXPG - User Exit for NATRJE	16
▪ USR0070P - User Exit for Editor Profiles	16
▪ USR2002P - User Exit for Help Window Text Strings	17
▪ USR2003P - User Exit for Main Menu	17

NATUEX1 - User Exit for Authorization Control

The user exit NATUEX1 is called whenever a user session is activated. It can be used to determine whether or not the user is authorized to use Natural. The security data used to determine this can be retrieved from the security system being used (for example, RACF or ACF2).

NATUEX1 is called using standard calling conventions:

Register	Contents
15	Entry address of NATUEX1
14	Return address of Natural
13	Address of a save area of 18 words
1	Address of a parameter list

The parameter list contains five addresses:

Address	Points to an 8-byte field containing the value which is used to fill the Natural system variable
1	*INIT-USER
2	*ETID
3	*INIT-ID
4	*INIT-PROGRAM
5	*USER (Note that this system variable will be overwritten during a Natural Security logon.)

These five values can be modified by the user exit.

For normal completion, the user exit must return control with Register 15 set to 0. If Register 15 does not contain 0, the Natural session is terminated with the condition code equal to the value in Register 15.

NATUEX1 can be linked to the environment-independent nucleus or to an environment-dependent nucleus. It is also possible to link it to an **alternative Natural parameter module**, or as a separate module if you are running with profile parameter RCA.

An example of the user exit is available as member XNATUEX1 in the Natural source library.

For CICS: See also *NCIUIDEX - User ID Exit Interface* in the *Natural TP Monitor Interfaces* documentation.

NATSREX2 and NATSREX3 - User Exits for Sort Processing

Natural provides two user exits for sort processing: NATSREX2 and NATSREX3.

The two user exits can be used with Natural's own sort program as well as with an external sort program. The exits are activated automatically when they are linked to the nucleus and so their addresses get resolved. Since, under z/OS and z/VSE, many external SORT programs already supply several exit functions, the exits NATSREX2 and NATSREX3 may especially be used either with Natural's internal sort program or with external SORT under BS2000.

NATSREX2 is always called when Natural passes a record to the sort program. NATSREX3 is called when the sort program, upon completion of the sort run, passes a record to Natural. The example delivered shows how you can establish your own collating sequence for a SORT.

When the user exits are activated, the following register conventions must be adhered to:

Register	Contents
15	Entry addresses of NATSREX2 and NATSREX3
14	Return address of Natural
13	Address of the 18-word save area
1	Address of the sort record
3	Length of the sort record

The user exits have to secure the Natural registers and restore them upon returning control to Natural.

As the sort exit module is linked to the module NAT2SORT, programming has to be reentrant. The format and structure of the sort records must not be modified.

NATUSKnn - User Exit for Computation of Sort Keys

Some national languages contain characters which are not sorted in the correct alphabetical order by a sort program or database system. With the system function SORTKEY you can convert such "incorrectly sorted" characters into other characters that are "correctly sorted" alphabetically.

When you use the SORTKEY function in a Natural program, the user exit NATUSK nn will be invoked - nn being the current language code (that is, the current value of the system variable *LANGUAGE).

You can write a NATUSK nn user exit in any programming language that provides a standard CALL interface. The character string specified with SORTKEY will be passed to the user exit. The user exit has to be programmed so that it converts "incorrectly sorted" characters in this string into corres-

ponding “correctly sorted” characters. The converted character string is then used in the Natural program for further processing.

For the conversion, NATUSKnn may use the translation table NTUTAB1 of the configuration module NATCONFIG; this means that NTUTAB1 may have to be adjusted accordingly.

NATUSKnn is called using standard calling conventions:

Register	Contents
15	Entry address of NATUSKnn
14	Return address of Natural
13	Address of a save area of 18 fullwords
1	Address of a parameter list

The parameter list contains the following addresses:

Offset	Address of
+0	The character string passed from Natural.
+4	The length of the character string (fullword).
+8	The character string resulting from the conversion.
+12	The length of the result string (fullword).
+16	The translation table NTUTAB1.

NATUSKnn has to secure all registers, except 14 and 15, and restore them upon returning control to Natural.

For normal completion, the user exit must return control with Register 15 set to Return Code 0. If Register 15 does not contain "0", a corresponding Natural error will be issued.

Sample User Exit Programs

The following sample user exits are provided in source code form:

Program	Function
NATUSK01	Applies to English and converts all English lower-case letters in the character string to upper-case.
NATUSK02	Applies to German and converts the German umlauts ä, ö, ü, and ß into their corresponding replacement characters ae, oe, ue, ss in order to provide a different sort sequence.

NATUSKnn can be linked to the environment-independent nucleus or to an environment-dependent nucleus. It is also possible to link it to an **alternative Natural parameter module**, or as a separate module if you are running with profile parameter RCA=NATUSKnn.

For linkage and loading conventions, see also the CALL statement in the Natural *Statements* documentation.

NATPM - User Exit for Inverted Output

The NATPM module is used to support inverse direction terminals. It contains the user exit routine for field and line conversion which is called by Natural at terminal I/Os if for some fields the print mode (profile parameter PM) has been set to I.

PM=I indicates inverse direction and is used to support languages writing from right to left (for example, bi-directional languages); see also the description of the profile parameter PM.

The module NATPM is delivered as a source module and can be modified if required.

Inversion Logic

Natural provides a user-exit routine which is called for each field where the resulting attribute is PM=I and for each line to be printed via hardcopy, additional report and primary batch output.

This exit is called with three parameters:

- the source field to be inverted,
- the target field to receive the inverted data,
- a length field specifying the length of the source and target fields.

As this user exit routine is available in source code to all users, it might be used as an explicit field exit triggered by the PM=I attribute. The user is then able to check and modify line contents or field contents.

Field User Exit

The user exit in NATPM will be called for every field where the attribute PM=I is set.

This attribute can be set by the Natural programmer, or is automatically set for numeric fields when the global print mode is set to PM=I. It does not matter whether the output is generated for the terminal, for hardcopy, for additional reports or for the primary output in batch.

For printing devices, Natural does not expect automatic inversion from the hardware, but calls NATPM again for the complete line. This feature can be used in countries where the field inversion is not required to establish interface logic with Natural based on a field attribute.

NREXPG - User Exit for NATRJE

NREXPG is a user exit for Natural Remote Job Entry (NATRJE). After the job is complete, each JCL card is passed to the exit before it is submitted to the operating system. The following data are available to the exit:

- the JCL card to be submitted,
- a return code field,
- the name of the Natural program currently being executed,
- the Natural user identification,
- a 240-byte work area.

After each call, the exit passes a return code to NATRJE indicating one of the following events:

Code	Explanation
0	Submission: the card is submitted; the exit may modify the card before submission.
4	Termination: the card is submitted; the exit is disabled for further cards of the current job.
8	Insertion: the card is skipped based on the assumption that it contains only an insert character, for example, the percent sign (%); additional specified cards are submitted.
10	Deletion: the card is not submitted.
12	The current job is flushed.

An example of the user exit, called NREXPG, is available as member XNATRJE in the Natural source library. The exit can be assembled and linked according to the rules of programs specified as CSTATIC. However, a CSTATIC entry for NREXPG is not required.

USR0070P - User Exit for Editor Profiles

The user exit routine USR0070P enables you to modify the parameter settings for the Natural program editor or data area editor in the default profile SYSTEM.

For further information on the editor profile, see *General Information* in the *Editors* documentation.

USR0070P provides a list of all parameters which are to receive a default setting.

With this user exit, you can also determine whether editor profiles are to be stored in the FNAT system file, the FUSER system file or the scratch-pad file.

In addition, USR0070P considers DBCS support and sets the editor profile options `Editing in Lower Case` and `Dynamic Conversion of Lower Case` correspondingly.

An example of this user exit routine is available in the library SYSEXT on the FNAT system file, both in object and source form. Information on how to use it is contained in the text object USR0070T.

USR2002P - User Exit for Help Window Text Strings

The user exit routine USR2002P can be used to customize the text strings for the **Current Natural Message** window that is invoked by pressing the Help key while the cursor is on the message line.

The object USR2002P itself contains the text strings used within the Current Natural Message window, for example, the window title and the descriptive texts, such as, the field names Sh (short message), Tx (long message), Ex (explanation) and Ac (action).

An example of this user exit routine is available in the library SYSEXT on the FNAT system file, both in object and source form. Information on how to use it is contained in the text object USR2002T.

USR2003P - User Exit for Main Menu

The user exit routine USR2003P can be used to customize the following settings for the Natural Main Menu and its subordinate menus:

- position and color of the message line,
- position and color of the PF key lines.

An example of this user exit routine is available in the library SYSEXT on the FNAT system file, both in object and source form. Information on how to use it is contained in the text object USR2003T.

3 Natural User Access Method for Print and Work Files

- NATAMUSR Module Description 20
- NATAMUSR Module Installation 20
- Invoking the Third Party Product 20

This document describes the Natural User Access Method which is an interface for third party vendor products for Natural print and/or work file support.

NATAMUSR Module Description

The NATAMUSR module provides an exit interface (entry point NATAM9EX) for software vendors to handle Natural print and work files, that is, it actually consists of two parts:

- the Natural User Access Method stub NATAMUSR delivered with Natural and
- the Natural User Access Method exit NATAM9EX delivered by a software vendor.

NATAMUSR Module Installation

The NATAMUSR module (with the access method exit) may be installed in one of the following ways:

- linked to the environment-independent nucleus,
- linked to the environment-dependent nucleus,
- linked to an **alternative Natural parameter module** (as loaded via profile parameter PARM),
- linked as a separate module; in this case, the following Natural profile parameters are required:

```
RCA=(NATAM09),RCALIAS=(NATAM09,xxx),
```

where *xxx* is the name of the separate module in the load library.

The environment-independent nucleus and the environment-dependent nucleus are described in *Natural Nucleus Components* for z/OS, z/VSE and BS2000 in the *Natural Installation* documentation.

Invoking the Third Party Product

➤ **To invoke the third party product for Natural print and/or work file processing**

- Specify AM=USER for the relevant files (see also NTPRINT and NETWORK).

For details about the Natural User Access Method exit installation and other information about the third party exit handler, refer to the documentation of the relevant software vendor.

4 Natural System Files

- Natural Scratch-Pad File 22

The table below lists and describes the Natural system files that are usually available in a Natural environment. The availability of the system files and the data contained in the files depends on the Software AG products installed in addition to base Natural.

The settings for the system files are defined with Natural profile parameters of the same names (exception: scratch-pad file). You can follow the hyperlinks in the table below to read details about these parameters in the *Parameter Reference* documentation.

System File	Supplied with	File Contents
FNAT	Base Natural	All objects required for Natural system applications.
FUSER	Base Natural	User-specific objects required for user-defined applications.
FPROF	Base Natural	Parameter profiles specified by the profile parameter PROFILE, provided no database information is supplied as subparameter of PROFILE.
Scratch-pad file	Base Natural	Data that is not stored explicitly as a Natural object in another system file. See also <i>Natural Scratch-Pad File</i> in the <i>Operations</i> documentation.
FDIC	Base Natural	Natural Data Definition Modules (DDMs). If Predict is installed, FDIC also contains data for the Predict dictionary system. If the Natural Development Server is installed, FDIC also contains application data and holds object locking information.
FREG	Base Natural	Registry data that is not stored explicitly in another system file.
FSEC	Natural Security	Control information required for security definitions.
FSPool	Natural Advanced Facilities	Control and spooling information required to output a report on a screen or printer and obtain print statistics.

Natural Scratch-Pad File

The Natural scratch-pad file is used to store recordings and screen captures which cannot be explicitly saved as a Natural object in the Natural FNAT or FUSER system file.

In contrast to FNAT and FUSER, a scratch-pad file is *not* mandatory in a Natural session. However, you must define a scratch-pad file if you are working with read-only access to system files (profile parameter ROSY=ON). Otherwise, the recordings and screen capture cannot be stored and a corresponding error message (NAT0106) is issued instead. The scratch-pad file is excluded from read-only access.

A reasonable estimate about the related storage requirements is hardly possible as the amount of storage used by the Recording utility and the NATPAGE utility (for screen captures) cannot be calculated beforehand. However, the scratch-pad file size required at your site can be estimated with

a better understanding of the types of records that are stored on it. The content of the scratch-pad file is described in the following section:

- [Recordings](#)
- [Screen Captures - NATPAGE](#)
- [File Maintenance](#)

Related Topic:

Defining a Scratch-Pad File in the *Installation for z/OS*, *Installation for BS2000* and *Installation for z/VSE* documentation.

Recordings

The Recording utility is activated using terminal commands as described in the *Utilities* documentation. Recordings are stored like Natural source programs (or other object types). The size of a recording depends on how many screen inputs have been done during a recording session. Recordings are like programs related to a library.

Currently, it is not possible to list recordings on the scratch-pad file by using the Natural LIST system command. SYSMAIN can be used, though, to list and maintain the recordings stored on the scratch-pad file. To store the recordings on the FNAT/FUSER file instead of on the scratch-pad file, set the profile parameter RFILE.

Recordings which are being stored on the system file FNAT or FUSER are affected (interrupted) by transaction backouts (BTs) which are issued in the user's application programs. This is a very common problem encountered by users of the recording facility and it can be avoided by using the scratch-pad file.

Screen Captures - NATPAGE

The screen capturing utility NATPAGE can be used to store screen images (in chronological sequence of their appearance) on the scratch-pad file. NATPAGE can be activated with the terminal command %P. From the moment %P is issued, all screens presented to the end user are stored onto the scratch-pad file (if it has been defined for your session) until the terminal command %0 is entered. The captured screens can be displayed using the terminal command %E.

For each screen image, the current content of the page buffer and the page attribute buffer is stored. This means that the amount of data being stored depends on the settings of the profile parameters PS/LS for the session and, of course, on the number of screen images. The number of possible screens per user session depends on the profile parameter PD (default is 50; valid values are 0-255).

The size of the page buffer can be calculated as:

PS * LS

The size of the page attribute buffer is determined dynamically.

File Maintenance

The scratch-pad file does not need any maintenance, provided it is of sufficient size.

- Recordings on the scratch-pad file can be deleted, copied, moved and listed by using the utility `YSMAIN`.
- Captured screens can be deleted by using the `%E` terminal command.
- Saved screen images, however, cannot be maintained in Natural at all.

Space on the scratch-pad file can be reclaimed by refreshing it with Adabas utilities in times of non-activity without affecting subsequent Natural sessions which are using the scratch-pad file.

5

Natural Text Modules and Macros

- Function and Usage of Text Modules 26
- NATTEXT - Natural Keyword Definitions 26
- NATTXT2 - Output Text, Keywords and User Termination Messages (Mixed Case) 27
- NATTXT2U - Output Text, Keywords and User Termination Messages (Uppercase) 29
- NATTXT3 - Text Fragments for Placeholders in Natural Error Messages 30
- NTERMSG - Natural Termination Messages and Return Codes 31

This document describes the Natural text modules NATTEXT, NATTXT2, NATTXT2U, NATTXT3 and the Natural macro NTERMSG.

Function and Usage of Text Modules

All Natural keywords, alternative keywords and standard output text are contained in the modules NATTEXT and NATTXT2. Natural system commands and alternate system commands are also included as keywords and alternative keywords in these modules. Substitution text fragments for Natural error messages are contained in module NATTXT3. The modules are contained in source form in the Natural source library and in load module form in the Natural load library.

If necessary, you can modify Natural keywords, alternative keywords and text contained in these modules. For example, Natural session termination messages can be changed from English to another language, Natural keywords can be disabled, or synonyms can be added.

If any modifications are made to a NATTEXT, NATTXT2 or NATTXT3 module, each modified module must be assembled, link-edited and included into the executable Natural module; refer to the corresponding Natural installation documentation.

NATTEXT - Natural Keyword Definitions

The NATTEXT module contains the macros NTKEY, NTALT and NTSYN for each keyword and alternative keyword to be recognized by Natural.

Modifying NATTEXT



Caution: It is recommended that you modify the NATTEXT module for very important reasons only, because once modified, it can no longer be properly maintained by Software AG personnel.

The following rules apply:

- A keyword value for a NTKEY or NTALT macro can be changed by replacing the current keyword value with the desired value.
- A keyword or alternative keyword can be disabled by replacing the keyword value with the character "%".
- The position of each NTKEY and NTALT macro within the module is fixed and must not be shifted. Additional NTKEY and NTALT macros must not be inserted.
- Synonyms can be assigned for any keyword or alternative keyword using the NTSYN macro. One or more NTSYN macros can be inserted after a NTKEY or NTALT macro. The NTSYN macro includes

one parameter, which is the value to be used as the synonym. If the synonym contains embedded blanks, the entire value must be enclosed in apostrophes.

Example of Modifying the NATTEXT Module

The following example illustrates how a NATTEXT module is modified. In this example

- the synonym RECHERCHE is to be used for the keyword FIND;
- the synonym LISEZ is to be used for the alternative keyword BROWSE;
- the keywords GET and HISTOGRAM are to be disabled.

NATTEXT **before** modification:

```
STATNAM NTKEY FIND
        NTALT BROWSE
        NTALT GET
        NTALT ACCEPT
        NTALT REJECT
        NTALT HISTOGRAM
```

NATTEXT **after** modification:

```
STATNAM NTKEY FIND
        NTSYN RECHERCHE
        NTALT BROWSE
        NTSYN LISEZ
        NTALT %
        NTALT ACCEPT
        NTALT REJECT
        NTALT %
```

NATTXT2 - Output Text, Keywords and User Termination Messages (Mixed Case)

The NATTXT2 module contains the macros NTKEYT, NTALTT and NTSYNT which define the following:

- [Standard Natural Output Texts](#)
- [Keywords and Alternative Keywords for Natural System Commands and Utilities](#)

- [User-Written Termination Messages](#)

Standard Natural Output Texts

The module NATTXT2 contains the following standard Natural output texts, each of which can also be displayed in another language if the language code is set accordingly (see also below):

- the literal `Page` used in the standard output page header;
- the name of each month as used in the Natural system variable `*DATG` (Gregorian date), date edit masks (L), and the name of each day as used in date edit masks (N);
- the `ENTER INPUT DATA` message and the skeleton error messages for error numbers 1104, 1105 and 1106 (used during online input processing);
- the error message used for system file open failure (which cannot be retrieved from the system file); an error number of the form `NAT8xxx` (where `xxx` is the decimal Adabas response code) is added to this error message by Natural;
- the constants `More`, `Top` and `Bottom` used in windows for position information to be displayed in text form;
- the table to define reports and report handling for reports greater than 33.

Any values contained in NATTXT2 can be modified by replacing the current text with the desired text. If a month-name synonym exceeds nine characters, only the first nine positions are used by the system variable `*DATG`.

NTSYNT macro statements can be added as described for module NATTEXT. However, with NATTXT2, a second parameter can be specified. This parameter is optional and represents the language indicator to be used for the synonym. When you specify the language indicator, Natural produces message output resulting from the use of this synonym in the corresponding language. In addition, if error message texts have been stored in the Natural system file using a language indicator other than 1 (which is the default and stands for `English`), error messages are returned in the corresponding language. For information on which language code stands for which language, refer to the profile parameter `ULANG`.

Keywords and Alternative Keywords for Natural System Commands and Utilities

The module NATTXT2 contains `NTKEYT` and `NTALTT` macros for each keyword and alternative keyword to be recognized by Natural for the following Natural system commands and utilities, parameters of commands and their values when applicable. Each of these can also be used in another language if the language code is set accordingly (see also below):

- all Natural system commands in general;
- for the `GLOBALS` system command, the parameters and their values when applicable;
- for the `COMPOPT` system command, the parameters and their values when applicable;

- public system commands (these system commands are permanently valid and cannot be disallowed, neither by means of Natural Security nor by the Natural profile parameter NC;
- Natural utilities

The NTKEYT and NTALTT macro statements can be used similar to the NTKEY and NTALT macro statements as described for module NATTEXT.

The NTSYNT macro statements can be used as described under *Standard Natural Output Texts*.

User-Written Termination Messages

User-written termination messages can be added with the macro NTERMSG for all return codes (1 - 255) which can be issued with a TERMINATE statement and which normally lead to the Natural termination message NAT9987.

You specify the termination message text with the first parameter, and the corresponding return code with the second parameter.

Example:

```
NTERMSG 'USR0077 THIS IS A SAMPLE USER MESSAGE FOR RETURN CODE 77',77
```

A TERMINATE 77 statement in a Natural application will result in the following termination message: USR0077 THIS IS A SAMPLE USER MESSAGE FOR RETURN CODE 77.

NATTXT2U - Output Text, Keywords and User Termination Messages (Uppercase)

The NATTXT2U module contains the same items as the NATTXT2 module. The difference is that certain keywords for the English language are contained in mixed case in NATTXT2 whereas they are in all uppercase in NATTXT2U. This affects the keywords MORE, TOP, BOTTOM, PAGE, and all month and weekday names.

NATTXT2U should be linked to the Natural nucleus instead of NATTXT2 in environments where lower-case code points H'81' to H'A9' are used to display national characters, for example, if code page 930 with half-width Katakana characters is used.

NATTXT3 - Text Fragments for Placeholders in Natural Error Messages

The NATTXT3 module contains the macros to define the text fragments which will be used to substitute the `:n:` place holder in Natural error messages.

Each text fragment can be defined in various languages. For information on which language code stands for which language, refer to the `ULANG` parameter.

The text fragments will be generated in EBCDIC and Unicode notation.



Note: To assemble the NATTXT3 module, a high level assembler must be used which supports the macro function `UPPER` and the definition of Unicode characters (`DC CU'unicode text'`).

Example:

The text for Natural error NAT0082 (when trying to execute a non existing program) looks as follows:

```
Invalid command, or :1: :2: does not exist in library.
```

Trying to execute the object `NOTEXIST` leads to following result:

```
NAT0082 Invalid command, or Program NOTEXIST does not exist in library.
```

`:2:` was replaced by the object name (`NOTEXIST`).

`:1:` was replaced by the text fragment `Program`.

The text fragment was declared in module NATTXT3 as follows:

```
*=====
*          PROGRAM          0002
*=====
*          MSGSDEF  &LC_PGM
*          SPACE
*-----
*          MSGSLAN  01,Program          1  ENGLISH
*          MSGSLAN  02,Programm         2  GERMAN
*          MSGSLAN  03,programme        3  FRENCH
*          MSGSLAN  04,programa         4  SPANISH
*          SPACE
*-----
*          MSGSGEN
```

Text fragment values for additional languages may be entered by adding further `MSGSLAN` macros.

NTERMSG - Natural Termination Messages and Return Codes

Natural has a number of standard session termination messages (NAT99...) that are delivered in macro NTERMSG and can be modified there (for example, to translate them into another language). The overall length of ID and text can be up to 72 characters. After the macro NTERMSG has been modified, the Natural parameter module has to be re-assembled and linked.

Apart from the message ID and text, each standard termination message also includes one of the following Natural system return codes, which are also defined within macro NTERMSG:

Code	Explanation
0	Normal termination.
4	Error occurred during execution/compilation (batch mode only).
8	Termination due to severe runtime error.
12	Session initialization failure.
16	Abnormal termination due toabend or severe environment failure..

With the profile parameter TS set to ON, the termination messages are translated to upper case using the upper case translation table NTUTAB1 as supplied in the NATCONFIG module before they are displayed.

In addition to TS=ON, further parameters to provide for translation of messages into upper case are provided by several Natural components. For further information, see *Other Parameters to Provide Upper Case Translation* in the TS profile parameter description.

6 Natural Configuration Tables

▪ NATCONFIG - Natural Configuration Tables	34
▪ General Overview of Macros Used by NATCONFIG	34
▪ NTDVCE - Terminal-Device Specification Table	35
▪ NTMSG - Message Log Table Definitions	36
▪ NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus	36
▪ NTCPAGE - Code Page Definitions	37
▪ Code Page Support	39
▪ Output Devices Supported	39
▪ Example of NTDVCE Macro	40
▪ Translation Tables	41
▪ Upper-/Lower-Case Translation	44
▪ CMULT Entry	45
▪ Output Translation	45
▪ Input Translation	46
▪ Code Translation of DBCS Data	46
▪ NTTZ - Time Zone Definitions	46

This document provides general information on the Natural configuration tables which are contained in the NATCONFIG module.

See also:

- *Input/Output Devices Supported*

NATCONFIG - Natural Configuration Tables

The NATCONFIG module contains the Natural configuration tables.

 **Caution:** In general, the default specifications in NATCONFIG need not and should not be modified. In particular, *do not modify* without prior consultation of Software AG support any of the tables marked with an asterisk (*) in the list below.

For most of the tables, there are corresponding macros in the **Natural parameter module** as well as dynamic profile parameters. If you need to modify a NATCONFIG table, use the corresponding parameter-module macro, or dynamic profile parameter, to overwrite the table. (If you made the modifications in the NATCONFIG tables themselves, you would have to modify and reassemble NATCONFIG again with subsequent Natural releases.)

The NATCONFIG module uses macros for the definition of the following Natural default configuration tables.

In addition, it uses the following tables:

- The default attention identifier table. It defines the physical terminal keys to Natural (*).
- Various other tables (*).

General Overview of Macros Used by NATCONFIG

The following table provides a general overview of the macros used by the NATCONFIG module for the definition of the Natural default configuration tables:

Macro	Purpose
NTDVCE *	Table of terminal types. Used to specify the terminal driver to be used, see description below, for details. Important: Do not modify an existing NTDVCE macro, rather create a new one.
NTMSG	Message log table. Natural messages which shall be written to the job message log or to the operator console.

Macro	Purpose
NTSTAT	Definition of Natural objects linked to the Natural nucleus.
NTCPAGE	Code page definitions.
NTTAB	Primary output translation table.
NTTAB1 NTTAB2	Secondary output/input translation tables.
NTUTAB1 NTUTAB2	Tables for translation between lower case and uppercase. These tables have to be modified, for example, for the German character set.
NTTABA1 NTTABA2	Tables for translation of EBCDIC characters to ASCII characters and vice versa. These tables are used by the Object Handler.
NTTABL	SYS* translation table. Translates output from programs contained in Natural SYS. . . libraries.
NTLANG *	Language translation table. Contains a list of all available language codes defined to Natural.
NTSCTAB	Scanner character type table. Determines which characters are lower-case alphabetical, uppercase alphabetical, numeric and special characters (applies to dynamic profile parameters, MASK and SCAN options).
NTTZ	Time zone definitions. The NTTZ macro enables specifications concerning time zones and automatic switching to and from summertime.
NTBUFID	The parameters MIN and MAX of this macro can be used to change the buffer size limits for variable buffers, see Customization of Buffer Characteristics . Important: The default values of the other parameters in this macro should not be modified, because the results may be unpredictable.

* Do not modify without prior consultation of Software AG support any of the tables marked with an asterisk (*) in this list.

For further details, see [Translation Tables](#).

NTDVCE - Terminal-Device Specification Table

For each terminal type supported by Natural, a terminal converter routine is provided. The corresponding terminal drivers are responsible for the actual terminal I/Os. They build the physical data stream from the screen buffer and the screen attribute buffer and place it in the terminal I/O buffer.

In addition, a telex driver is provided for Connect in order to provide faster telex, telefax and teletext communication from and to the Topcall messaging server. This driver supports the Topcall full-page protocol.

With the NTDVCE macro, it is possible to add new terminal drivers to Natural to specify modifications of the terminal-specific input/output or lower-to-upper case translation tables. Other information which can be specified is the frame character, the position of the message line, whether screen

optimization is to be on or off, as well as various flags in the IOCB. In addition, the terminal specification can be routed to an existing driver by using other translate tables or can hook into a driver routine.

The `NTDVCE` macro is invoked by either the terminal command `%T=` from the Natural command line or the `SET CONTROL 'T=...` statement from within a Natural program. At the start of a Natural session, the translation tables `NTTAB`, `NTTAB1`, `NTTAB2`, `NTUTAB1` and `NTUTAB2` are copied from the `NATCONFIG` module into the user area where they are modified by `NTDVCE`.

Note that the translation tables can be modified by the same macros dynamically or within the [Natural parameter module](#).

NTMSG - Message Log Table Definitions

The macro `NTMSG` is used to define Natural messages which shall be written to the operator console or to the job message log (if available). A defined message will be written in addition, that is, the usual Natural processing remains unchanged. To find the log message definition table, locate label `NATMSGT` in `NATCONFIG`. There you can add your `NTMSG` definitions on a one message per line basis.

NTMSG Macro Syntax

The syntax of the `NTMSG` macro is as follows:

```
NTMSG NATnnnn,logid
```

NTMSG Macro Parameters

Parameter	Description
<code>NATnnnn</code>	<code>nnnn</code> is the Natural message number (mandatory).
<code>logid</code>	Indicates the log destination, that is, the operator console or job message log or both. Possible values: <code>WTO</code> , <code>WTL</code> or <code>WTO+WTL</code>

NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus

Any object to be linked to the Natural nucleus must be specified with an `NTSTAT` macro. When searching for an object, Natural always scans this list first, regardless of the library specified. For information on how to link Natural objects to the Natural nucleus, see the `ULD OBJ` utility in [Linking Natural Objects to the Natural Nucleus](#).

NTSTAT Macro Syntax

The syntax of the NTSTAT macro is as follows:

```
NTSTAT object-name[,TYPE=W]
```

NTSTAT Macro Parameters

Parameter	Description
<i>object-name</i>	Specifies the name of the object linked to the Natural nucleus.
TYPE=W	Means that the entry point of the linked object is defined as a “weak external” to the Natural nucleus. This avoids a linkage editor error message in case of the object is not linked to the Natural nucleus.

NTCPAGE - Code Page Definitions

All code pages to be used during a Natural session must be predefined in the source module NATCONFIG. For each code page to be defined, a specific macro NTCPAGE must be entered. During session initialization, the code page specified by the profile parameters CP, CPOBJIN, CPSYNIN, CPPRINT and the CP keyword subparameter of profile parameter PRINT or parameter macro NTPRINT are verified. If this code page is not defined in NATCONFIG, an error message is issued.

NTCPAGE Macro Syntax

The syntax of the NTCPAGE macro is as follows:

```
NTCPAGE IANA=value, *
        CCSID=value, *
        CCSN=value, *
        ALIAS=value, *
        PHC=value, *
        MULTI=value, *
        ECS=value
```

The parameters CCSID and CCSN are platform-specific (IBM/SNI) and mutually exclusive.

NTCPAGE Macro Parameters

Parameter	Description	
IANA	This parameter is required under all operating systems. It specifies the standard name of the code page. Maximum length: 64 characters.	
CCSID	This parameter is required under z/OS, z/VSE. It specifies the coded character set identification; that is, a numeric value with up to 5 digits. Examples: 1141 German EBCDIC code page 62243 Hebrew/Latin (ISO 8859) code page	
CCSN	This parameter is required under BS2000. It specifies the coded character set name; that is, an alphanumeric string of up to 8 characters. Examples: EDF041 Latin code page for Western Europe EDF045 Latin/Cyrillic code page	
ALIAS	This parameter is optional. It specifies the code page alias name. Maximum length: 32 characters.	
PHC	This parameter is optional. It specifies the place holder character. Length: 2 bytes hexadecimal.	
ECS	This parameter is optional. It specifies the key number of the code page in Entire Conversion Service (ADA ECS), which is used by Adabas.	
MULTI	This parameter is optional. It specifies whether the code page is a single-byte code page or a multi-byte or ASCII code page. Possible values:	
	ON	The code page is a single-byte code page; for example, IBM01140. The code page can be used as a Natural session code page. The session code page is defined by the Natural profile parameter CP.
	OFF	The code page is a multi-byte code page or ASCII code page. It cannot be used as a Natural session code page. Any attempt to use this code page results in initialization message NAT7019. This is the default setting.
VALID	The code page is a multi-byte code page, but can be used as a Natural session code page. For example, IBM-939 is a Japanese EBCDIC code page that contains DBCS characters.	

Examples:

NTCPAGE IANA=IBM819,	*
CCSID=819,	*
ALIAS='ISO-8859-1',	*
PHC=003F	

NTCPAGE IANA='IBM-939',	*
CCSID=939,	*
ECS=3035,	*
ALIAS='ibm-939_P120-1999',	*
PHC=3013,	*
MULTI=VALID	

See also *Configuration and Administration of the Unicode/Code Page Environment*.

Code Page Support

By using the [NTDVCE](#) macro, different code pages can be defined and associated with a specific terminal type and name. If Natural is then started with `PM=C`, all terminal I/O is translated on input and retranslated on output. Thus, as long as the code pages are compatible, a common data representation can still be maintained.

See also *SYSCP Utility - Code Page Administration* in the *Utilities* documentation.

Output Devices Supported

Attribute control variables and formats define attributes to generate a certain representation on the output device. Natural offers a wide range of possible attributes to allow the end user the best use in designing maps and reports on the terminal.

Unfortunately not all terminals support all features available with Natural. These features are mostly ignored on such devices or are simulated via other techniques. Basically there are two data stream definitions in an IBM environment called standard data stream and extended data stream and a multitude of data stream definitions in an SNI environment.

The following output devices are supported:

- [Sequential Output Devices for Batch, Additional Reports](#)
- [Line-Oriented Online Terminals](#)

- [Block-Mode-Oriented Online Terminals](#)

Sequential Output Devices for Batch, Additional Reports

The output data contain standard ASA control characters controlling the line advance and page-eject facility of the given printer. This printer can be either the central printer in the computer center supported by the online or batch spooling system or the SCS printer used as online terminal printers.

The following devices can be used to print reports generated in this form:

Device	Type
Impact printer	Standard central printer hardware
Laser printer	High-speed printer, terminal printer
Daisy printer	Terminal printer
Inkjet	Terminal printer

Line-Oriented Online Terminals

Terminal Make	Description
TTY	Data sent to TTY devices are generated using the standard formfeed, linefeed, etc. characters.

Block-Mode-Oriented Online Terminals

Terminal Make	Description
IBM	All models and sizes which support standard data stream and/or extended data stream.
SNI	All 9750 and compatible monochrome devices and all 9763 and compatible color devices.
Wang	All models.
PC	All models and sizes which support standard data stream and/or extended data stream.

Example of NTDVCE Macro

For information on how the NTDVCE macro is specified and for descriptions of the individual parameters, refer to the [NTDVCE](#) macro itself.

Example of NTDVCE macro:

```

NTDVCE TYP=EBS2, *
      NAME=BS2CHAR, *
      ENTRY=VC3270, *
      WXTRN=OFF, RTAL=5, *
      FLAG1=CM3270, *
      TCIO=(X'C0',X'FB',X'6A',X'4F',X'D0',X'FD', *
            X'4A',X'BB',X'E0',X'BC',X'5A',X'BD',X'A1',X'FF', *
            X'4F',X'5A')

```

This sample macro converts internal SNI code pages to external IBM code pages. This enables you to develop applications on IBM terminals, which internally work with SNI code pages to, for example, avoid data collision when migrating from IBM to SNI.

Translation Tables

All data printed, displayed or written by Natural programs are translated by Natural. This guarantees that no illegal control characters can cause terminal I/O errors or display garbage information on the terminal.

Another feature is the translation to and from character sets different from the Latin definition, especially Arabic, Cyrillic, Greek and Hebrew characters.

This section describes all features and functions concerning field translations when data are written to external devices such as CRT (screen terminals) or online and batch spooling systems.

The statements `INPUT`, `DISPLAY`, `PRINT` and `WRITE` write data to or read data from external devices such as CRT, TTY or sequential files. All these statements use parameters such as constants, variables, edit masks, attribute control variables and formats to control the output image and the input representation. Constants and variables are generated by using their respective values in the output image. The representation of these values is then controlled by the attribute control variables, formats, edit masks and translation tables.

Natural uses several translation tables and also provides the use of alternative translation tables, all included in `NATCONFIG`.

The following tables are provided:

Macro	Table
NATSCTU	<p>Required scanner table for Unicode characters. It maps the properties of Unicode characters of the Unicode Specification (as supported by the delivered ICU version) to be used by the Natural nucleus.</p> <p>Important: This table must never be changed.</p>

Macro	Table
NATCPTAB	<p>Optional <i>single-byte</i> code page conversion accelerator tables.</p> <p>If the table is present, conversion from one code page to another code page will be faster since it is performed via this table rather than by calling ICU functions.</p> <p>The following code pages are supported by the delivered NATCPTAB:</p> <p>IBM01140 IBM01141 IBM01145 IBM01146 IBM01147 ASCII</p> <p>It is possible to add new entries by using the NTCPCNV macro. For each conversion direction, an entry is needed that contains the IANA name of the source code page, the IANA name of the target code page and optionally a blank character, a substitution character and a place holder character, followed by a complete list of character mappings.</p>
NTSCTAB	<p>The table which defines the properties of characters</p> <ul style="list-style-type: none"> ■ used in mask definitions for the MASK option, ■ recognized as delimiters in the EXAMINE and SEPARATE statements. <p>This table can be used to define upper-case attributes, lower-case attributes, special characters, hexadecimal characters and numeric characters.</p> <p>To modify this table, use the macro NTSCTAB in the Natural parameter module or the corresponding dynamic profile parameter SCTAB.</p> <p>If the CP profile parameter is set to a value other than OFF, the modification is ignored and the table is adjusted according to the code page used for the Natural session. See also <i>Translation Tables</i> in the <i>Unicode and Code Page Support</i> documentation.</p>
NTTAB	<p>The standard (primary) output translation table used for screen or printer output.</p> <p>Basically this table is used to translate all characters below X'40', that is from the space character to the question mark (X'00' is not translated). This guarantees that all terminal-control characters are translated before output and no control escape sequences can influence the screen output. Special characters (X'FE' and X'FF') which could influence the screen output are translated to the question mark (?).</p> <p>If nothing else is specified, all Natural output data are translated with NTTAB.</p> <p>To modify this table, use the macro NTTAB in the Natural parameter module or the corresponding dynamic profile parameter TAB.</p> <p>The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=<i>code - page</i>), and the table is adjusted by ICU according to the code page used at session start.</p>

Macro	Table
	<p>Furthermore, all characters below X '40' are translated to the question mark (?) as described above. A character is excluded from this translation if either of the following conditions is true:</p> <ul style="list-style-type: none"> ■ The character is explicitly translated to the same character. ■ The character is one of the logical shift-out/shift-in characters specified with the <code>SOSI</code> profile parameter (see the <i>Parameter Reference</i> documentation), and the specified code page is not an MBCS code page.
NTTAB1	<p>The alternative (secondary) output translation table for the secondary character set used when the Natural parameter <code>PM</code> is set to <code>C</code>.</p> <p>The important aspect is the translation of all possible terminal-control characters. If <code>PM=C</code> is specified, all Natural output data are translated with <code>NTTAB1</code>. A possible application of <code>NTTAB1</code> is to avoid the translation of escape sequences for printer control.</p> <p>To modify this table, use the macro <code>NTTAB1</code> in the Natural parameter module or the corresponding dynamic profile parameter <code>TAB1</code>.</p> <p>The modification is ignored if a code page is specified using profile parameter <code>CP</code> (<code>CP=ON</code>, <code>CP=AUTO</code> or <code>CP=code-page</code>), and the table is not used.</p>
NTTAB2	<p>The secondary input translation table used when the Natural parameter <code>PM</code> is set to <code>"C"</code>. If <code>PM=C</code> is specified, all Natural input data are translated with <code>NTTAB2</code>. Conversion between different languages or code pages can be performed with this table together with <code>NTTAB1</code>.</p> <p>To modify this table, use the macro <code>NTTAB2</code> in the Natural parameter module or the corresponding dynamic profile parameter <code>TAB2</code>.</p> <p>The modification is ignored if a code page is specified using profile parameter <code>CP</code> (<code>CP=ON</code>, <code>CP=AUTO</code> or <code>CP=code-page</code>), and the table is not used.</p>
NTTABS	<p>This table defines all valid characters that can be used in Natural variable names; it is used for the Natural syntax processor.</p> <p>It also defines all valid characters that can be used in the first position of a Natural variable name.</p> <p>In addition, it defines whether the variable is a global variable, a non-database variable or a source-code variable.</p> <p>If a code page is specified using profile parameter <code>CP</code> (<code>CP=ON</code>, <code>CP=AUTO</code> or <code>CP=code-page</code>), the table is adjusted by ICU according to the code page used at session start.</p>
NTUTAB1	<p>The sample user-specific translation table for input translation from lower to upper case.</p> <p>In addition, this table performs the translation specified with the statement <code>EXAMINE TRANSLATE INTO UPPER CASE</code>.</p> <p>To modify this table, use the macro <code>NTUTAB1</code> in the Natural parameter module or the corresponding dynamic profile parameter <code>UTAB1</code>.</p> <p>The modification is ignored if a code page is specified using profile parameter <code>CP</code> (<code>CP=ON</code>, <code>CP=AUTO</code> or <code>CP=code-page</code>), and the table is not used.</p>

Macro	Table
NTUTAB2	<p>The sample user-specific translation table which performs the translation specified with the statement EXAMINE TRANSLATE INTO LOWER CASE.</p> <p>To modify this table, you can use the macro NTUTAB2 in the Natural parameter module or the corresponding profile parameter UTAB2.</p> <p>The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=<i>code-page</i>), and the table is not used.</p>
NLANG	<p>The language-code table, which defines which language number is assigned to which language code in the system variable *LANGUAGE.</p>
NTTABL	<p>The SYS* output translation table, which is controlled by the Natural profile parameter TS. With TS=ON, this table is used to translate output produced by programs located in Natural SYS* libraries (except modifiable fields) from Latin lower case to upper case.</p> <p>This table allows the use of all upper- and lower-case characters in Latin oriented countries, but still allows the use of these applications in countries where the lower-case characters have been replaced with a native alphabet.</p> <p>To modify this table, use the macro NTTABL in the Natural parameter module or the corresponding dynamic profile parameter TABL.</p> <p>If Natural is running with an MBCS code page (for example, CP=' IBM-939 '), the table is not used, but translation is performed via ICU according to the current locale settings.</p>
WRDFCUC1 WRDFCUC2 WRDFCSP2	<p>The DBCS translation tables used to translate double-byte characters into Latin characters and vice versa.</p> <p>Important: These tables have to be activated explicitly, for example, for Far East countries.</p>

Upper-/Lower-Case Translation

For modifiable and input fields, upper- and lower-case translation can be specified. In general, lower-case translation means that data are taken as they come in; no translation is performed. This even makes it possible in batch mode, for instance, to read in hexadecimal data without translation.

There are several ways of specifying upper-/lower-case translation:

LC=OFF	<p>Lower-case translation is switched off, which means that global upper-case translation is in effect.</p> <p>This profile parameter can be specified in the Natural parameter module or as dynamic parameter. (Note that the session parameter LC has a completely different function.)</p>
%U	<p>Upper-case translation is globally on.</p> <p>On the field level, the attribute AD=T or AD=W can be specified. These attributes only take effect when the global upper-case translation is deactivated (LC=ON, %L). Then it is possible to control the translation on a field level from within a Natural program.</p>

EXAMINE TRANSLATE	Upper-/lower-case translation can also be performed with the EXAMINE TRANSLATE statement. By default, EXAMINE TRANSLATE translates to upper case by using the translation table NTUTAB1, and to lower case by using the translation table NTUTAB2.
----------------------	---

CMULT Entry

It is no longer recommended to use the CMULT entry; use the EXAMINE TRANSLATE statement instead (see above).

Output Translation

All fields, after having been formatted by possible edit masks, AL or NL parameter values, filling characters, etc. are translated using a translation table. This ensures that no data can be sent to the front-end printing device with embedded control information which is not explicitly generated by Natural. This means that fields can be sent to a display device even if they contain hexadecimal information which is identical to internal attributes. These attributes are translated before an output operation and so Natural guarantees the screen layout as defined by the output statement.

There are several translation tables available. If nothing explicit is defined, the primary translate table NTTAB is used.

If PM=C is specified, the secondary translation table NTTAB1 is used. For modifiable fields, PM=C also means that the incoming data are translated again; that is, translated for output and retranslated for input.

With this translation table logic it is possible, for example, to convert Arabic numerals to Latin numerals. Arabic numerals have a different hexadecimal representation from the normal Latin numerals on the terminal hardware. So on output, the Latin numerals can be translated into the Arabic equivalent and on input, the Arabic numerals can be retranslated into Latin.

Special considerations have to be made for the Natural system applications which use Latin lower-case and upper-case characters. Especially on terminals supporting Arabic, Greek, Cyrillic, etc., the hardware can be switched to not display lower-case Latin characters, but rather the native characters.

Unfortunately, Latin lower-case characters are crabbed when displayed in, for instance, Cyrillic. So Natural can be used with the parameter TS=ON (translate system output). TS=ON translates "SYS*" libraries (not including library SYSTEM) and all Natural system commands by using a third translation table called NTTABL. By default, this translation table performs upper-case translation for all lower-case Latin characters. Of course, only output data are treated this way. So this allows data entry in the native character set even in Natural editors or system applications.

However, if Natural utilities are used to display data typed in the native character set, this results in an upper-case translation even for data in, for example, Cyrillic representation. The result would again be unreadable. So all Natural system utilities can use the format `PM=C` for fields containing data entered in the native character set. In this case, neither the `NTTABL` translation table nor the secondary translation table `NTTAB1` is used. The data are simply translated by the primary translation table `NTTAB`.

For further information, see the profile parameters `PM`, and `TS` in the *Parameter Reference* documentation.

Input Translation

The translation table `NTUTAB1` is available to control translation from lower to upper case. This might cause problems in countries where special characters are used which are not set up with the simple logic that just one bit controls the status of this letter. This especially concerns German umlauts or Danish special characters. In such cases, translation can only be achieved by customizing the `NTUTAB1` table, where for each character the corresponding lower-/upper-case character can be specified.

If upper-case translation (`%U`) and `PM=C` is specified, first upper-case translation (using `NTUTAB1`) and then the secondary input translation (using `NTTAB2`) is performed.

Code Translation of DBCS Data

So that double-byte character set (DBCS) data can be processed the user application programming interface `USR4213N` is provided to translate double-byte characters into Latin characters, see [Double-Byte Character Sets \(DBCS\)](#).

NTTZ - Time Zone Definitions

The `NTTZ` macro is used to specify a time zone and an automatic switch to and from summertime.



Note: Time definitions are determined by the system administrator, and the user can reference these definitions by using the Natural profile parameter `TD=zonename`. With this parameter, users from different countries and time zones are able to select their own local time.

The `NTTZ` macro can be used on a minimal basis to define a time difference for a time zone. In addition, an automatic switch to and from summertime can be specified, either as a fixed date or in a more flexible definition like “first Sunday in April”. The automatic switch to and from summertime is processed during a running Natural session, without requiring any user interactions. Pre-

defined samples of NTTZ macro definitions are available in the Software AG-delivered `NATCONFIG` module.

Reference point for automatic switching to and from summertime is the current machine time, which is UTC (GMT) time. Depending on the time period the current machine time is in, the current local time is determined. The support of automatic switching to and from summertime is currently for years in the range from 2002 to 2041.

The following topics are covered below:

- [NTTZ Macro Considerations and Restrictions](#)
- [NTTZ Macro Syntax](#)
- [NTTZ Macro Parameters](#)
- [Example of NTTZ Macro](#)

NTTZ Macro Considerations and Restrictions

The following considerations and restrictions apply:

1. Time Format

The basic time format is:

```
+hh:mm:ss
```

or:

```
-hh:mm:ss
```

ranging from 00:00:00 through 23:59:59; abbreviations are also allowed, for example: `hh:mm` or simply `hh`. The plus sign (+) is assumed by default, the minus sign (-) may be necessary with the parameters `TDON` or `TDOFF`.

2. UTC versus Local Time

In order to have a unique point of reference for the time switch, the NTTZ macro parameters `SWTON` and `SWTOFF` are given in UTC time, whereas the weekday names and day numbers in the NTTZ macro parameters `DSTON` and `DSTOFF` are specified in local time.

3. Concurrent Use of Natural Profile Parameters DD, YD, and TD

The Natural profile parameters `DD` and `YD` do not have any effect on the automatic switching to and from summertime, since the switch is done on the basis of the current machine time.

It is recommended to avoid the concurrent use of `DD` or `YD` and profile parameter `TD=zonename`.

4. Concurrent Use of Natural Profile Parameter TD and User Exit CMCOTIME

Concurrent use of profile parameter `TD=zonenumber` and user exit `CMCOTIME` (override machine time) is not recommended, because a change of machine time (TOD clock) may cause unpredictable results for automatic switching invoked with `TD=zonenumber`.

NTTZ Macro Syntax

The syntax of the NTTZ macro is as follows:

```

NTTZ ZONE=value;          *
      TDON=value,         *
      TDOFF=value,        *
      SWTON=value,        *
      SWTOFF=value,      *
      DSTON=value,       *
      DSTOFF=value
    
```

NTTZ Macro Parameters

[ZONE](#) | [TDON](#) | [TDOFF](#) | [SWTON](#) | [SWTOFF](#) | [DSTON](#) | [DSTOFF](#)

ZONE - Time Zone Name

`ZONE=value` specifies the Software AG or user-defined time zone name which can be referenced with the `TD` parameter. The first occurrence of a name will be selected.

Value	Explanation
32 characters.	The maximum length of a time zone name is 32 characters to allow for descriptive user defined zone names, for example, the name of the capital city of a country.

TDON - Difference of Local Daylight Saving Time to UTC Time

`TDON=value` denotes the difference of local daylight saving time (summertime) to UTC time (formerly GMT).

Value	Explanation
<code>+hh:mm:ss</code> or <code>-hh:mm:ss</code>	See Time Format .



Notes:

1. If only the parameter `TDON` is defined, the user gets display of local time as his zone time, without automatic switching to and from summertime.
2. The parameter `TDON` corresponds to the parameter [SWTON](#).

TDOFF - Difference of Local Zone Time to UTC Time

$TDOFF=value$ denotes the difference of local zone time to UTC time (formerly GMT).

Value	Explanation
$+hh:mm:ss$ or $-hh:mm:ss$	See also Time Format .



Note: This parameter corresponds to the parameter [SWTOFF](#).

SWTON - Time when Daylight Saving Time Starts

$SWTON=value$ denotes the UTC point of time when daylight saving time (summertime) is switched on.

Value	Explanation
$hh:mm:ss$	See also Time Format .

SWTOFF - Time when Daylight Saving Time Ends

$SWTOFF=value$ denotes the UTC point of time when daylight saving time (summertime) is switched off.

Value	Explanation
$hh:mm:ss$	See also Time Format .

DSTON - Date when Daylight Saving Time Starts

$DSTON=(value1, value2, value3, value4, day-number)$ denotes the day when daylight saving time (summertime) is switched on.

Value	Possible Settings
$value1$	FIRST, SECOND, THIRD, FOURTH or LAST.
$value2$	MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY or SUNDAY.
$value3$	AFTER, BEFORE or IN.
$value4$	JANUARY ... DECEMBER.
$day-number$	A valid day number for the respective month. The default value is 1.

**Notes:**

1. The keyword `LAST` requires the keyword `BEFORE` or `IN`.
2. No *day number* must be specified if the keyword `IN` is specified.

DSTOFF - Date when Daylight Saving Time Ends

DSTOFF=(*value1*, *value2*, *value3*, *value4*, *day-number*) denotes the day when daylight saving time (summertime) is switched off.

Value	Possible Settings
<i>value1</i>	FIRST, SECOND, THIRD, FOURTH or LAST.
<i>value2</i>	MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY or SUNDAY.
<i>value3</i>	AFTER, BEFORE or IN.
<i>value4</i>	JANUARY ... DECEMBER.
<i>day-number</i>	A valid day number for the respective month. The default value is 1.



Notes:

1. The keyword LAST requires the keyword BEFORE or IN.
2. No *day number* must be specified if the keyword IN is specified.

Example of NTTZ Macro

For daylight saving time switching in Western Europe:

```

NTTZ ZONE=MEZ, *
      TDON=2, *
      TDOFF=+01:00:00, *
      SWTON=01:00:00, *
      SWTOFF=01:00:00, *
      DSTON=( LAST, SUNDAY, IN, MARCH), *
      DSTOFF=( LAST, SUNDAY, IN, OCTOBER)
    
```

Additional examples of different time zones (North and South America, Asia, etc.) can be found in the Software AG-delivered [NATCONFIG](#) module.

7 Natural Storage Management

- Thread and Non-Thread Environments 52
- Buffer Types 52
- Fixed Buffers 53
- Variable Buffers 53
- Customization of Buffer Characteristics 53

This document describes how Natural allocates and uses main storage. A chunk of storage requested by a Natural nucleus component is called a “buffer”.

Thread and Non-Thread Environments

There are two different types of storage environments:

- Thread storage environment (typical for multi-user environments, for example, CICS)
- Non-thread storage environment (typical for single-user environments, for example, batch)

In a thread environment, a big piece of storage called “thread” is pre-allocated for a session. The thread size must be predefined by the system administrator. During a session each buffer allocation request (GETMAIN) is satisfied within its thread by Natural itself. Free space due to release buffer requests (FREEMAIN) can be reused.

Upon certain events (terminal I/Os and long waits), the thread storage may be compressed and rolled out (or swapped out) to external storage (swap pool or roll file). The released thread can be reused by other Natural sessions. When a suspended session is to be resumed, it is rolled in from external storage into a free thread again.

The place on the swap pool or roll file where the compressed thread storage is stored, is called a “slot”. The slot size has a fixed length and is defined by the system administrator. It must be large enough to contain the largest compressed thread storage. In the worst case, it may be equal to the thread size.

In a non-thread environment, all storage requests are directly passed to the operating (sub)system. No roll-out/roll-in is performed, that is, the buffers for a session are kept until session termination, unless they were explicitly released before.

Buffer Types

There are three different types of buffers:

- fixed buffers
- variable buffers
- physical buffers

Fixed buffers and *variable buffers* have a 32-byte prefix with a common layout for all environments. The buffer prefix starts with the buffer name followed by 5 buffer length fields (total, used low-end, max. used, used high-end, max. used high-end). The used length fields are maintained by the buffer-owning components and are used for thread compression. Each buffer has a unique ID number (1-255) and can exist only once. Some buffers are allocated during session initialization,

others are allocated when required. The system command `BUS` can be used to show information about all fixed and variable buffers currently allocated. The characteristics of the buffers are defined in the source module `NATCONFIG`, which can be customized in exceptional cases (see [Customization of Buffer Characteristics](#) below). The size of some buffers can be specified by a profile parameter. For a complete list of such buffers, see the profile parameter `DS`.

Physical buffers are allocated outside the thread. They do not have a buffer prefix and they are not unique. They are used in exceptional cases and temporarily only. Physical buffers are automatically released at the next terminal I/O. It is possible to define work pools for physical buffers by profile parameter `WPSIZE`.

Fixed Buffers

In a thread environment, fixed buffers are allocated from the low end of the thread only. In contrast to variable buffers, fixed buffers cannot be moved relatively to the thread and their size cannot be increased or decreased.

Variable Buffers

In a thread environment, variable buffers are allocated from the high end of the thread. If there is no more space in the thread, variable buffers are allocated temporarily outside of the thread. Upon thread compression, all buffer parts used are compressed into the thread. If they do not fit into the thread, the session is terminated abnormally. This may happen especially when large dynamic variables are used.

After thread decompression, the variable buffers may have been moved to a different place inside or outside of the thread. Variable buffers can be increased or decreased in size on request by the owning component. Some variable buffers are defined to be reduced or released automatically during thread compression.

The total amount of storage allocated outside the thread can be limited by profile parameter `OVSIZE`.

Customization of Buffer Characteristics

All buffers are defined in the source module `NATCONFIG` by `NTBUFID` macro definitions.



Caution: Please, do not change any buffer characteristics except the `MIN`, `MAX` and `CMPR` parameter settings explained below, because the results may be unpredictable.

It is possible to change the buffer size limits by the parameters `MIN` and `MAX` of the macro `NTBUFID`. This makes sense for variable buffers (`TYPE=VAR`) only. Limits for all buffers are defined either by default (0 - 2097151 KB) or by the limits of the corresponding profile parameters. For further information, see the profile parameter `DS`. The limits of the buffer size profile parameters in the Natural parameter module are not affected by the `MIN` and `MAX` parameters of `NTBUFID`, but the limits for the dynamic profile buffer size parameters are overwritten by `MIN` and `MAX`.

Setting the `MAX` parameter to a value in KB means that the size of this buffer cannot exceed this maximum during session execution. This may cause runtime errors if more buffer storage is requested for the desired buffer.

Setting the `MIN` parameter to a value in KB means that the size of this buffer cannot be less than this value during session execution. For example, in the case of the 3GL `CALLNAT` interface (`NAT3GCAN`), the setting of a buffer minimum value makes sense for the following buffers, because the sizes of these buffers may not be increased on a lower Natural program level called by a 3GL program.

Buffer	Purpose
DATSIZE	Data areas
GLBT00L	Utility GDA
GLBUSER	User GDA
GLBSYS	System GDA
AIVDAT	AIV area
CONTEXT	Context variables

The parameter `CMPR` of the macro `NTBUFID` defines the compression optimization algorithm for the buffer. It corresponds to the profile parameter `CMPR` which defines the default. For more information about the possible parameter values, see *CMPR – General Default Compression Optimization Algorithm* in the *Parameter Reference* documentation.

Example of a buffer characteristics definition:

```
DATSIZE NTBUFID ID=GETMDATA,TYPE=VAR+INI,CMPR=OPT2,MAX=512
```

For further information on profile parameters affecting the buffer sizes, see *Storage Management* in the *Parameter Reference* documentation.

II Profile Parameter Usage

This part describes the fundamentals and rules that apply to the use of Natural profile parameters in a mainframe environment.

Natural Parameter Hierarchy	Overview of the hierarchical structure of the different levels on which Natural parameters can be set. Examples are provided to illustrate the various scenarios.
Assignment of Parameter Values	Assigning values to profile parameters statically, dynamically and at runtime.
Building a Natural Parameter Module	Building a Natural parameter module from NTPRM and other parameter macros.

Related Topics:

- For details of the individual profile parameters, see the *Parameter Reference* documentation.
- For an overview of the profile parameters grouped by category, see *Profile Parameters Grouped by Category* in the *Parameter Reference* documentation.

8 Natural Parameter Hierarchy

- Natural Parameter Hierarchy Overview 58
- General Rules for Parameter Usage 58
- Natural Parameter Module 59
- Predefined Dynamic Parameter Sets 60
- Predefined User Parameter Profiles 60
- Dynamic Parameter Entry 60
- Natural Security Definitions 61
- Session Settings for Profile Parameters 61
- Program/Statement Level Settings 61
- Development Environment Settings 62
- Examples of Parameter Evaluation 62

Natural Parameter Hierarchy Overview

Natural profile parameters affect the appearance and the response of a Natural user's working environment. These parameters are set at different hierarchically organized levels as illustrated in the table below (priority from high to low).

Level	Short Description/References to Detailed Descriptions
During Session	<ul style="list-style-type: none"> ■ Development Environment Settings ■ Program/Statement Level Settings ■ Session Parameter Settings ■ Natural Security Definitions
Dynamic during Session Start	<ul style="list-style-type: none"> ■ Dynamic Parameter Entry ■ Predefined User Parameter Profiles ■ Predefined Dynamic Parameter Sets ■ Alternative Natural Parameter Module
Static	<ul style="list-style-type: none"> ■ Natural Parameter Module

The hierarchically organized levels are discussed in the referenced sections, starting from the lowest and ending with the highest priority.

General Rules for Parameter Usage

The following general rules apply:

- A parameter value set on a higher level overwrites the value defined on a lower level (exceptions: PROFILE, SYS, DYNPARM and some other parameters that work by adding values).
- Dynamic parameters during session start have sequence priority, that is, they are evaluated from left to right.

Example:

```
ESIZE=20,DATSIZE=60,ESIZE=100
```

The resulting value is `ESIZE=100`.

- Not all of the parameters available at a lower level can be defined on a higher level, too.

Natural Parameter Module

A Natural parameter module contains a set of profile parameters required to configure your Natural environment.

A Natural parameter module is built from the `NTPRM macro` and **additional macros** during the installation process as described in *Building a Natural Parameter Module*.

You can have more than one Natural parameter module depending on your personal preferences, for example, one module for Natural batch and one for Natural online sessions.

The Natural parameter module constitutes the bottom level of the Natural parameter hierarchy.

In addition to the Natural parameter module, you may require an additional parameter module for a Natural add-on product to be used in your environment, for example, the Natural CICS Interface.

- [Alternative Natural Parameter Module](#)

Alternative Natural Parameter Module

In addition to a Natural parameter module which is statically linked to the nucleus, you can define alternative Natural parameter modules which are stored in a TP or operating-system library. They can be used to overwrite the parameter definitions of the static Natural parameter module for a Natural session by specifying the profile parameter `PARM` as described in the *Parameter Reference* documentation. Exception: `CSTATIC` parameter definitions are not overwritten.

 **Important:** `PARM` should appear as the first parameter in a dynamic parameter string, because otherwise the alternative Natural parameter module overwrites all parameter settings previously entered in the dynamic parameter string.

Usage Restrictions

You can restrict the use of an alternative Natural parameter module to a certain user or to several users by using the `NTUSER` macro.

In this macro, define the IDs of those users who are authorized to use that parameter module. Only these users will be allowed to specify the name of that parameter module with the profile parameter `PARM`.

Predefined Dynamic Parameter Sets

The assembler macro `NTSYS` can be used to predefine parameter sets which are named in a Natural parameter module. These sets can be addressed under their names when Natural is invoked, provided that the corresponding parameter module is active.

When invoked, the predefined parameter sets react in the same way as dynamically entered parameters in that position.

See also the profile parameter `SYS`.

Predefined User Parameter Profiles

You can use the Natural utility `SYS Parm` to create individual profiles which are stored in a system file. Each profile is given a unique character name. You can set values for any dynamic Natural parameters in such a profile.

The profiles created with the utility `SYS Parm` are activated by using the parameter `PROFILE` when Natural is invoked.

You can use the profile parameter `USER` to restrict the use of a profile to a certain user or to several users.

When invoked, the predefined parameter profiles behave in the same way as dynamically entered parameters in that position.

Dynamic Parameter Entry

Almost all of the parameters can be dynamically overwritten when Natural is started. Dynamic parameters are evaluated strictly sequential.

This general overwrite facility can, however, be limited generally or for certain parameters through the use of the profile parameter `DYN PARM` (only dynamically, for instance in a profile).

You can use the macro `NTDYN P` in the Natural parameter module to make analog settings. This, however, will prohibit the use of the profile parameter `DYN PARM`.

You can use the data set `CMPRMIN` to define dynamic parameters in batch mode under z/OS, BS2000 and z/VSE, or in batch-like systems such as TSO, TIAM or BMP environments under IMS TM.

The advantage of this method is that you need not modify the JCL when you wish to change Natural settings. In addition, it overcomes the length limitation of the parameter string (for example, 100 characters under z/OS).

Natural Security Definitions

Apart from protecting the libraries, files and commands, Natural Security enables the setting of certain session-relevant profile parameters. The definitions apply to the current library of the user.

The users can also define settings for their private or default libraries.

The current security settings (session parameters) can be displayed using the Natural system command `PROFILE`.

The Natural Security parameter definitions are evaluated after the regular profile parameters, that is, they can overwrite them.

Session Settings for Profile Parameters

The Natural system command `GLOBALS` or the Natural statement `SET GLOBALS` can be used to display and to set (modify) certain session-relevant profile parameters within and for the duration of a Natural session.

These definitions apply to the command mode and to all programs that are executed during the current session.

See also *[Session Parameters for Runtime Assignment of Parameter Values](#)* or `SET GLOBALS`.

Program/Statement Level Settings

The Natural statement `FORMAT` can be used in a program to set parameter values which are valid for that specific program.

In addition, it is possible to set certain parameters at statement level by a terminal command.

Development Environment Settings

You can use the Natural Main Menu option *Development Environment Settings* to invoke a submenu which enables selection of the tools that are available for monitoring and setting up the Natural development environment.

Examples of Parameter Evaluation

The examples below are based on the following parameter settings:

Parameter	Parameter Module	Alternative Parameter Module	User Profile
		ALTPARM	MYPROF
DATSIZE	40	50	60
DSIZE	6	2 (default)	Not specified
ESIZE	28 (default) NTSYS A: 40 NTSYS B: 50	NTSYS A: 60	80

The following examples show the results for various dynamic parameter strings.

Example 1: No Dynamic Parameters

Resulting Values	Origin
DATSIZE 40	Parameter module
DSIZE 6	Parameter module
ESIZE 28	Parameter module
Others: Default	Parameter module

Example 2: PARM=ALTPARM

Resulting Values	Origin
DATSIZE 50	ALTPARM
Others: Default	ALTPARM

Example 3: SYS=A

Resulting Values	Origin
DATSIZE 40	Parameter module
DSIZE 6	Parameter module
ESIZE 40	NTSYS macro in the parameter module

Example 4: PARM=ALTPARM, SYS=A

Resulting Values	Origin
DATSIZE 50	ALTPARM
DSIZE 2	ALTPARM
ESIZE 60	NTSYS macro in ALTPARM

Example 5: PARM=ALTPARM, SYS=B

Resulting Values	Origin
Error	ALTPARM does not have an NTSYS B specification

Example 6: SYS=A, PROFILE=MYPROF

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Parameter module
ESIZE 80	MYPROF

Example 7: SYS=A, PROFILE=MYPROF, ESIZE=100

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Parameter module
ESIZE 100	Dynamic parameter

Example 8: PROFILE=MYPROF , SYS=A

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Parameter module
ESIZE 40	NTSYS macro in the parameter module

Example 9: DSIZE=8 , SYS=A , PROFILE=MYPROF , PARM=ALTPARM

Resulting Values	Origin
DATSIZE 50	ALTPARM
Others Default	ALTPARM

9 Assignment of Parameter Values

- Sources for Parameter Value Assignment 66
- Static Assignment of Parameter Values 67
- Dynamic Assignment of Parameter Values 68
- Session Parameters for Runtime Assignment of Parameter Values 70

This document provides information on how values are assigned to profile parameters statically, dynamically and at runtime.

For details of the individual profile parameters, refer to the *Parameter Reference* documentation.

Sources for Parameter Value Assignment

The values for profile parameters are taken from three sources:

1. **Static assignments**

Profile parameters specified in the parameter macro `NTPRM` and **additional parameter macros** contained in the **Natural parameter module**.

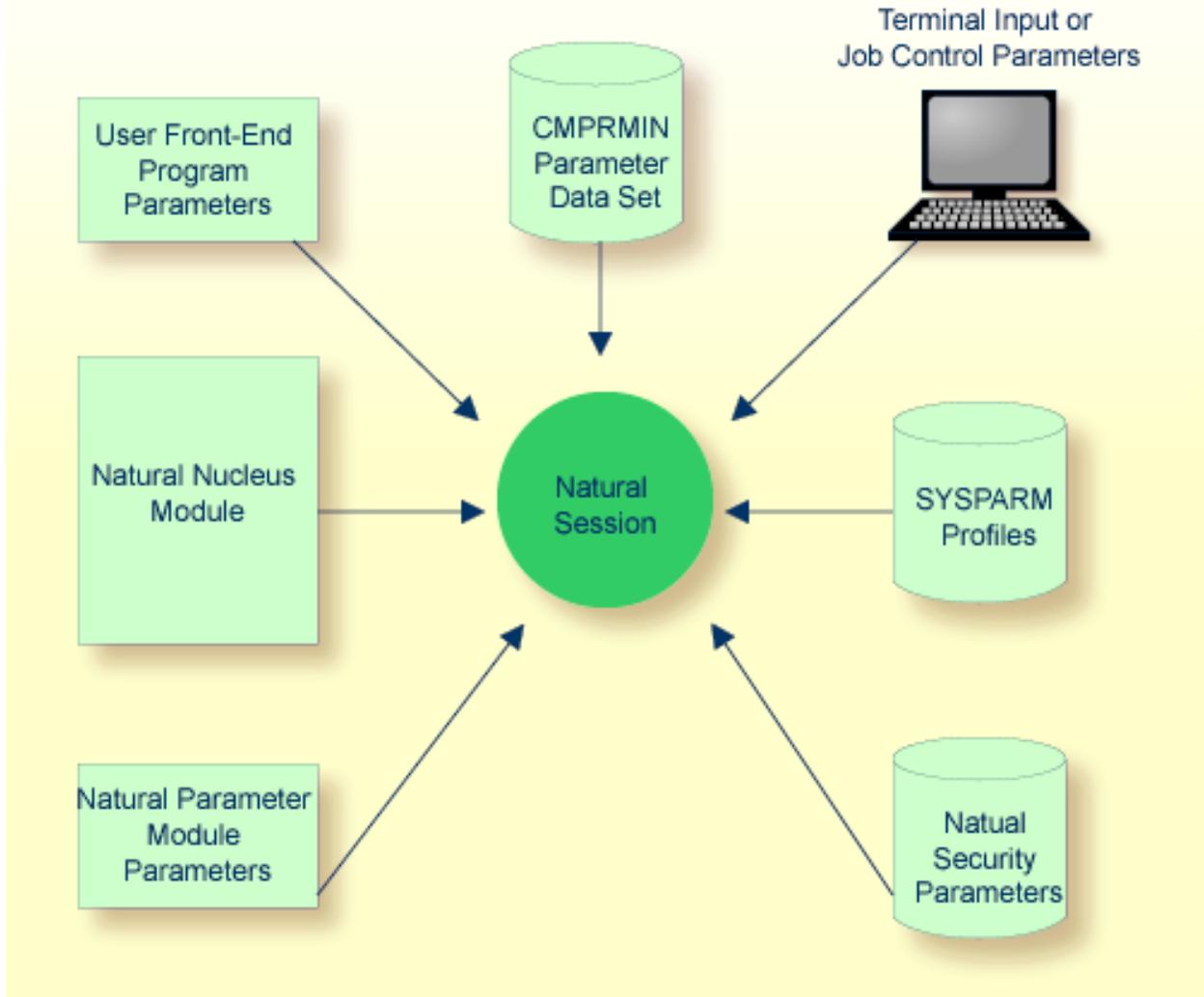
2. **Dynamic assignments**

Parameters specified for the Natural session execution. These parameters override the static assignments and are valid for the current Natural session. Dynamic parameters can be passed by a front-end program, the parameter data set (`CMPRMIN`), session-initialization JCL, terminal input or Natural Security. In addition, it is possible to overwrite certain parameters by Natural program statements.

3. **Session parameters**

Parameters specified with the system command `GLOBALS` (or a `SET GLOBALS` statement) within the current Natural session. The parameters override static and dynamic assignments.

Illustration of the Natural Parameter Assignment:



Static Assignment of Parameter Values

The Natural parameter module is used for the static assignment of profile parameters for all Natural environments.

In the Natural parameter module, you use the macro `NTPRM`, and several other macros, to specify the parameters.

All parameter settings (except the parameter `CSTATIC`) made in the Natural parameter module can be overwritten dynamically at the start of a Natural session.

For some profile parameters a corresponding macro is used for static assignment in the Natural parameter module. Consequently, the syntax of the static and dynamic specifications differs slightly, taking the following general form:

Static:	<code>macro-name keyword1=value,keyword2=value1,value2,...</code>
Dynamic:	<code>parameter-name=(keyword1=value,keyword2=value1,value2,...)</code>

Example:

- Macro in the Natural parameter module: `NTSORT WRKSIZE=500,EXT=ON`
- Equivalent dynamic profile parameters: `SORT=(WRKSIZE=500,EXT=ON)`

If there is a parameter macro for a profile parameter other than `NTPRM`, this macro is indicated in the individual parameter description.

See also the section [Building a Natural Parameter Module](#).

Dynamic Assignment of Parameter Values

You can specify profile parameters dynamically at the start of a Natural session to override - for the duration of a single Natural session - individual profile parameter settings of the [Natural parameter module](#).

Example:

```
NUCNAME='NATNUC/#5',IM=D,INTENS=1,DU=OFF,FUSER=(10,32),PROGRAM=' ',
WORK=((1),AM=STD,DEST=WORK1,OPEN=INIT),PS=60,LS=120
```

All profile parameters can be specified dynamically - except `CSTATIC` which can be specified statically in the Natural parameter module only:

The dynamic parameter assignments are separated by (one or more) commas or blanks. If the value for a dynamic parameter contains non-alphanumeric or special characters, the value must be specified enclosed in apostrophes. Which characters are special characters is defined in the character table macro `NTSCTAB` of `NATCONFIG`; see [Natural Configuration Tables](#).

The use of dynamic parameters can be enabled/disabled by the macro `NTDYNP` or the corresponding dynamic profile parameter `DYNPARM`.

For a more comfortable specification of sets of dynamic parameters, you can use the profile parameter `PROFILE` or `SYS`. In addition, it is possible to set a number of dynamic parameters in Natural Security.

It is possible to insert comment strings within dynamic parameters. A comment starts with “/*” and ends with “*/”. If the comment string end delimiter is missing, an error message is issued during session initialization.

Example:

```
PARM=MYPARMS /* my comment */ ADANAME=ADALNKR,PROFILE=MYPROF
```

The dynamic parameter settings are passed to Natural when the session is started. The method used for passing the parameter values to Natural varies depending on the environment.

Example for z/OS in Batch Mode:

- The values are specified by the PARM keyword in the EXEC job control statement that initiates Natural.
- In addition, dynamic parameters can be specified in the data set `CMPRMIN`.
- Moreover, it is possible to write a front-end program which passes control to Natural with dynamic parameters for the session according to z/OS standards.

Specifying Dynamic Parameters under z/VSE

The dynamic parameters can either be passed directly with a PARM specification in the JCL EXEC statement:

```
// EXEC NATBATCH,PARM='dynamic parameters...',SIZE=...
```

Or you can specify PARM='SYSRDR' to cause Natural to read the dynamic parameters from SYSRDR:

```
// EXEC NATBATCH,SIZE=...,PARM='SYSRDR'
dynamic parameters
...
/* END OF DYNAMIC PARAMETERS
```

If the PARM keyword is not specified in the JCL EXEC statement, the SYSPARM parameter of the JCL OPTION statement is checked for compatibility reasons:

```
...
// OPTION SYSPARM='SYSRDR'
// EXEC NATBATCH,SIZE=...
dynamic parameters
...
/* END OF DYNAMIC PARAMETERS
```

Session Parameters for Runtime Assignment of Parameter Values

To some profile parameters a value can be assigned within a Natural session at runtime, using a corresponding session parameter. The session parameter value will override the profile parameter value.

If a corresponding session parameter exists for a profile parameter, this is indicated in the description of the profile parameter.

Session parameters are specified with the system command `GLOBALS`. Session parameters are described in the *Parameter Reference* documentation. Further details on system commands can be found in the *Command Reference* documentation.

Example:

```
GLOBALS SA=ON IM=D
```

Session parameters can also be specified with the `SET GLOBALS` statement in a program.

Some profile parameters can also be overridden within a Natural session by a terminal command. If a corresponding terminal command exists for a profile parameter, this is indicated in the description of the profile parameter. Terminal commands are described in the *Terminal Commands* documentation.

Example:

```
SET CONTROL 'T=3279'
```

The value of the profile parameter `TTYTYPE` is overwritten.

10 Building a Natural Parameter Module

- NTPRM Parameter Macro 72
- Additional Macros in the Natural Parameter Module 73
- Example of Macros in the Natural Parameter Module 75

You build (generate) a Natural parameter module during the installation of Natural by running the appropriate installation jobs provided by System Maintenance Aid (SMA). These jobs are described in the relevant installation steps in the *Installation for z/OS*, *Installation for z/VSE* and *Installation for BS2000* documentation.

A Natural parameter module is built from the NTPRM parameter macro and **additional parameter macros** if required. You can change the default parameter settings provided by SMA and adapt the installation jobs according to your needs.

» To build a Natural parameter module using SMA jobs

- 1 Adapt the profile parameters in the default NTPRM parameter macro according to your needs by using the NTPRM macro syntax (see the *Parameter Reference* documentation).
- 2 If required, add **additional parameter macros**. They must follow NTPRM in any order.
- 3 Assemble the Natural parameter module and link it to the environment-dependent nucleus (see the *Installation* documentation for z/OS, z/VSE and BS2000).
- 4 Link the Natural parameter module to the environment-independent nucleus (see the *Installation* documentation for z/OS, z/VSE and BS2000) if either of the following is true:
 - Your NTPRM macro contains CSTATIC entries.
 - Your Natural parameter module contains an NTCSTAT macro.

The CSTATIC profile parameter and the NTCSTAT parameter macro are described in the *Parameter Reference* documentation.

NTPRM Parameter Macro

The NTPRM parameter macro is mandatory; it must be specified in the Natural parameter module. NTPRM contains the main profile parameter settings required to configure Natural. All profile parameters for which no parameter macro is indicated in the individual parameter description in the *Parameter Reference* documentation are defined in the NTPRM macro.

See also *NTPRM Macro Syntax* (*Parameter Reference* documentation) and [Example of Macros in the Natural Parameter Module](#).

Additional Macros in the Natural Parameter Module

In addition to the `NTPRM` macro, in the Natural parameter module you can specify the parameter macros listed in the following table. You can specify the macros in any order.

The use of one or more additional parameter macros depends on your individual system requirement and the products installed in your Natural environment.

The name of an additional parameter macro and its syntax are contained in the individual description of the corresponding profile parameter in the *Parameter Reference* documentation.

See also [Example of Macros in the Natural Parameter Module](#).

Naming Conventions and Overview of Macros

Each additional parameter macro usually has a corresponding dynamic profile parameter.

The name of an additional parameter macro starts with `NT` followed by the name of its corresponding profile parameter. For example: the `NTBPI` parameter macro corresponds to the profile parameter `BPI`. Any exceptions to this rule are indicated in the following table.

The following is an overview of available macros:

Macro	Purpose
NTALIAS	Defines external alias names for the modules linked to the Natural nucleus. Corresponding dynamic profile parameter: <code>RCALIAS</code> .
NTBPI	Assigns buffer pools to Natural sessions.
NTCCTAB	Defines printer-control sequences.
NTCFICU	Enables Unicode and code page support.
NTCMPO	Specifies compilation options.
NTCOMP	Specifies configuration settings for the Natural Complete/SMARTS Interface (Natural under Complete/SMARTS).
NTCSTAT	Defines the modules to be linked to the Natural nucleus. Corresponding dynamic profile parameter: <code>CSTATIC</code> .
NTDB	Defines database types and options for databases.
NTDB2	Specifies configuration settings for Natural for DB2, Natural for SQL/DS and Natural SQL Gateway.
NTDBGAT	Allows debugging of external Natural applications.
NTDS	Defines the sizes of storage buffers.

Macro	Purpose
NTDYNP	Controls the use of dynamic profile parameters. Corresponding dynamic profile parameter: DYNPARM.
NTEDBP	Controls buffer pool operations of the Software AG Editor.
NTIMSP	Specifies configuration settings for the Natural IMS TM Interface (Natural under IMS TM). No dynamic parameter specification possible.
NTIMSPE	Defines environment-specific parameter sets for the Natural IMS TM Interface (Natural under IMS TM). No dynamic parameter specification possible.
NTIMSPT	Defines Natural transaction codes for the Natural IMS TM Interface (Natural under IMS TM). No dynamic parameter specification possible.
NTLFILE	Associates physical database files with logical system files.
NTOPRB	Controls the use of database open/close commands for Adabas or VSAM.
NTOPT	Controls the use and the option settings of the Natural Optimizer Compiler.
NTOSP	Specifies configuration settings for the z/OS batch interface.
NTPGP	Defines properties for external programs.
NTPRINT	Specifies print file assignments.
NTRDC	Configures the Natural Data Collector and its trace recording function used by the SYSRDC and the Profiler utilities.
NTRPC	Controls the handling of Natural RPC (Remote Procedure Call).
NTSCTAB	Overwrites the scanner character definitions in the NATCONFIG module.
NTSORT	Controls the sort program used for the SORT statement.
NTSYS	Defines sets of dynamic profile parameters.
NTTAB	Overwrites the output character translation definitions in the NATCONFIG module.
NTTAB1	Defines alternative output character translation tables.
NTTAB2	Defines alternative input character translation tables.
NTTABA1	Overwrites the EBCDIC-to-ASCII conversion definitions in the NATCONFIG module.
NTTABA2	Overwrites the ASCII-EBCDIC conversion definitions in the NATCONFIG module.
NTTABL	Overwrites the "SYS" library output translation definitions in the NATCONFIG module.
NTTF	Converts database IDs and file numbers during program execution.
NTTRACE	Defines the Natural components to be traced.
NTTSOP	Specifies configuration settings for the Natural TSO Interface (Natural under TSO).
NTUSER	Restricts the use of dynamic parameter strings and alternative Natural parameter modules.
NTUTAB1	Overwrites the lower-case/upper-case conversion definitions in the NATCONFIG module.
NTUTAB2	Overwrites the upper-case/lower-case conversion definitions in the NATCONFIG module.

Macro	Purpose
NTVEXIT	Specifies user exits for VSAM files. Corresponds to the keyword subparameter EXIT of the dynamic profile parameter VSAM.
NTVLSR	Defines local shared resources subpools for VSAM files. Corresponds to the LSR keyword subparameter of the dynamic profile parameter VSAM.
NTVSAM	Specifies configuration settings for Natural for VSAM.
NTVSEP	Specifies configuration settings for the z/VSE batch interface.
NTVTVSD	Activates DFSMS Transactional VSAM Services. Corresponds to the TVSD keyword subparameter of the dynamic profile parameter: VSAM.
NTWEBIO	Enables or disables features of the Natural Web I/O Interface display.
NTWORK	Specifies the work files to be used during a session.
NTXML	Activates or deactivates the PARSE XML and REQUEST DOCUMENT statements.
NTZIIP	Configures zIIP (System z Integrated Information Processor) processing for z/OS.

See also [Example of Macros in the Natural Parameter Module](#).

Example of Macros in the Natural Parameter Module

In the following example of macro definitions in the Natural parameter module, *vrs* and *vr* denote a Natural product version.

```

NTPRM FNR=8,           System File for NTPRM *
      DBID=001,        Database ID for NTPRM *
      FNAT=(001,8),    Natural System File *
      FUSER=(001,9),   Natural User File *
      FDIC=(001,11),   Predict System File *
      FSEC=(001,10),   Natural Security File *
      FREG=(001,52),   Registry System File *
      ESIZE=128,       User Extension Area *
      SLOCK=SPOD,      Source Locking *
      THSIZE=0,        Thread Size *
      UCONMAX=0,       Max. Session Number *
      CSTATIC=(CMMSG,  Static. Modules Links *
      NSPPFUNC),       Dummy Static. Module *
      LE=OFF,          Record Limit Error *
      RECAT=OFF,       Allow Stow of Macros *
      PROFILE=,        Profile Batch *
      ADANAME=ADABAS,  Adabas Link Routine *
      ADASBV=OFF,     Form. Buffer not Pass.*
      DFOUT=S,         Output Format of Date *
      DFSTACK=S,      Date Format for Stack *
      NUCNAME=NATvrsSH, Natural Nucleus Name *
```

	AUTO=OFF,	Automatic Logon	*
	PC=ON,	PC Connection	*
	LS=250,	Default Line Size	*
	PS=80,	Default Page Size	*
	STACK=OFF,	Initial Natural Cmds.	*
	ET=OFF	END/BACKOUT TRANSACT.	*

NTDB2	BTIGN=ON,	Ignore Trans. Error	*
	CONVERS=ON,	Convers. Mode CICS	*
	CONVRS2=OFF,	Convers. Mode2 CICS	*
	DB2PLAN=PQANDBvr,	Plan Name	*
	DB2SSID=DB2A,	Subsystem ID	*
	DB2XID=ON,	Global Transaction ID	*
	DDFSERV=CMFSERV,	DD Name File Server	*
	DELIMID=OFF,	Delimited Identifiers	*
	MAXLOOP=10,	Nested Program Loops	*
	MAXSTMT=10,	Dynamic SQL Statements	*
	NNPSF=OFF,	Set Positive Sign	*
	NSBHOST=IBM2.HQ.SAG,	NSB Server Host Name	*
	NSBPORT=7311,	NSB Server TCP/IP Port	*
	PSCIGN=OFF,	Positive SQLCODEs	*
	REFRESH=OFF,	Refresh Setting	*
	RETRYPO=10,	Positioning Retries	*
	RWRDONL=ON,	Delimited Identifiers	*
	STATDYN=NEVER	Static Dynamic Switch	*

NTOSP	ABEXIT=ESTAE,	Abend Processing	*
	LBPNAME=' ',	Local Shared Buffer	*
	LEHDLR=ON,	LE Error Handler	*
	SUBPOOL=0,	Subpool for GETMAIN	*
	TIOBSZ=(8,64),	Primary I/O Buffer	*
	USERID=OFF	Init-User Job Name	*

NTVSAM	BTSUPP=ON,	BACKOUT TRANSACTION	*
	CLSUPP=ON,	Close Call at Session	*
	DDMCHK=OFF,	Support of DDM	*
	DDSWITE=0,	Maximum Entries DLBLY	*
	DFBE=10,	Decoded Format Buffer	*
	DFBN=100,	Format Buffer Entries	*
	ENADIS=OFF,	Enable Disabled Files	*
	ENAUNE=OFF,	Enable Unenabled Files	*
	ETSUPP=ON,	END TRANSACTION	*
	FORMAT=ON,	Record Formatting	*
	KEYLGH=126,	Length of VSAM Keys	*
	OPSUPP=OFF,	Dynamic Open Calls	*
	PATH=CHECK,	Path Processing	*
	PSIGNF=OFF,	Compiler Option PSIGNF	*
	RETRY=(OFF,OFF),	Retry ON ERROR Clause	*
	RLS=OFF,	Record-Level Sharing	*
	ROLLSIZ=550,	Session Status Info.	*
	SFILE=ON,	Support of VSAM Files	*
	TAFE=10,	Maximum No. DDMs	*

TAFN=50,	Maximum No. DDM Fields*
TIMEOUT=0,	Timeout RLS Request *
TSAE=20,	READ/FIND Statement *
TVS=OFF,	Support of DFSMSTVS *
UPDL=32768	Size of Update Table

III z/OS Environment

This part contains information about Natural under the operating system z/OS.

Natural under z/OS	Contains an overview of special considerations that apply when you are running Natural under z/OS online or in batch mode.
Authorized Services Manager	Describes the functionality and operation of the Authorized Services Manager (ASM) which is available under z/OS.
Natural Roll Server Functionality	Explains the functions of the Natural Roll Server in general, its use in a single z/OS system and in a z/OS Parallel Sysplex environment.
Natural Roll Server Operation	Provides information on the roll server system requirements, operation, performance tuning and restart capability.



Note: The codes that Natural may receive when the Roll Server is used during a Natural session runtime are output by the corresponding teleprocessing interfaces (*Natural under CICS* or *Natural under IMS TM*). For a list of these codes, refer to the *Return Codes and Reason Codes of the Roll Server Request* in the *Messages and Codes* documentation.

11 Natural under z/OS

▪ Natural Subsystem	82
▪ TP Monitor Interfaces	82
▪ Interfaces to Database Management Systems	83
▪ Natural in Batch Mode under z/OS	83
▪ Natural as a Server under z/OS	83

This document contains an overview of special considerations that apply when you are running Natural under z/OS.

Natural Subsystem

A Natural subsystem under z/OS consists of the following components:

- one or more **Global Buffer Pools**,
- an **Authorized Services Manager**,
- a **Roll Server**.

The Natural subsystem is identified by the Natural profile parameter `SUBSID` and by corresponding startup parameters for the components mentioned above. The default subsystem name is `NATv`, where `v` is the first digit of the current Natural version.

Via the Natural subsystem technique, multiple roll servers can be used simultaneously and multiple independent sets of global buffer pools can be created - in fact, multiple Natural runtime environments can be created which will be totally independent of one another.

TP Monitor Interfaces

For information on the TP monitor interfaces that are available with Natural under z/OS, refer to the sections

- *Natural under Com-plete*
- *Natural under CICS*
- *Natural under TSO*
- *Natural under IMS TM*

in the *Natural TP Monitor Interfaces* documentation.

Interfaces to Database Management Systems

Except for Software AG's database management system Adabas, all operations requiring database interaction are performed by a corresponding Natural interface module.

For information on the database interfaces that are available with Natural under z/OS, refer to the relevant separate documentation:

- *Natural for DB/2*
- *Natural for VSAM*
- *Natural for DL/I*

Natural in Batch Mode under z/OS

See *Natural in Batch Mode (All Environments)* and *Natural in Batch under z/OS*.

Natural as a Server under z/OS

Besides being a programming language, Natural can also act as a server in a client/server environment. For detailed information, see *Natural as a Server under z/OS*.

12 Authorized Services Manager under z/OS

■ ASM Overview	86
■ ASM System Requirements	87
■ ASM Operation	88

This document describes functionality and operation of the Authorized Services Manager (ASM) which is available with Natural under z/OS.

ASM Overview

The Authorized Services Manager (ASM) provides authorized operating system functions to Natural. These functions include writing SMF records and z/OS Parallel Sysplex communication through the Coupling Facility (CF). The ASM provides its functions via PC routines and runs in its own address space.

The following authorized functions are provided:

- communicating Natural buffer pool administration messages,
- write-access to global buffer pools in system key,
- writing SMF records,
- holding Natural session information in the Session Information Pool (SIP),
- executing authorized system services for IBM zIIP (System z Integrated Information Processor) support.

The first three functions are always available, whereas the SIP is optional and can be made available via startup parameter. For more information on starting the ASM, see [Starting the ASM](#).

You must use the ASM in the following cases:

- The Natural profile parameter `BPPROP` is set to `PLEX` or `GLOBAL` or `GPLEX` (buffer pool propagation is used).
- Natural global buffer pools are allocated in system key; see also [Allocation of the Natural GBP](#) in the section *Natural Global Buffer Pool under z/OS*.
- Natural under CICS is used in a z/OS Parallel Sysplex environment (SIP function required).
- Natural under IMS TM is used in terminal-oriented, non-conversational mode (with the SIP function).
- Natural under IMS TM is used, with the Accounting function writing SMF records.
- Enablement of zIIP support.

The Session Information Pool (SIP) holds the Natural session information records. In terminal-oriented non-conversational mode, the Natural CICS Interface and the Natural IMS TM Interface need these records to continue a Natural session after a terminal I/O. When running in a z/OS Parallel Sysplex environment, the SIP is created in the CF and a memory object is used as an intermediate buffer to avoid unnecessary access to the CF. Otherwise, the SIP is created in a memory object. (A memory object resides in 64-bit-addressable storage above the 2-gigabyte address).

If the ASM is used in a z/OS Parallel Sysplex environment, one ASM instance must be started in each participating z/OS image.

Note concerning Natural/CICS: The CICS System Recovery Table should include the z/OS system abend code 0D6.

ASM System Requirements

This section describes the ASM system requirements.

- [APF Authorization](#)
- [System Linkage Index](#)
- [CF Structure](#)
- [XCF Signaling Paths](#)

APF Authorization

Link the modules NATASM vr (where vr represents the relevant product version) and NATBPMGR to an Authorized Program Facility (APF) library, specifying IEWL parameter AC(1). Refer to *Installing Natural on z/OS*.

System Linkage Index

As the ASM reserves one system linkage index (System LX), check whether there is a high enough value of NSYSLX in member IEASYS xx of library SYS1.PARMLIB.



Note: If you terminate the ASM, the address space ID is no longer available because a System LX has been used. It becomes available again with the next IPL.

CF Structure

A CF structure is used if you run the SIP in a z/OS Parallel Sysplex environment.

XCF Signaling Paths

The XCF Signaling Services are used to propagate buffer pool administration messages in a z/OS Parallel Sysplex environment. The minimum message is 64 bytes long, the maximum is 2048 bytes. How often messages are sent depends on how often Natural objects are manipulated (with the system command CATALOG, STOW or DELETE).

ASM Operation

The following is covered below:

- [Starting the ASM](#)
- [ASM Messages, Condition Codes and Abend Codes](#)
- [ASM Operator Commands](#)

Starting the ASM

You start the ASM either as a batch job or as a started task by executing module NATASM vr , where vr represents the relevant product version. On the JCL EXEC statement, specify as PARM the following parameters:

subsystem-id, XCF-group-name, CF-structure-name, number-of-SIP-slots, SIP-slot-size, message-case

All parameters are positional and must be separated by a comma; they are explained in the table below:

Parameter	Possible Values	Default	Comment
<i>subsystem-id</i>	4-byte non-blank string	NAT v	The specified value must match the value of the Natural profile parameter SUBSID (v is version). Note: With Natural under CICS, refer to the CICSPLX parameter in the NCMDIR macro for setting the appropriate subsystem ID.
<i>XCF-group-name</i>	any valid XCF group name	none	The name of the XCF group for Signaling services.
<i>CF-structure-name</i>	any valid CF structure name	none	Optional, only needed if SIP is used. The name of the CF structure used for the SIP function.
<i>number-of-SIP-slots</i>	1 - 2147483647	none	Optional, only needed if SIP is used. The number of slots to be allocated if the CF structure has not yet been allocated. If omitted or specified as 0, the entire structure will be used for as many slots as it can hold.
<i>SIP-slot-size</i>	256, 512, 1024, 2048, 4096	1024	The specified value is ignored if a CF structure has already been allocated.
<i>message-case</i>	UCTRAN or blank	blank	Specify UCTRAN if the Authorized Services Manager is to issue all its messages in upper case.

Examples:

In the following examples, v or vr represents the relevant one- or two-digit product version

```
//ASM EXEC PGM=NATASMvr,PARM='NATv,NATXCF,CFSIP,1500,512'
```

The subsystem ID is `NATv`, the message group for buffer pool communication is `NATXCF`, the structure for the Session Information Pool is `CFSIP`. 1500 SIP slots are to be used, each having a size of 512 bytes.

```
//ASM EXEC PGM=NATASMvr,PARM='NATv,NATXCF,CFSIP'
```

Same as above, except SIP slots:

The ASM will use as many SIP slots as the `CFSIP` structure can hold, each having a size of 1024 bytes.

```
//ASM EXEC PGM=NATASMvr,PARM='NATv,NATXCF,,500,512'
```

The SIP service is not to use the Coupling Facility, but to build 500 SIP slots in storage, each having a size of 512 bytes.

```
//ASM EXEC PGM=NATASMvr,PARM='NATv,NATXCF'
```

The SIP service will not be available.

ASM Messages, Condition Codes and Abend Codes

The ASM writes informational and error messages to `JESMSG LG` using the `WTO` macro (`ROUTCDE=11`). The messages are preceded by a message identifier and the ASM's job name, for example:

```
ASM0005 FBASMvr
```

In this example, Authorized Services Manager Version `vr` (where `vr` represents the relevant product version) is active

The following condition codes are used:

Condition Code	Explanation
0	Normal completion
12	Wrong parameter input
16	Runtime error has occurred
20	Subtask has failed
24	Abend has occurred
>100	Working storage could not be allocated

The following user abend codes are used:

Abend Code	Reason	Comment
U0100	IXCJOIN failed.	Abend Register 14 contains the reason code.
U0101	IXCQUERY failed.	Abend Register 14 contains the reason code.
U0103	Active member list full.	Contact Software AG Support.
U0104	IXCMSGI failed.	Abend Register 14 contains the reason code.
U0105	Message Exit could not obtain a Purge Task Request Block.	Contact Software AG Support.
U0106	Work Space for IXLCONN could not be obtained.	Contact Software AG Support.
U02xx	DSPSERV CREATE failed.	xx is the reason code.
U03xx	ALESERV ADD failed.	xx is the reason code.
U04xx	ALESERV ADD failed.	xx is the reason code.
U05xx	IXLCONN failed.	xx is the reason code.
U06xx	IXLLIST WRITE failed.	xx is the reason code.

To find a description of reason codes, refer to *Programming: Sysplex Services Reference* (IBM documentation). If the error was environment-specific, and it is not clear what the reason was, contact Software AG Support.

ASM Operator Commands

The following commands can be passed to the ASM using the `MODIFY` command:

Command	Description
TERM	Terminates the ASM.
SNAP	Debugging function. The ASM's address space is dumped to SYSUDUMP.
VLIST	Display name, version, and assembly time of modules that are linked to the ASM.

For a list of return codes and reason codes of the SIP Service, refer to *SIP Service Return Codes and Reason Codes* in the *Messages and Codes* documentation.

13

Natural Roll Server Functionality

- Natural Roll-Server Overview 92
- Roll Server in a Single z/OS System 93
- Roll Server in a z/OS Parallel Sysplex Environment 94
- Roll File and LRB 96

See also *Natural Roll Server Operation*.

Natural Roll-Server Overview

With the Natural Roll Server, Natural can execute in a multiple-address-space system like CICS or IMS TM; these address spaces may be located in multiple z/OS images (z/OS Parallel Sysplex environment). You can, of course, also use the Roll Server if you are running a single z/OS system.

When Natural performs terminal I/O, it must save the application's context data (the thread): Before the terminal I/O is started, the thread is given to the Roll Server which keeps it in its Local Roll Buffer, or in the roll file. This is referred to as “roll out”. When the terminal I/O is completed, Natural requests the thread from the Roll Server, and continues the application. This is referred to as “roll in”. In a z/OS Parallel Sysplex environment, the Roll Server keeps information about the threads (the roll file directory) in a data structure in the Coupling Facility. Thus, it is possible for a Natural application to execute in different z/OS systems at different times: A thread can be given to the Roll Server on one system, and requested back from another system.

Before a roll out is performed, Natural compresses the thread into one contiguous buffer, and decompresses it after the roll-in is performed. Depending on the Natural version installed at your site, the CPU load of compression and decompression can be taken off the hosting TP system and moved to respective routines within the Roll Server. If you want to take advantage of this, use the appropriate Roll Server module NATRSM83 instead of NATRSM82 as described in the appropriate installation step in the section *Installing Natural on z/OS* in the *Installation for Natural on z/OS* documentation.

The Roll Server runs in its own address space. It provides its services as PC routines. In a z/OS Parallel Sysplex environment, one instance of the Roll Server must be started in each participating z/OS image.

A list of applied Roll Server Zaps is displayed in the JESMSG LG data set of the Roll Server started task, and by the SYSTP utility, function code R, line command Z.

Note concerning Natural under CICS: The CICS System Recovery Table should include the z/OS system abend code 0D6.

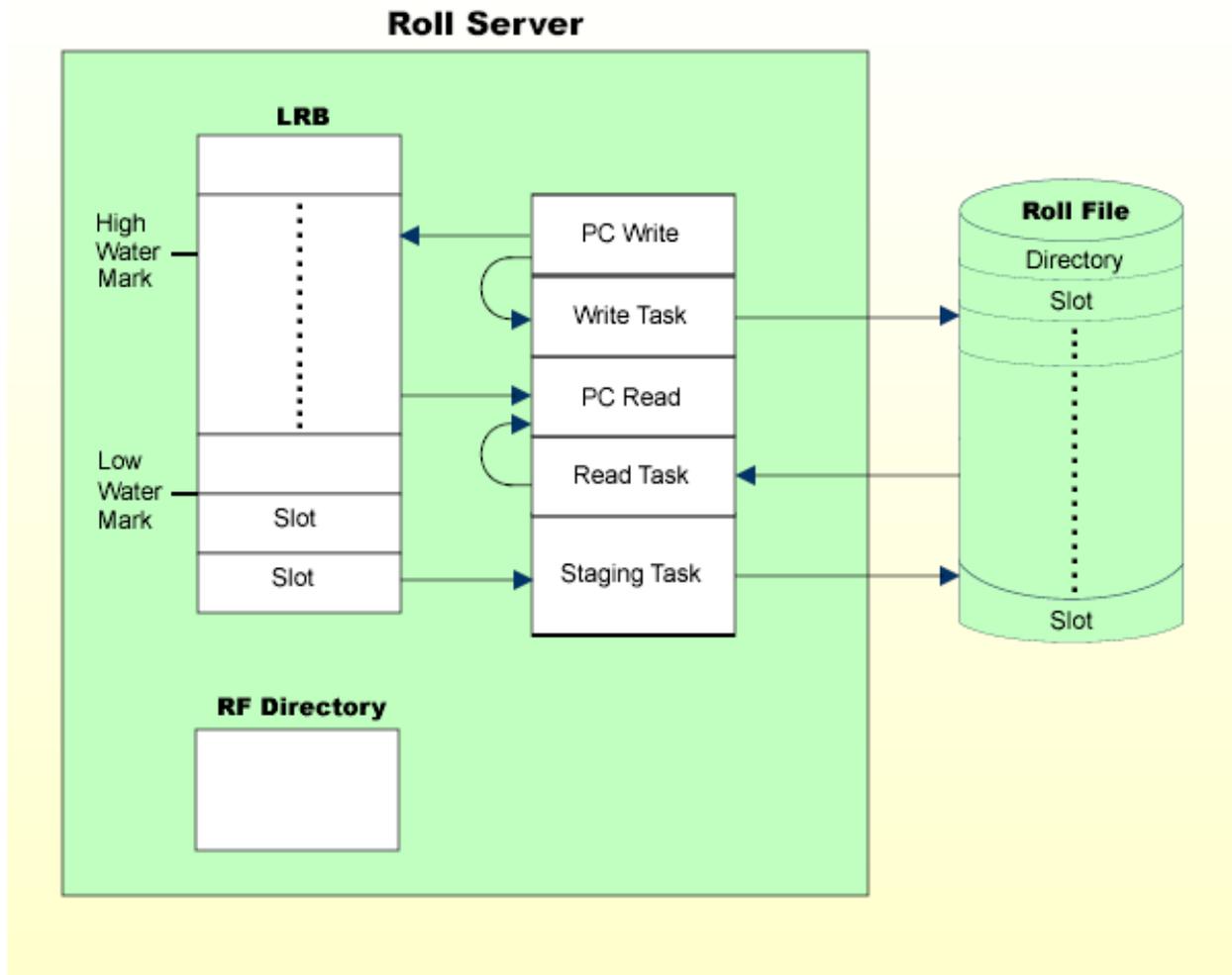
Roll Server in a Single z/OS System

When the Roll Server receives a thread through a write request (before terminal output), it checks whether enough space is available in the local roll buffer (LRB). If there is, the thread is copied to the LRB. If not, the thread is written to the roll file. If the thread is larger than the roll file slot size, additional overflow slots are allocated to accommodate the thread. Allocation of overflow slots is restricted to the roll file that the Natural session was initially assigned to. If the roll file does not have enough free space to allocate the necessary overflow slots, an error is generated and the requesting Natural session terminates abnormally. Overflow slots are implicitly freed by a subsequent write request with a smaller thread.

When the Roll Server receives a read request for the thread (after terminal input), it tries to locate the thread in the LRB. If the thread is found, it is copied from the LRB to the requestor's address space. If not, the thread is read from the roll file and copied to the requestor's address space.

To ensure that the system performs well and that there is always enough space in the LRB, there are “water marks”. If the LRB's high water mark is reached, the staging task is activated and copies the LRB content to the roll file until the low water mark is reached. Where the high water mark and the low water mark are placed is therefore an important issue of performance tuning. For more information on performance tuning, see the section [Roll Server Performance Tuning](#). For a Roll Server running in a single z/OS system, the default high water mark is 80 percent and the low water mark 70 percent.

Illustration of the Roll Server in a Single z/OS System:



Roll Server in a z/OS Parallel Sysplex Environment

In a z/OS Parallel Sysplex environment, the Roll Servers in the participating z/OS images communicate through the Coupling Facility's (CF) XCF Signaling Services, and the roll file directory resides in an XES data structure.

When the Roll Server receives a thread through a write request (before terminal output), it checks whether enough space is available in the local roll buffer (LRB). If there is enough space, the thread is copied to the LRB. If there is not enough space in the LRB, the thread is written directly to the roll file. The roll file directory in the CF structure is updated accordingly. Thread overflow is handled as described under *Roll Server in a Single z/OS System*.

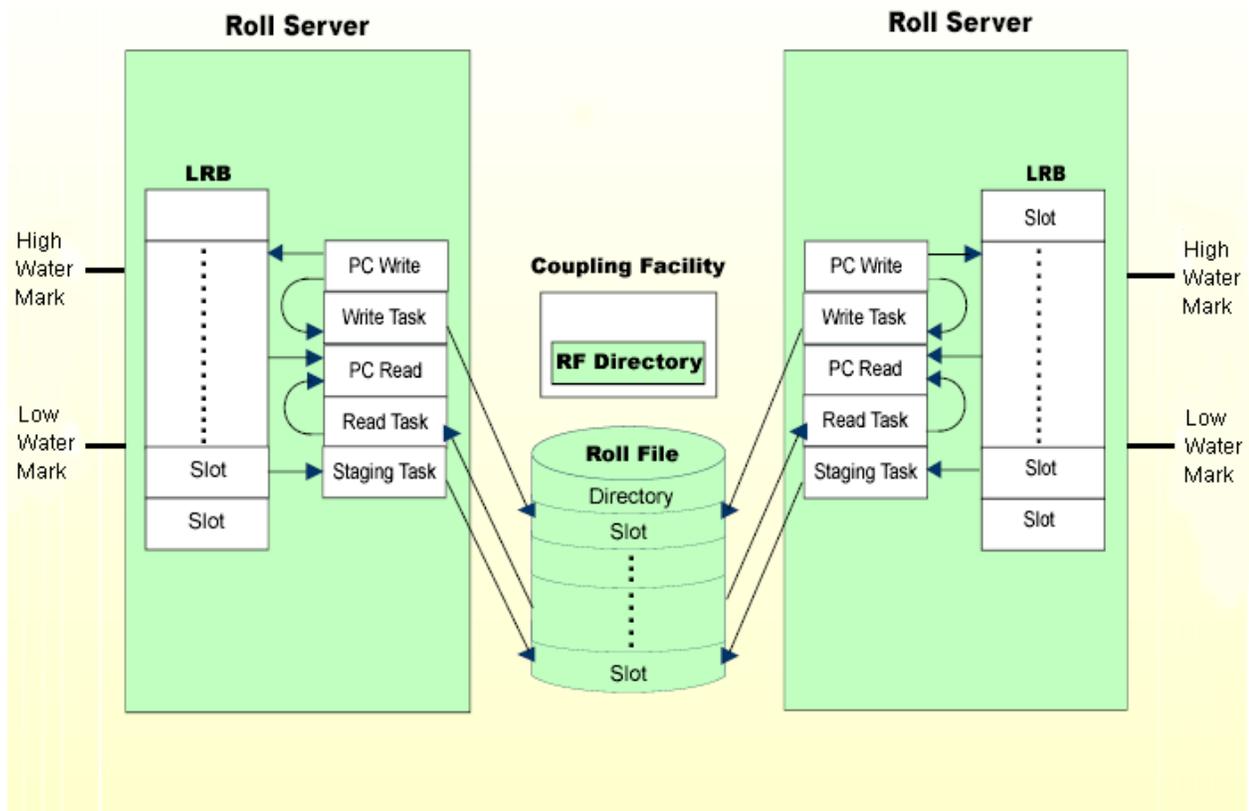
You can also set high and low water marks in a Parallel Sysplex environment. This option is not provided by older Natural versions. The staging task writes threads to disk until the low water mark is reached only when the LRB high water mark is reached. If a thread is requested from another z/OS image and has not yet been written to disk, the Roll Server on the other z/OS image sends a stage request message for this particular thread. The requested thread is then written to disk regardless of the high and low water marks.



Note: When you specify a high water mark of zero, the Roll Server performs identically to earlier versions of Natural in that all threads are immediately scheduled for staging to disk. For a Roll Server running in a Parallel Sysplex environment, both the default high and the low water marks are zero.

When the Roll Server receives a read request for a thread (after terminal input) and the last write request was issued in the same z/OS image, the Roll Server copies the thread directly from the LRB into the requestor's address space. If the last write request did not come from the same z/OS image, the thread is read from the roll file and then copied into the requestor's address space. If the thread does not yet reside on the roll file, the Roll Server on the other z/OS image receives a stage request message. When the thread resides on the roll file, it is read from there and then copied into the requestor's address space.

Illustration of Roll Servers in a z/OS Parallel Sysplex Environment:



Roll File and LRB

The roll file is a BDAM file logically subdivided into a directory and fixed-length slots. The slot size is a parameter of the roll-file formatting routine NATRSRFI. There should be at least as many slots as there are active Natural sessions. The slot size should be large enough to accommodate an average compressed Natural thread. Threads that are larger than the slot size will occupy multiple slots. You can check the distribution of Natural thread sizes with the SYSTP utility, function code R, line command R: Scroll down (using PF8) to the page entitled **Roll Server Peak Loads and Thread Sizes**.

The roll file directory contains one entry for each active Natural session, together with a timestamp of its last write request. In a single z/OS system, the directory resides in the Roll Server's address space. In a z/OS Parallel Sysplex environment, it resides in the Coupling Facility. The directory is written back to the roll file only when the Roll Server terminates.

The local roll buffer acts as a cache for the roll file. The roll buffer is contained in a z/OS memory object and subdivided into fixed-length slots. The LRB slot size is identical to the slot size of the corresponding roll file. If the Roll Server is to run without a roll file, the LRB slot size must be

specified as parameter on the JCL EXEC statement. See *Natural Roll Server Operation, Starting the Roll Server*.

The Roll Server can run with up to five different roll files. Each of these roll files is logically connected to one local roll buffer. If there are five roll files, there are five corresponding LRBs. Each roll file is accessed by its own dedicated read, write, and staging tasks. Thus, if the roll files are created on different disks on different channels, the roll files can be accessed simultaneously.

Natural users are allocated to the roll file that has the most free slots. You can use the **NATRSU14 user exit** to implement your own allocation method. For more information on this user exit, see the relevant section in *Natural Roll Server Operation*.

You can see how your user IDs are distributed in the roll files with the **Natural Sub-Systems and Roll Server Information** statistics function (function code **R**) of the SYSTP utility.

14 Natural Roll Server Operation

▪ Roll Server System Requirements	100
▪ Formatting the Roll File	101
▪ Starting the Roll Server	104
▪ Roll Server Messages, Condition Codes and Abend Codes	107
▪ Return Codes and Reason Codes of the Roll Server Request	108
▪ Operating the Roll Server	109
▪ Roll Server Performance Tuning	110
▪ Roll Server User Exits	111

See also [Natural Roll Server Functionality](#).

Roll Server System Requirements

This section describes the Roll Server system requirements.

- [APF Authorization](#)
- [System Linkage Index](#)
- [Virtual Storage](#)
- [CF Structure](#)
- [XCF Signaling Paths](#)

APF Authorization

Link the module NATRSM vr (vr represents the relevant product version) to an Authorized Program Facility (APF) library, specifying IEWL parameter AC(1). Refer to *Installing Natural on z/OS*.

System Linkage Index

As the Roll Server reserves one system linkage index (System LX), check whether there is a high enough value of NSYSLX in member IEASYSxx of library SYS1.PARMLIB.

When the Roll Server terminates, its address space ID is no longer available because a System LX has been used. It becomes available again with the next IPL.

Once a System LX has been reserved, it is reused with every restart of the Roll Server until the next IPL.

Virtual Storage

Storage	Size
ECSA	84 bytes
Private program storage	30 KB above
Fixed subpool storage (incl. ELSQA):	10 KB below, 50 KB above
LRB directory	32+(64*number of LRB slots)
100 slots per roll file	4 KB above
Every additional roll file	30 KB above
Working storage	depending on load, about 1 MB above

CF Structure

A CF structure is used to hold the roll file directory in a z/OS Parallel Sysplex environment.

XCF Signaling Paths

In a z/OS Parallel Sysplex environment, the Roll Servers communicate via the XCF Signaling Services. As the default XCF group name, the leftmost eight bytes of the CF structure name are used.

If you want to specify your own XCF group name, use the NATRSU24 user exit. For more information on this user exit, see [NATRSU24 User Exit](#).

Formatting the Roll File

To format the roll file, proceed as follows:

1. Allocate it as a physical, sequential data set with a fixed-record format.
2. Format it using module NATRSRFI.

During formatting, the roll file is converted to BDAM format with a device-dependent block size.



Note: If you plan to use an existing roll file of a previous version, it is sufficient to execute the NATRSRFI RESET function.

To format, enter the following parameter string under the DD name RFIParams, or as PARM on the JCL EXEC statement:

```
function,dd-name,slot-size,number-of-slots
```

All parameters are positional; they are explained in the table below:

Parameter	Description	
<i>function</i>	FORMAT	Format the roll file.
	RESET	All roll file slots are reset (marked as free). You can only use this parameter value if the roll file has already been formatted. The only other parameter allowed is <i>dd-name</i> .
	LIST	Print a list of session IDs contained in the roll file and their last activity. The only other parameter allowed is <i>dd-name</i> .
<i>dd-name</i>	The name of the DD statement under which the roll file has been specified.	

Parameter	Description
<i>slot-size</i>	<p>The size of a roll file slot in bytes. This size is rounded to the next higher multiple of the block size used.</p> <p>It is recommended to initially use a slot size equal to the size of the Natural thread. Then look at the Roll Server statistics. They also show the largest occurrence of a thread size. Use this value to reduce the slot size, if necessary.</p>
<i>number-of-slots</i>	<p>The number of roll file slots to be allocated. This number is the maximum number of concurrently active users.</p> <p>This parameter is optional. If omitted, the entire roll file, as allocated, will be formatted.</p> <p>Note that during formatting, the system abend code SB37 or SD37 (or S209 for a VSAM file) can be encountered. This abend is intercepted by the formatting routine and can be ignored.</p>

To calculate the required disk space in cylinders for a roll file (`SPACE` parameter of the `DD` statement), use the following formula:

```
number-of-cylinders = ceiling (number-of-slots * slot-size / (30*block-size))
```

or in tracks

```
number-of-tracks = ceiling (number-of-slots * slot-size / (2*block-size))
```

The block size used is:

23476 for 3380 DASD

27998 for 3390 DASD

22928 for 9345 DASD

In addition, space is needed for the roll file directory header (40 bytes) and one directory entry for each roll file slot (24 bytes). Thus, one additional block is needed for roughly 976 slots on 3380, 1164 slots on 3390, or 953 slots on 9345 DASD.

NATRSRFI Output

If a `DD` statement with `ddname` `RFIPRINT` is specified, `NATRSRFI` directs its output to this data set. When `RFIPRINT` is omitted, output is written to `JESMSG LG` using the `WTO` macro (`ROUTDCE=11`). Note that `RFIPRINT` must be specified for the `LIST` function.

NATRSRFI Condition and Abend Codes:

The following condition codes are used:

Code	Explanation
0	Normal completion.
4	Number of slots formatted is less than requested.
20	Parameter error.

The following user abend codes are used:

Abend Code	Cause
U0100	Open for RFIPARMS or RFIPRINT failed.
U0101	Open for roll file failed.

Examples:

In the following examples, *vr* or *vrs* represents the relevant product version.

Example 1:

```
//FBRUNRFI JOB (FB,218),FB,CLASS=K,MSGCLASS=X,NOTIFY=FB
//FORMAT EXEC PGM=NATRSRFI
//STEPLIB DD DISP=SHR,DSN=NATURAL.NAT $vr$ .LOAD
//RF1 DD DISP=SHR,DSN=FB.SYSF.ROLLF1
//RF2 DD DISP=SHR,DSN=FB.SYSF.ROLLF2
//RFIPARMS DD *
FORMAT,RF1,200000,1000
FORMAT,RF2,200000
```

Excerpt from resulting JESMSGLG:

```
+FBRUNRFI: FORMAT,RF1,200000,1000
+FBRUNRFI: RF1: FB.SYSF.ROLLF1
+FBRUNRFI: Creation date: 2001/06/13 Volume: ADA002(3390)
IEC031I D37-04,IFG0554P,FBRUNRFI,FORMAT,RF1,305B,ADA002,FB.SYSF.ROLLF1
+FBRUNRFI: Not enough space for 1000 slots.
+FBRUNRFI: 60 Blocks written. Block size is 27998.
+FBRUNRFI: 1 Directory block.
+FBRUNRFI: 8 Blocks per slot. Slot size is 223984.
+FBRUNRFI: 7 Slots initialized. Roll file version  $vrs$ .
+FBRUNRFI: 3 Blocks unused.
+FBRUNRFI: FORMAT,RF2,200000
+FBRUNRFI: RF2: FB.SYSF.ROLLF2
+FBRUNRFI: Creation date: 2001/06/08 Volume: USRF08(3380)
IEC031I D37-04,IFG0554P,FBRUNRFI,FORMAT,RF2,020F,USRF08,FB.SYSF.ROLLF2
+FBRUNRFI: 60 Blocks written. Block size is 23476.
+FBRUNRFI: 1 Directory block.
+FBRUNRFI: 9 Blocks per slot. Slot size is 211284.
+FBRUNRFI: 6 Slots initialized. Roll file version  $vrs$ .
+FBRUNRFI: 5 Blocks unused.
```

Example 2:

```
//FBRUNRFI JOB (FB,218),FB,CLASS=K,MSGCLASS=X,NOTIFY=FB
//FORMAT EXEC PGM=NATRSRFI,PARM='FORMAT,RF1,200000'
//STEPLIB DD DISP=SHR,DSN=NATURAL.NATvr.LOAD
//RF1 DD DISP=SHR,DSN=FB.SYSF.ROLLF1
//RFIPRINT DD SYSOUT=X
```

Resulting RFIPRINT:

```
Natural Roll Server - Roll File Utility Version
vrs
FORMAT,RF1,200000
RF1: FB.SYSF.ROLLF1
Creation date: YYYY/MM/DD Volume: ADA002(3390)
60 Blocks written. Block size is 27998.
1 Directory block.
8 Blocks per slot. Slot size is 223984.
7 Slots initialized. Roll file version vrs.
3 Blocks unused.
```

Notes Concerning the Formatting or Resetting of Roll Files

- You can format or reset several roll files at once by specifying several parameter lines in RFIPARMS.
- You cannot format or reset a roll file while the roll server is active.
- When the roll file is formatted in a z/OS Parallel Sysplex environment, the roll server Coupling Facility structure must also be cleared using the SETXCF operator command, for example:

```
SETXCF FORCE,STR,STRNAME=NATROLL1
```

Starting the Roll Server

You start the Roll Server either as a batch job or as a started task by executing module NATRSM vr (where vr represents the relevant product version). The roll file(s) must be defined as DD statements with $ddname$ ROLLF1 to ROLLF5.

On the JCL EXEC statement, specify as PARM the following parameters:

subsystem-id, number-of-roll-files, number-of-LRB-slots, LRB-slot-size, CF-structure-name, low-water-mark, high-water-mark, non-activity-time, timeout-check-time, message-case

All parameters are positional and must be separated by a comma. They are explained in the table below:

Parameter	Possible Values	Default	Comment
<i>subsystem-id</i>	4-byte non-blank string	NAT <i>v</i>	The specified value must match the value of the Natural profile parameter SUBSID (<i>v</i> = version number). Note: With Natural under CICS, refer to the ROLLSRV parameter in the NCMDIR macro for setting the appropriate subsystem ID.
<i>number-of-roll-files</i>	0 - 5	1	In a z/OS non-Parallel Sysplex environment, the Roll Server can operate without a roll file, using only the in-storage Local Roll Buffer.
<i>number-of-LRB-slots</i>	1 - 32767	none	The number of LRB slots multiplied by the slot size must not exceed 2 GB. The same number of LRB slots is assigned for each LRB, i.e. for each roll file used. The total number of LRB slots is calculated by the formula: $\text{number-of-roll-files} * \text{number-of-LRB-slots}$
<i>LRB-slot-size</i>	any numeric value	roll file slot size	Value in number of bytes. This parameter must be specified if no roll file is used. If roll files are used, this parameter is ignored and the roll file slot size is used instead.
<i>CF-structure-name</i>	any valid structure name	none	If you specify less than 16 characters, blanks are appended. Only specify this parameter if you use the Coupling Facility (with z/OS Parallel Sysplex).
<i>low-water-mark</i>	0 - 9	7 (single z/OS) 0 (sysplex)	Specifies the low water mark in steps of 10 percent of the number of LRB slots.
<i>high-water-mark</i>	0 - 10	8 (single z/OS)	Analogous to <i>low-water-mark</i> parameter. Value "10" means that the staging task will never be activated. It is only recommended to specify "10" if

Parameter	Possible Values	Default	Comment
		0 (sysplex)	the LRB is large enough to serve all simultaneously active Natural sessions.
<i>non-activity-time</i>	1 - 999999	none	Number of hours a session can be inactive before it is deleted from the roll file. If this time is exceeded, the session is deleted during the next scheduled timeout check. If this parameter is omitted, no timeout check will be executed. This parameter can be changed using operator command TIMEOUT , see below.
<i>timeout-check-time</i>	0000 - 2359	none	The time of day that the timeout check is to be run. Sessions will be deleted if they have been inactive longer than the non-activity time specified by the preceding parameter. If this parameter is omitted, no timeout check will be scheduled. This parameter can be changed using operator command TIMEOUT , see below.
<i>message-case</i>	UCTRAN or blank	blank	Specify UCTRAN if the Roll Server is to issue all its messages in upper case.

 **Note:** The Local Roll Buffer resides in a Memory Object “above the bar”. Use the **MEMLIMIT** parameter on the **EXEC** statement to ensure enough memory can be allocated “above the bar”.

Examples for Starting the Roll Server as a Batch Job

In the following examples, *vr* represents the relevant product version

```
// EXEC PGM=NATRSMvr,PARM='NAvr,,1000'  
//ROLLF1 DD DSN=SYSF.ROLLFILE
```

The subsystem ID is *NAvr*, one roll file is used (default), and the Local Roll Buffer has 1000 slots. The slot size used is identical with the roll file's slot size. The low water mark is 70 percent (default), the high water mark is 80 percent (default).

```
// EXEC PGM=NATRSMvr,PARM=',5,1000,150000,NATROLL1',MEMLIMIT=800M
//ROLLF1 DD DSN=DASD1.ROLLFILE
//ROLLF2 DD DSN=DASD2.ROLLFILE
//ROLLF3 DD DSN=DASD3.ROLLFILE
//ROLLF4 DD DSN=DASD4.ROLLFILE
//ROLLF5 DD DSN=DASD5.ROLLFILE
```

The subsystem ID is NATV (default), five roll files are used, and each of the five Local Roll Buffers has 1000 slots. The LRB slot size is 150000 bytes. The roll file directory resides in the Coupling Facility structure NATROLL1. Low and high water marks are ignored, because every thread is written to the roll file (see [Natural Roll Server Functionality](#)). Since this job is intended for z/OS, the MEMLIMIT option specifies 800 Megabytes for the Local Roll Buffers.



Note: The Roll Server will not start in the following cases:

- Another Roll Server is running with the same *subsystem-id*.
- Another Roll Server is accessing a roll file specified in its JCL
- A roll file has been reformatted without resetting the CF structure, using the SETXCF FORCE command.

Roll Server Messages, Condition Codes and Abend Codes

The Roll Server writes informational and error messages to JESMSG LG using the WTO macro (ROUTCDE=11). The messages are preceded by a message identifier and the Roll Server's job name, for example:

```
RSM0019 FBRSMvrs: Roll Server Version vrs is active
```

where *vrs* represents the relevant product version.

The messages are explained in the section *Roll Server Messages* in the *Messages and Codes* documentation.

Condition Codes of the Roll Server Started Task

The following condition codes are used:

0	Normal completion
12	Wrong parameter input
16	Runtime error
20	Abend has occurred
>100	Initialization error

User Abend Codes

When an unexpected return code is issued by an XCF or XES Service Call, an abend with a dump is forced. Register 14 of the abend register contains the reason code. To find a description of the reason, refer to *Programming: Sysplex Services Reference* (IBM documentation). If the error was not environment-specific, send the dump to Software AG support.

The following user abend codes are used:

Abend Code	Cause
U0200	IXLCONN failed
U0201	IXLFORCE failed
U0202	IXLLIST failed
U0203	IXLDISC failed
U0204	IXCLEAVE failed
U0301	IXLLIST failed
U0302	IXCMMSGO failed
U0401	IXLLIST failed
U0501	IXLLIST failed

Return Codes and Reason Codes of the Roll Server Request

These are codes that Natural may receive from the Roll Server's PC services routines. They are reported by the respective teleprocessing interfaces (Natural CICS Interface or Natural IMS TM Interface). For a list of these codes, refer to the *Return Codes and Reason Codes of the Roll Server Request* in the *Messages and Codes* documentation.

Operating the Roll Server

The following commands can be passed to the Roll Server via the MODIFY operator command:

Command	Description	
DIAGNOSE	Debugging function, only to be used at Software AG's advice. This command does not have any function. Its intended future use is in connection with special Zaps to aid in diagnosing specific customer problems, as the need arises.	
HWM <i>n</i>	Sets the LRB high water mark to <i>n</i> times 10 percent of the number of LRB slots (<i>n</i> =0-10). If <i>n</i> is not specified, the current low and high water marks are displayed.	
LWM <i>n</i>	Sets the LRB low water mark to <i>n</i> times 10 percent of the number of LRB slots (<i>n</i> =0-9). If <i>n</i> is not specified, the current low and high water marks are displayed.	
SNAP	Debugging function. The Roll Server's address space is dumped to SYSUDUMP.	
STATS	Write Roll Server statistics to JESMSG LG using the WTO macro (ROUTCDE=11). Statistics include information about roll-out and roll-in activity, as well as roll file I/O.	
TERM	Stops the Roll Server. The roll file directory and all modified LRB slots are written to the roll file and the address space is terminated. The address space ID is no longer available until the next IPL. Statistics are written to JESMSG LG using the WTO macro (ROUTCDE=11). Statistics include information about roll-out and roll-in activity, as well as roll file I/O.	
TIMEOUT	NAT <i>nnn</i>	Specifies or replaces the non-activity time parameter.
	TOC <i>hhmm</i>	Specifies or replaces the time of day of the timeout check.
	OFF	Disables timeout checking.
	ON	Reinstates timeout checking.
	NOW	Starts an immediate timeout check. Normal timeout check scheduling (if specified) remains in effect.
	TERSE	Suppresses the messages RSM0072 and RSM0074 during TIMEOUT NOW processing. Message RSM0047 Operator command: TIMEOUT NOW is also suppressed.
	VERBOSE	Displays the messages RSM0072, RSM0074 and RSM0047 Operator command: TIMEOUT NOW. This is the default setting.
	? (or no specification)	Displays current timeout settings. The question mark (?) is optional and can be omitted.

Roll Server Performance Tuning

As a general rule for Roll Server performance tuning, give the Roll Server a higher dispatching priority than the address spaces where Natural runs.

To find out where the weaknesses in performance are, analyze the system performance using the *Natural Subsystems and Roll Server Information* function of the SYSTP utility.

When looking at Roll-Server Statistics, keep an eye especially on the following values:

- The number of direct writes.

"Direct write" means that the Natural thread that was received was written to the roll file directly.

There are two possible reasons:

1. No LRB slot available. Increase the LRB.
2. The compressed thread was larger than a single LRB slot. Increase the LRB slot size.

- The number of direct reads.

"Direct read" means that the requested thread was no longer in the LRB and had to be read directly from the roll file.

If the ratio of direct reads to the total number of reads is very high in a single z/OS system, the LRB is too small (increase it).

If the ratio of direct reads to the total number of reads is very high in a z/OS Parallel Sysplex environment, this may also mean that there are many inter-system activities, which in turn means that a Natural session changes z/OS images quite frequently during its lifetime.

- The number of staging waits (in a single z/OS environment).

A "staging wait" is a situation where a write request had to wait until the Staging Task had written the LRB slot to the roll file. If the ratio of staging waits to the total number of write requests is very high, this indicates that the high and low water marks are set inappropriately or that there is a bottleneck on the roll file device/roll file channel.

Based on experience with stress tests, the following is recommended:

If the ratio of maximal number of active users to number of LRB slots is very small, increase the high water mark. If not, decrease the high water mark.

The difference between high water mark and low water mark should not be larger than three (30 percent).

Ideally, if the number of LRB slots is definitely larger than the maximum number of concurrent users, the high water mark should be set to 10.

Roll Server User Exits

The roll server has two user exits.

- NATRSU14
- NATRSU24

Sample source modules are delivered for these.

NATRSU14 User Exit

Specifies the roll file number to be used.

Entry calling conventions:

- Register 1 addresses the parameter list that is described by the following DSECT:

```

PLIST      DSECT
PLRSVER    DS CL4    Roll server version (= 'vrs')
PLNRF      DS H      Number of roll files
PLUID      DS CL16   Userid
PLTSNUM1   DS H      Total number of slots Roll file 1
PLUSNUM1   DS H      Number of slots in use Roll file 1
PLTSNUM2   DS H      Total number of slots Roll file 2
PLUSNUM2   DS H      Number of slots in use Roll file 2
PLTSNUM3   DS H      Total number of slots Roll file 3
PLUSNUM3   DS H      Number of slots in use Roll file 3
PLTSNUM4   DS H      Total number of slots Roll file 4
PLUSNUM4   DS H      Number of slots in use Roll file 4
PLTSNUM5   DS H      Total number of slots Roll file 5
PLUSNUM5   DS H      Number of slots in use Roll file 5
PLISTL    EQU *-PLIST

```

where *vrs* represents the relevant product version.

- Register 13 points to a 36-fullword save area.
- Register 14 contains the return address.
- Register 15 contains the entry address of NATRSU14.

Return calling convention:

- Register 15 contains the number of the roll file in binary format.



Note: If access registers are modified within this user exit, these access registers must be saved and restored on return. This user exit is called in primary addressing mode with PSW Key 8. Since it runs in cross-memory mode, no SVC except SVC 13 may be used.

NATRSU24 User Exit

Specifies the XCF group name to be used.

Entry calling conventions:

- Register 1 points to an 8-byte area in which the group name must be generated.
- Register 13 points to an 18-fullword save area.
- Register 14 contains the return address.
- Register 15 contains the entry address of NATRSU24.

As a group name default, the Roll Server will use the leftmost 8 bytes of the CF structure name.

This user exit is called in primary mode, PSW Key 8 and in task mode.

IV z/VSE Environment

15 z/VSE Environment

- Natural Subsystem 116
- TP Monitor Interfaces 116
- Interfaces to Database Management Systems 116
- Natural in Batch Mode under z/VSE 117

This document contains special considerations that apply when you are running Natural under z/VSE.

Natural Subsystem

A Natural subsystem under z/VSE consists of the following components:

- one or more **global buffer pools**.

The Natural subsystem is identified by the Natural profile parameter `SUBSID` and by corresponding startup parameters for the components mentioned above. The default subsystem name is `NAT v` , where v is the first digit of the current Natural version.

TP Monitor Interfaces

For information on the TP monitor interfaces that are available with Natural under z/VSE, refer to the sections

- Natural Com-plete Interface
- Natural CICS Interface

in the Natural *TP Monitor Interfaces* documentation.

Interfaces to Database Management Systems

Except for Software AG's database management system Adabas, all operations requiring database interaction are performed by a corresponding Natural interface module.

For information on the database interfaces that are available with Natural under z/VSE, refer to the relevant separate documentation:

- Natural for SQL/DS
- Natural for VSAM
- Natural for DL/I

Natural in Batch Mode under z/VSE

See *Natural in Batch Mode (All Environments)* and *Natural in Batch under z/VSE*.

V

BS2000 Environment

This part contains special considerations that apply when running Natural on the operating system BS2000.



Important: Software AG products on BS2000 are affected by the BS2000 C/C++ time overflow problem which can occur after January 19, 2018. For instructions on how to proceed to avoid this problem, refer to *BS2000 C/C++ Time Overflow* in the current *Natural Release Notes*.

Refresh of Natural Load Pool

Explains the applicability and the use of the load-pool refresh program.

Optimization of Message Handling

Describes the screen output optimization method used by Natural and the facilities to restore the most recent terminal screen content.

Terminal Types Supported under BS2000

Provides information on the various types of terminals that are supported by Natural under BS2000.

Function Keys Supported under BS2000

Describes the specific Natural function-key assignments that are supported for terminals under BS2000.

Common Memory Pools

Provides information on the global and local common memory pools.

Calling Dynamically Reloadable 3GL Programs

Defines rules for address mode selection when calling dynamically reloadable 3GL programs in a Natural application.

Print File/Work File Server NATPWSV2

Describes the print file/work file server NATPWSV2 for an RPC batch server environment under BS2000.

Using Network Attached Storage Files as Natural Print and Work Files

Provides information on how files on Network Attached Storage servers can be processed as Natural print and work files.

RPC Server Front-End

Describes the RPC server front-end for an RPC batch server environment under BS2000 with the print file/work file server NATPWSV2.

Related Topics

See also:

- [Natural in Batch Mode under BS2000](#)
- [Using Natural with TP Monitors](#) in the *TP Monitor Interfaces* documentation
- [Natural under openUTM](#) in the *TP Monitor Interfaces* documentation
- [Natural under TIAM](#) in the *TP Monitor Interfaces* documentation

Other Natural Functions for BS2000-Specific Purposes

Natural provides the following functions for BS2000-specific purposes:

- **P-Key Utility**
Supports the loading of programmable P keys on terminal devices of the 975*n* series (types 974*n*, 975*n* and 976*n*).
- **Swap Pool Manager**
Controls the use of the Natural swap pool (under *openUTM* and under CICS).

These functions are part of the Natural utility SYSTP.

16 Refresh of Natural Load Pool

- Prerequisites and Restrictions 122
- Procedure 122
- Keyword Parameters for the Program PREFRESH 123

This document describes the prerequisites, restrictions and procedures that are applicable for refreshing a Natural load pool and contains a list of the keyword parameters provided in the `PREFRESH` program.

Prerequisites and Restrictions

- The Natural load pool must have been started with the keyword parameter `ACCS=WRITE`, using the program `CMPSTART`.
- A Natural load pool which is also used by batch applications must *not* be refreshed while the Natural batch applications are in operation. A refresh is admissible only with `TIAM` and `openUTM` applications.
- A new Natural nucleus can be loaded only into a *global* common memory pool.

Procedure

- When a new Natural nucleus is to be loaded into the common memory pool, the name of the linked environment-independent nucleus must be identical with the existing name. The name of the Natural nucleus is equal to the name of the global common memory pool.

Example:

The existing Natural nucleus was started with the following parameters using the program `CMPSTART`:

```
/EXEC (CMPSTART,NATURAL.MOD)
NAME=NATSHARE, POSI=ABOVE, ADDR=250, PFIX=YES, SIZE=2MB, ALNK=NO
ACCS=WRITE, LIBR=NATURAL.USER.MOD.A
```

- The newly linked Natural nucleus is to be loaded from the library `NATURAL.USER.MOD.B` into the global common memory pool. This is accomplished with the program `PREFRESH`.

Example:

```
/.PREFRESH LOGON
/OPTION DUMP=YES
/SYSFILE SYSOUT=LST.PREFRESH.NATSHARE
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (PREFRESH,NATURAL.MOD)
NAME=NATSHARE, LIBR=NATURAL.USER.MOD.B
/LOGOFF N
```

or:

```

/load (prefresh,natural.mod) <enter>
% BLS0517 MODULE 'PREFRESH' LOADED
/r <enter>
*name=natshare,libr=natural.user.mod.b <enter>
* <enter>
REFR050: LOAD POOL NATSHARE IS SUCCESSFULLY REFRESHED
/

```

- The successful loading of the new Natural nucleus is confirmed by the message:

```
REFR050: LOAD POOL name IS SUCCESSFULLY REFRESHED
```

Keyword Parameters for the Program PREFRESH

The program PREFRESH has the following keyword parameters:

NAME | **LIBR** | **LOAD** | **ALNK** | **TIM1** | **TIM2**

The program PREFRESH has the following syntax (If available, default values are shown.):

```
REFRESH NAME=name,LIBR=library,LOAD=BIND,ALNK=NO,TIM1=10,TIM2=20
```

NAME - Common Memory Pool and Module Name

This parameter determines the name of the module and the name of the common memory pool. The name must be specified. No default value exists.

NAME=XXXXXXXX	XXXXXXXX: valid module and common memory pool name. The name must be identical with the existing module/common memory pool name. The maximum number of characters is 8.
---------------	---

LIBR - Load Library

This parameter determines from where the defined module is to be loaded. The name must be specified. No default value exists.

LIBR= <i>library</i>	<i>library</i> is the name of the load library.
----------------------	---

LOAD - Module Load Method

This parameter determines which macro shall be used for loading a module into a common memory pool.

LOAD=ASHARE	The macro ASHARE will be used.
LOAD=BIND	By default, the macro BIND will be used.

 **Important:** When LOAD=ASHARE is defined, for the start of the common memory load pool (with program CMPSTART), LOAD=ASHARE also must be defined.

ALNK - Activate AUTOLNK Function

This parameter determines whether the AUTOLNK function of the dynamic binder loader (DBL) is activated.

ALNK=YES	The AUTOLNK function is activated.
ALNK=NO	By default, the AUTOLNK function is deactivated.

TIM1 - Wait Time in Seconds before the Load Pool Refresh is Started

This parameter determines the waiting time in seconds before the new Natural nucleus is loaded. It serves to synchronize Natural sessions which are currently active in the nucleus.

TIM1=xx	xx must be in the range from 1 up to 99.
TIM1=10	The default value is 10 seconds.

TIM2 - Wait Time in Seconds after the New Natural Nucleus was Loaded

This parameter determines the waiting time in seconds after the loading of the new Natural nucleus is complete until the serialization identification for the corresponding application has been enabled. It serves to synchronize the relativizing of all address constants in the newly loaded nucleus.

TIM2=xx	xx must be in the range from 1 up to 99.
TIM1=20	The default value is 20 seconds.

17 Optimization of Message Handling

- Screen Output Handling 128
- Restoring the Screen Content 128

Screen Output Handling

Natural provides an extensive message optimization capability. Prior to sending an output screen, Natural determines which portion of the screen has been modified; only data which have actually been modified are sent.

This is to be considered when, between two successive terminal outputs, portions or the entire terminal contents are changed

- by using the `CLEAR` key or
- by intervening dialog steps at system level (K2 interruption or similar interruption).

This is particularly true if a subprogram called from Natural by an external `CALL` interface produces dialog output.

Restoring the Screen Content

In the above-mentioned cases, you can use one of the following methods to cause Natural to restore the most recent terminal screen contents.

- Issue the terminal command `%R`.
- Use the statement `SET CONTROL 'R'`

18 Terminal Types Supported under BS2000

▪ Type 9748	130
▪ 975n Series	130
▪ Type 9763M	131

This document contains information on how Natural supports terminal types in a BS2000 environment.

The following types are supported:

- 974_n
- 975_n
- 976_n
- telex devices

Type 9748

As various terminal types which were all defined as 9750 in PDN are often found in networks, the terminal type can also be modified during a Natural session with the terminal command %T= and thereby be made consistent with the device type currently in use.

975n Series

The various devices of the 975_n series differ considerably (for example, possible number of field separation characters per line, default brightness for protected blank lines, standard arrangement of display characteristics to field properties, etc.).

Four terminal driver routines are provided which support these devices. This permits optimum support of black/white devices of the type 9755 or 9756 with respect to their varying display characteristics. The different devices can be generated in PDN as 975_n.

Some device types cannot be distinguished by an operating system inquiry (SVC 70). Therefore, Natural permits these "logical terminal types" to be associated with various physical device types during generation.

- Under TIAM, this is done with the parameter T975X.
- Under *openUTM*, the parameter TERMN in the PTERM statement for the KDCDEF application generation is used for this purpose.

Type 9763M

Terminals of type 9763M (monochrome) are treated like 9756-type terminals.

19

Function Keys Supported under BS2000

▪ Key Assignment	134
▪ Modes for Key Assignment	134

This document describes the Natural function-key assignments that are supported for terminal devices of the 975*n* series (types 974*n*, 975*n* and 976*n*).

Key Assignment

In Natural, function keys serve to transfer data together with specific command/execution information to a program.

For terminal device types that only support the keys F1 to F5, the programmable P keys (P1 to P20) are used for this purpose. This means that these keys are assigned the function key values PF1 to PF20 (in 3270 terminology).

The identification of the key pressed is made from Natural-loaded key assignments in connection with the send-key code F5. This allows the distinction of similar data types which were sent using DUE1. Using F5, Natural recognizes the function-key resolution and interprets the P-key value as a code. In the other instance the data are transferred to the executing program.

The loading of keys is controlled by terminal commands or from the executing program using SET CONTROL statements.

Modes for Key Assignment

There are three types of modes for key assignment:

Mode	Description
KN	For terminal types 974 <i>n</i> , 9750 - 9755, the literals %K1 to %K20 are assigned to the keys (terminal command %KN or statement SET CONTROL 'KN'). For terminal types 9756, 9758, 976 <i>n</i> , send-key codes F1 to F20 are loaded to the keys P1 to P20.
KO	The literals 01 to 20 and the send-key code F5 are assigned to the keys (terminal command %KO or statement SET CONTROL 'KO').
KS	The literals A to T as well as the send-key code F5 are assigned to the keys (terminal command %KS or statement SET CONTROL 'KS').

In KS mode, a dummy field is generated in the last two terminal positions of each output message. This field is used for receiving and transferring the key value. Prior to data transfer the cursor is moved in this field using the movement functions assigned to the keys.

If an N is specified after the respective terminal command (that is, %KNN, %KON or %KSN), only the corresponding function-key mode is activated, but no values are loaded to the P keys.

For all modes, cursor-position-dependent key processing, according to current assignment, can lead to differing results. For example, the help key, dependent on field assignment, can invoke either the global or local help processing for a particular field. Such functions should be controlled using PF21 to PF23 interpreted keys (F1 to F3).

20 Common Memory Pools

- Global Common Memory Pools 138
- Local Common Memory Pools 142

This document describes the programs that are provided to start and stop global common memory pools in Natural under BS2000 and the macros that enable you to define local (or global) common memory pools in Natural under BS2000

Global Common Memory Pools

The following programs are provided to start and stop global common memory pools in Natural under BS2000:

- [CMPSTART](#)
- [CMPEND](#)



Note: In the following, *vrs* or *vr* represents the relevant product version.

CMPSTART Program

The program `CMPSTART` does the following:

- It starts global common memory pools with its own start task.
- It loads a defined module into a global common memory pool.
- It initializes a global common memory pool.

The keyword parameters `TXTSIZE` and `BPLIST` (see below) are only valid for program `CMPSTART` and when starting a Natural global buffer pool.

The keyword parameters `JV` and `JVSUFEX` (see below) are only valid for program `CMPSTART` when starting a global common memory pool.

All other keyword parameters are identical with the keyword parameters for the macro `ADDON` used for generating the module `BS2STUB`.

The following keyword parameters are available:

`TXTSIZE` | `BPLIST` | `JV` | `JVSUFEX`

TXTSIZE - Buffer-Pool Text-Record Size

This keyword parameter defines the Natural buffer-pool text-record size in KB.

TXTSIZE= <i>xx</i>	Possible values for <i>xx</i> are: 1, 2, 4, 8, 12, 16.
TXTSIZE=4	By default, the Natural buffer pool has a text-record size of 4 KB.

BPLIST - Preload List For Global Buffer Pool

This keyword parameter defines the name of a preload list for a Natural global buffer pool. The defined Natural programs of the preload list will be loaded into the Natural global buffer pool when the first user logs on.

BPLIST= <i>name</i>	See the Natural profile parameter BPLIST.
---------------------	---

JV - Create a Job Variable

This keyword parameter defines whether a job variable shall be created. This job variable enables the status of the common memory pool to be controlled in the job control language.

JV=0	The common memory pool is not ready (in creation mode).
JV=1	The common memory pool is ready (successfully enabled and initialized).

The name of the job variable has 2 parts:

- Part 1 is the name of the common memory pool (operand of keyword parameter NAME)
- Part 2 is the operand of keyword parameter JVSUFFIX (see below).

Logic of Job Variable Navigation:

When the program CMPSTART has started, a check is made whether the job variable is available. If so, the value of the job variable is set to "0". If not, the job variable is cataloged and its value is set to "0". When the common memory pool was successfully enabled and initialized, the value of the job variable is set to "1". When the global common memory pool is terminated, the job variable is erased.

JV=YES	A job variable shall be created.
JV=NO	By default, no job variable will be used.

JVSUFFIX - Suffix of the Job Variable Name

This keyword parameter defines the second part of the job variable name.

JVSUFFIX= <i>xxxxxxxx</i>	Maximally 8 characters for the second part of the job variable name.
JVSUFFIX=.SAG.JV	This is the default value.

Example:

```
NAME=EDT $vrs$ GA,TYPE=EDT,JV=YES,JVSUFFIX=.SAG##JV . . . . .
```

where vrs represents the relevant product version.

The job variable name is EDT vrs GA.SAG##JV.

Operator Commands

These operator commands terminate a global common memory pool:

```
/INTR $tsn$ ,STOP
```

or

```
/INTR $tsn$ ,END
```

This operator command displays the global common memory pool's name, position, address, size and activation time on the console:

```
/INTR $tsn$ ,DPRM
```

This operator command terminates the global common memory pool's start task with a dump:

```
/INTR $tsn$ ,DUMP
```

Examples:

■ To start a global load pool (environment-independent nucleus)

```
/.NATSHRE LOGON
/OPTION DUMP=YES
/SYSFILE SYSDTA=(SYSCMD)
/SYSFILE SYSOUT=LST.NATSHARE
/EXEC (CMPSTART,NAT $vrs$ .MOD)
NAME=NATSHARE,SIZE=2MB,POSI=ABOVE,ADDR=250,SCOP=GLOBAL
PREFIX=YES,ALNK=NO,LIBR=NAT $vrs$ .USER.MOD
/SYSFILE SYSDTA=(PRIMARY)
/LOGOFF
/* NATSHARE IS THE NAME OF THE LINKED NATURAL REENTRANT MODULE. IT IS ALSO THE
/* NAME OF THE COMMON MEMORY POOL. THE ADDRESS OF THE GLOBAL NATURAL LOAD POOL
/* MUST BE DEFINED. THE ADDRESS MUST BE FIXED (PREFIX=YES).
```

where vrs represents the relevant product version.

■ To start a Natural global buffer pool

```
/.BPvrsGA LOGON
/OPTION DUMP=YES
/SYSFILE SYSDTA=(SYSCMD)
/SYSFILE SYSOUT=LST.BPvrsGA
/EXEC (CMPSTART,NATvrs.MOD)
NAME=BPvrsGA,TYPE=NAT,POSI=ABOVE,SIZE=2048KB,SCOP=GLOBAL
/SYSFILE SYSDTA=(PRIMARY)
/LOGOFF
/* FOR A NATURAL BUFFER POOL, THE OPERAND OF PARAMETER "TYPE" MUST BE DEFINED
/* AS 'NAT'.
```

where *vrs* represents the relevant product version.

■ To start a Natural global buffer pool with ESA data space

```
/.BPvrsA LOGON
/OPTION DUMP=YES
/SYSFILE SYSOUT=LST.BPvrsGA
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (CMPSTART,NATvrs.BS2.MOD)
NAME=BPvrsGA,TYPE=NAT,POSI=ABOVE,SIZE=10MB,ADDR=260,DESA=YES
DATA=32MB
/SYSFILE SYSDATA=(PRIMARY)
/LOGOFF N
```

where *vrs* represents the relevant product version.

CMPEND Program

Program **CMPEND** terminates the start tasks for all global common memory pools. The input for **CMPEND** are the names of the global common memory pools.

Example:

```
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (CMPEND,NATvrs.MOD)
NATSHARE,BPvrsGA
/* THE DELIMITER FOR THE DEFINED NAMES IS ' ' OR ', '.
```

where *vrs* represents the relevant product version.

Local Common Memory Pools

The following section describes the macros that enable you to define local (or global) common memory pools in Natural under BS2000:

- [BS2STUB Macro](#)
- [ADDON Macro](#)
- [ADDEND Macro](#)
- [Example of Assembling Macro BS2STUB](#)

BS2STUB Macro

The macro BS2STUB does the following:

- Starts local common memory pools.
- Connects to a defined global common memory pool.
- Loads a defined module into a local common memory pool.
- Loads dynamically called 3GL programs.

The BS2STUB macro has the following parameters:

```
name BS2STUB PARMOD=nn,PROGMOD=xxx
```

name - CSECT Name

<i>name</i>	Specifies the CSECT name. The first three characters must not contain the value NAT.
<i>name</i> BS2STUB	This is the default name.

PARMOD - Application Address Mode and Location

This parameter specifies whether 24 or 31 bit addressing mode is to be used.

PARMOD= <i>nn</i>	Possible values for <i>nn</i> : 24 or 31 (bit).
PARMOD=31	By default, the address mode setting is 31 bit.

PROGMOD - Loading above or below the 16-MB Line

This parameter specifies whether dynamically loaded programs are to be loaded above or below the 16-MB line.

PROGMOD=ANY	ANY means that the module is loaded above or below the 16-MB line. This is the default setting.
PROGMOD=24	24 means that the module is loaded below the 16-MB line.

ADDON Macro

The macro `ADDON` defines a common memory pool in the `ADDON` table of program `BS2STUB`. It contains the following keyword parameters which are also applicable to program `CMPSTART`:

`ACCS` | `ADDR` | `ALNK` | `DATA` | `DESA` | `LIBR` | `LOAD` | `NAME` | `PFX` | `POSI` | `SCOP` | `SIZE` | `STAT` | `TYPE` | `WAIT`

ACCS - Access To Common Memory Pool

This parameter determines how the common memory pool can be accessed.

ACCS=READ	This means the access is read-only (write-protected). To be able to set <code>ACCS=READ</code> , the user ID must be authorized for the <code>BS2000 CSTMP</code> macro in the user catalog (<code>JOIN</code> command with <code>C-M=YES</code>).
ACCS=WRITE	By default, the common memory pool is write-enabled.

ADDR - Size of Common Memory Pool Address

This parameter determines the number of megabytes for the defined address of the common memory pool. The size must be specified. No default value exists.

<code>ADDR=number</code>	<i>number</i> must be ≥ 0 .
--------------------------	----------------------------------

ALNK - Activate AUTOLNK Function

This parameter determines whether the `AUTOLNK` function of the dynamic binder loader (DBL) is activated.

ALNK=NO	The <code>AUTOLNK</code> function is deactivated.
ALNK=YES	By default, the <code>AUTOLNK</code> function is activated.

DATA - Size of Data Space Area

This parameter can be specified in conjunction with the `DESA` parameter and defines the size of the data space area for the buffer pool or swap pool to be started. The following settings are possible:

DATA= <i>nnn</i> MB	Specifies the size of the data space area in megabytes.
DATA= <i>nnn</i> KB	Specifies the size of the data space area in kilobytes.

Using the DATA parameter in the ADDON macro

- To *start* a Natural *local* buffer pool you specify DESA=YES and use this parameter to determine the size of the data space area in megabytes/kilobytes. The size must be specified, because no default value exists.
- To *connect* a Natural *global* buffer pool or a global swap pool, you specify DESA=YES and omit the DATA parameter, because it has been specified for the [CMPSTART Program](#).

Using the DATA parameter for the CMPSTART program

To *start* a Natural *global* buffer pool you specify DESA=YES and use this parameter to determine the size of the data space area in megabytes/kilobytes. The size must be specified, because no default value exists.

DESA - ESA Data Space Area

This parameter must be specified to determine whether or not an ESA data space area is to be created for a Natural buffer pool or a Natural swap pool.

DESA=YES	An ESA data space area is to be created.
DESA=NO	By default, no ESA data space area is to be created.

- An ESA data space is only supported for buffer pools of TYPE=NAT or TYPE=SWP.
- The parameter DESA=YES is relevant only if a global common memory pool (CMPSTART having its own start task) with ESA data space or a local common memory pool (BS2STUB/ADDON) with ESA data space is to be created.
- For the connection (BS2STUB/ADDON) to an existing global common memory pool, the parameter DESA has no significance.



Caution: An ESA data space should be created only for one global common memory pool which has its own start task. The ESA data space will no longer be available when the task that created the ESA data space terminates normally or abnormally.

LIBR - Load Library

This parameter determines from where the defined module is to be loaded. No default value exists. If the operand of parameter LIBR is not defined, only a common memory pool will be enabled (ENAMP+REQMP).

LIBR= <i>library</i>	<i>library</i> is the name of the load library.
LIBR=BLSLIB	The libraries with the link names BLSLIB and BLSLIB01 to BLSLIB99 are to be used.
LIBR=CLASS-4	Module is loaded as subsystem in class 4 memory.

LOAD - Method for Loading a Module into a Common Memory Pool

This parameter determines which macro shall be used for loading a module into a common memory pool.

LOAD=ASHARE	The macro ASHARE will be used. If ASHARE is defined, the operand of parameter PFIX must be YES.
LOAD=BIND	By default, the macro BIND will be used..

NAME - Common Memory Pool/Module Name

This parameter determines the name of the module and/or the name of the common memory pool. The name must be specified. No default value exists.

NAME= <i>name</i>	<i>name</i> is a valid name of common memory pool or module.
-------------------	--

The maximum number of characters in a name is:

8 characters	Module name (name of common memory pool); Natural buffer pool.
16 characters	All other common memory pools.

PFIX - Fixed Address

This parameter determines whether or not the common memory pool's address should be fixed.

PFIX=YES	The common memory pool's address should be fixed.
PFIX=NO	By default, the common memory pool's address should not be fixed.

For a global Natural load pool, this parameter must be set to YES.

POSI - Position Relative to 16-MB Line

This parameter determines the position of the common memory pool, which can be above or below the 16-MB line.

POSI=ABOVE	The common memory pool is to be located above the 16-MB line.
POSI=BELOW	By default, the common memory pool is to be located below the 16-MB line.

SCOP - Scope of Common Memory Pool

This parameter determines the scope of the common memory pool.

SCOP=LOCAL SCOP=GROUP SCOP=GLOBAL	For information on the scopes of a common memory pool, see the description of the ENAMP macro in the BS2000 documentation.
SCOP=GLOBAL	This is the default setting.

SIZE - Size of Common Memory Pool

This parameter specifies the size of the common memory pool in megabytes/kilobytes.

SIZE= <i>n</i> KB SIZE= <i>n</i> MB	Specifies the size of the common memory pool in <i>n</i> kilobytes or <i>n</i> megabytes.
SIZE=1MB	By default, the common memory pool has a size of 1 megabyte.

STAT - Status of Common Memory Pool

This parameter determines the status of the common memory pool.

STAT=GLOBAL	The status of the common memory pool is GLOBAL (started by CMPSTART).
STAT=LOCAL	The status of the common memory pool is LOCAL (started by BS2STUB). By default, the status of the common memory pool is LOCAL.

 **Note:** The STAT parameter will be ignored when the program [CMPSTART](#) runs.

TYPE - Type of Common Memory Pool

This parameter determines the type of the common memory pool. The type must be specified. No default value exists.

TYPE=COM	Natural DCOM pool
TYPE=EDT	Editor buffer pool
TYPE=MON	Natural monitor pool (SYSMON)
TYPE=NAT	Natural buffer pool
TYPE=SRT	Sort buffer pool
TYPE=SWP	Natural swap pool
TYPE=USR	User buffer pool

WAIT - Enabling or Waiting of Common Memory Pool During Application Startup

This parameter determines during startup of an application whether the common memory pool is to be enabled at once or whether the common memory pool is to wait for a request from Natural and is enabled then.

WAIT=YES	The common memory pool is to wait for a request from Natural and is enabled then.
WAIT=NO	By default, the common memory pool is to be enabled at once.



Note: The WAIT parameter will be ignored when the program CMPSTART runs.

ADDEND Macro

The macro ADDEND defines the end of macro ADDON's definitions. There are no parameters for ADDEND.

Example of Assembling Macro BS2STUB

```
BS2STUBA BS2STUB PARMOD=31,PROGMOD=24          31-BIT ADDRESSING MODE,
*                                               LOAD 3GL PROGRAMS BELOW
* +-----+
* I Define the Natural global load pool with Name NATSHARE
* +-----+
*         ADDON NAME=NATSHARE,STAT=GLOBAL
* +-----+
* I Define the Natural local swap pool
* +-----+
*         ADDON NAME=SWAPvrsLA,TYPE=SWP,SIZE=16MB,STAT=LOCAL,POSI=ABOVE
* +-----+
* I Connecting a Natural global buffer pool with ESA data space
* +-----+
*         ADDON NAME=BPvrsGA,TYPE=NAT,STAT=GLOBAL
* +-----+
* I Creating/Connecting a Natural local buffer pool with ESA data space
* +-----+
*         ADDON NAME=BPvrsLA,TYPE=NAT,POSI=ABOVE,SIZE=10MB,
*             STAT=LOCAL,SCOP=LOCAL,DESA=YES,DATA=32MB
*         ADDEND
*         END
```

where *vrs* represents the relevant product version.

21 Calling Dynamically Reloadable 3GL Programs in a Natural Application

▪ Storage Allocation Rule	150
▪ Thread-Creation Rule	150
▪ Address-Mode Dependencies	150

This document contains rules for address mode selection that apply when dynamically reloadable 3GL programs are called in a Natural application.

Storage Allocation Rule

Whether a dynamically reloadable 3GL program is loaded above or below the 16 MB line depends on the keyword parameter `PROGMOD` for macro `BS2STUB`.

Parameter	Explanation
<code>PROGMOD=ANY</code>	The program is loaded above or below the 16 MB line. This depends on the application's address mode and on the possible existence of <code>AMODE</code> or <code>RMODE</code> statements in the 3GL program to be loaded.
<code>PROGMOD=24</code>	The 3GL program is always loaded below the 16 MB line.

Thread-Creation Rule

Whether the Natural user thread is created above or below the 16 MB line depends on the keyword parameters `NUAADDR` for macro `NATUTM` and on `REQMLOC` for the macros `NAMTIAM` and `NAMBS2`.

Address-Mode Dependencies

The following paragraphs give you an overview of which address mode is used in which generation configuration to call dynamically reloadable 3GL programs.

1. **Calling a 3GL program using the ILCS or CRTE interface**
2. **Calling of 3GL programs without using the ILCS or CRTE interface**
3. **Calling of *open*UTM partial programs which are not 31-bit enabled from Natural/UTM driver via `PEND PR`**

1. Calling a 3GL program using the ILCS or CRTE interface

'ILCS=YES' or 'ILCS=CRTE'

Case	The application was generated with	The 3GL program is called with
1	PARMOD=31 or PARMOD=(31, ABOVE)	AMODE=31
2	PARMOD=24	AMODE=24

2. Calling of 3GL programs without using the ILCS or CRTE interface

'ILCS=NO'

Case	The application was generated with	The 3GL program is called with
1	PARMOD=31 The Natural user thread is located above the 16 MB line and the 3GL program is loaded above or below the 16 MB line.	AMODE=31
2	PARMOD=31 The Natural user thread and the 3GL program are located below the 16 MB line.	AMODE=24
3	PARMOD=(31, ABOVE) The Natural user thread is located above the 16 MB line and the 3GL program is loaded above or below the 16 MB line.	AMODE=31
4	PARMOD=(31, ABOVE) The Natural user thread is located below the 16 MB line and the 3GL program is loaded below the 16 MB line.	AMODE=24
	1.	The module BS2GLUE must be located in the same library as the loaded 3GL program, or the load module library for module BS2GLUE must be defined as BLSLIB in the STARTJOB.
	2.	If such a configuration exists in the case of a Natural/UTM application, the keyword parameter KB has to be defined as KB=NO.
5	A Natural/UTM application was generated using PARMOD=31. The Natural user thread is located below or above the 16 MB line and keyword parameter CALLM31 for macro NURENT is defined as CALLM31=YES.	AMODE=31

3. Calling of openUTM Partial Programs which are not 31-bit enabled from Natural openUTM driver via PEND PR

The application was generated using PARMOD=31 and the keyword parameter SWAMODE for macro NATUM is defined as SWAMODE=YES:

Prior to each calling of the *openUTM* KDCS interface, Natural switches back to the 24-bit address mode, and when control is returned to the *openUTM* driver, a switch-back occurs to the 31-bit address mode.

22

Print File/Work File Server NATPWSV2

- Setup 154
- Operation 155

This document describes the print file/work file server NATPWSV2 for the RPC batch server environment under BS2000 that is started via the [RPC Server Front-End](#).

See also *Print File/Work File Server NATPWSV2 Error Messages* in the *Messages and Codes* documentation.

Setup

The print file/work file server NATPWSV2 communicates with the RPC batch server NATFSTB2 by using the forward eventing method.

To setup the print file/work file server, perform the following steps:

- Link the module NATFSTB2 to the Natural nucleus. The module NATFSTB2 *replaces* the program NATWKFB2.
- The module NATPWSV2 must be linked, together with an ADDON parameter definition, for the common memory pool with the new pool type PWK (print file/work file control pool) in the program BS2STUB. This common memory pool must be set up using a defined fixed address, for example:

```
PWKSTUB BS2STUB PARMOD=31,PROGMOD=ANY,UNRES=*DBLOPT
        ADDON NAME=PWK#POOL,          NAME OF CONTROL POOL          -
              TYPE=PWK,              TYPE OF CONTROL POOL          -
              SIZE=1MB,              POOL SIZE IN MB              -
              STAT=LOCAL,            POOL STATUS IS LOCAL          -
              SCOP=GLOBAL,           SCOPE IS GLOBAL               -
              POSI=ABOVE,            POOL POSITION IS ABOVE         -
              ADDR=19,               ADDRESS IS X'1300000'         -
              PFIX=YES,              POOL ADDRESS IS FIXED         -
              ACCS=WRITE              NO POOL PROTECTION           -
        ADDEND
```

The same ADDON parameter definition (except for ADDR=) must be contained in the program BS2STUB which is linked to the front-end part of the RPC batch server.

Example of linking the print file/work file server:

```
/START-PROGRAM $BINDER
START-LLM-CREATION INTERNAL-NAME=PWKSRV, -
SLICE-DEFINITION=*BY-ATTRIBUTES(RESIDENCY-MODE=*YES)
INCLUDE-MODULES NATPWSV,LIB=(NATvrs.MOD)
INCLUDE-MODULES PWKSTUB,LIB=(user-lib)
MODIFY-SYMBOL-ATTRIBUTES ADD-MODE=*31,RES-MODE=*ANY,READ-ONLY=*NO
MODIFY-SYMBOL-VISIBILITY -
        SYMBOL-NAME = *ALL, -
        SYMBOL-TYPE = *DEFINITIONS, -
```

```
VISIBLE = *NO(KEEP-RESOLUTION=*YES)
SAVE-LLM LIB=NATURAL.USER.MOD,ELEMENT=E.NATPWSV2,LOAD-ADDR=X'10000000'
END
```

where:

vrs is the relevant product version of Natural for Mainframes and
user-lib is the user library.

Operation

Data exchange between the print file/work file server and the RPC batch server takes place in the print file/work file control pool (TYPE=PWK).

Starting the Print File/Work File Server

The print file/work file server has to be started *before* the RPC batch server.

The RPC batch server expects the presence of an initialized print file/work file control pool. This initialization occurs when the print file/work file server is started.

A Natural RPC batch server communicates with exactly one print file/work file server and vice versa (TSN1 <=> TSN2).

All print files and work files (link names P01 to P32 and W01 to W32) to be used have to be defined by a FILE command in the print file/work file server's job control.

Example of a start job:

```
/.PWKSRV LOGON
/ER LST.PWKSERVER.
/STEP
/OPTION DUMP=YES
/FILE WORK.W01,LINK=W01
/FILE WORK.W02,LINK=W02
.
.
/FILE PRINT.P01,LINK=P01
/FILE PRINT.P02,LINK=P02
.
.
/EXEC (NATPWSV2,NATURAL.NATvrs.MOD)
/LOGOFF N
```

where *vrs* represents the relevant product version.

Terminating the Print File/Work File Server

The print file/work file server can be terminated by way of P1 eventing, using the program `COMPEND`. The event name for terminating the print file/work file server is the name of the print file/work file control pool.

Example of a print file/work file server termination procedure:

```
/BEGIN-PROCEDURE LOGGING COMMANDS
/ASSIGN-SYSDTA TO=*SYSCMD
/SET-JOB-STEP
/START-PROGRAM FROM-FILE=*MODULE(LIBRARY=NATURAL.NAT $vrs$ .MOD,-
/      ELEMENT=COMPEND)
PWK#POOL      <== name of the print file/work file control pool
/SET-JOB-STEP
/ASSIGN-SYSDTA TO=*PRIMARY
/EXIT-PROCEDURE
```

where vrs represents the relevant product version.

All error messages (abnormal termination of the print file/work file server) are written to SYSLST99 into the file `LST.PWKSERVER.tsnn`.

23 Using Network Attached Storage Files as Natural Print and Work Files

You can allocate and process files contained on remote Network Attached Storage (NAS) servers as Natural print files and work files. Prerequisite: BS2000/OSD V9.0 (or above) must be installed at your site.

Starting with BS2000/OSD V9.0, Net Storage supports access to Network Attached Storage (NAS) servers over the BS2000 Network File System (NFS). Since NFS access is transparently mapped to the access methods of the BS2000 data management system, files on Net Storage volumes can be processed with the Natural print/work file access method in the same way as files on local disk storage.

For more information on Net Storage and NFS, refer to the appropriate BS2000 manuals from Fujitsu.

The following are JCL examples for allocating a Natural work file on an NAS volume mounted to a Pubset named NFS9.

Example of Work File Allocation - ISP Format:

```
/FILE :NFS9:$USERID.NETSTORAGE.WORK.01,LINK=W01,DEVICE=NETSTOR,-  
/VOLUME=NET000  
...
```

Example of Work File Allocation - SDF Format:

```
/CREATE-FILE FILE-NAME=-  
/:NFS9:$USERID.NETSTORAGE.WORK.01,SUPPORT=PRIVATE-DISK(VOLUME=NET000,-  
/DEVICE-TYPE=NETSTOR,SPACE=STD),SUPPRESS-ERRORS=*FILE-EXISTING  
...  
/ADD-FILE-LINK LINK-NAME=W01,FILE-NAME=-  
/:NFS9:$USERID.NETSTORAGE.WORK.01  
...
```

24

RPC Server Front-End

- Setup 160

This document describes how to set up the RPC server front-end for an RPC batch server environment under BS2000 with the print file/work file server [NATPWSV2](#).

Setup

For the generation of the Natural RPC batch server, the non-reentrant part of the Natural batch driver (macro `NAMBS2`) has to be assembled with the new keyword parameter `SERVER=YES`.

Example:

```

SERVFRNT NAMBS2  CODE=FRONT,           -
                  APPLNAM=NATSERV,      -
                  NUCNAME=RPCSERV,      -
                  DYNPAR=SYSDTA,         For server parameters -
                  SERVER=YES,           Generate RPC server   -
                  ROLLTSZ=384,          Roll thread size in KB -
                  .                      All other parameter definitions as for
                  .                      the generation of the non-reentrant part
                  .                      of the Natural batch driver
                  END

```

For the generation of the reentrant part of the Natural RPC batch server, you can use the same keyword parameter definitions as for the generation of the Natural batch driver.

For the generation of the module `BS2STUB` (front-end part of the RPC batch server), you have to define the necessary common memory pools. If you intend to use the print file/work file server [NATPWSV2](#), define a print file/work file control pool and replace the module `NATWKFB2` with the module `NATFSTB2` in the Natural reentrant part.

```

RPCSTUB2 BS2STUB PARMOD031,PROGMOD=ANY,UNRES=*DBLOPT
ADDON NAME=RPCSERV,           Name of reentrant part (load pool)   -
    .
    .
    .
ADDON definition for Natural Buffer Pool
    .
ADDON definition for Natural Editor Pool
    .
ADDON definition for Natural Swap Pool
    .
ADDON NAME = PWK#POOL,        Name of print file/work file control pool -
    TYPE=PWK,                 Pool type                               -
    SIZE=1MB,                 Pool size in MB                           -
    STAT=LOCAL,               Pool status is local                       -
    SCOP=GLOBAL,              Scope is global                             -
    POSI=ABOVE,               Pool position is above                     -
    PFIX=YES,                  Pool address is fixed                       -

```

```

        ACCS=WRITE           No pool protection
ADDEND

```

The Natural RPC batch server stores the different client context in user threads. These user threads are managed either in the swap pool or in the Natural roll file. Hence, a Natural roll file and a Natural swap pool is required.

For the processing of print files and work files, a print file/work file server has to be generated (see [Print File/Work File Server NATPWSV2](#)), using the new type "PWK".

Data interchange between RPC batch server and print file/work file server takes place in a common memory pool (print file/work file control pool), using the new type "PWK".

Communication between RPC batch server (module NATFSTB2) and print file/work file server (module NATPWSV2) is accomplished by way of P1 forward eventing. If you intend to work with the print file/work file server, then the module NATWKFB2 has to be replaced by the module NATFSTB2 in the link job for the reentrant part.

Example of linking the environment-dependent nucleus for the RPC batch server:

```

/START-PROGRAM $BINDER
START-LLM-CREATION INTERNAL-NAME=SERVER,           -
SLICE-DEFINITION=*BY-ATTRIBUTES(RESIDENCY-MODE=*YES)
INCLUDE-MODULES NATSFED2,LIB=(NATvrs.MOD)  Must be bound as first module
INCLUDE-MODULES RPCSFE,LIB=(NATvrs.MOD)    RPC front-end stub
INCLUDE-MODULES SERVFRNT,LIB=(user-lib)    Non-reentrant part of Natural batch
INCLUDE-MODULES SERVRENT,LIB=(user-lib)    Reentrant part of Natural batch
INCLUDE-MODULES RPCSTUB2,LIB=(user-lib)    BS2STUB (see previous example) ←

INCLUDE-MODULES SWPRMSRV,LIB=(user-lib)    Swap pool parameter module
INCLUDE-MODULES NATPRMSV,LIB=(user-lib)    Natural parameter module
INCLUDE-MODULES ADAUSER,LIB=(ADAvrs.MOD)
INCLUDE-MODULES SSFB2C,LIB=(ADAvrs.MOD)
MODIFY-SYMBOL-ATTRIBUTES ADD-MODE=*31,RES-MODE=*ANY,READ-ONLY=*NO
MODIFY-SYMBOL-VISIBILITY                    -
        SYMBOL-NAME = *ALL,                  -
        SYMBOL-TYPE = *DEFINITIONS,         -
        VISIBLE = *NO(KEEP-RESOLUTION=*YES)
SAVE-LLM LIB=NATURAL.USER.MOD,LOAD-ADDR=X'10000000',ELEMENT=BATCH.SERVER
END

```

where:

vrs is the relevant product version of Natural for Mainframes or Adabas for Mainframes and

user-lib is the user library.

For information on how to generate the swap pool parameter module, refer to the section [Defining the Natural Swap Pool, Keyword Parameters of Macro NTSWPRM](#).

Example of linking the environment-independent nucleus for the RPC batch server:

```

/START-PROGRAM $BINDER
START-LLM-CREATION INTERNAL-NAME=RCPSERV, -
SLICE-DEFINITION=*BY-ATTRIBUTES(RESIDENCY-MODE=*YES)
INCLUDE-MODULES NATINV,LIB=(NATvrs.MOD) Must be bound as first module
INCLUDE-MODULES NATURAL,LIB=(NATvrs.MOD) Natural nucleus
REMARK INCLUDE-MODULES NATWKFB2,LIB=(NATvrs.MOD) Is replaced by
INCLUDE-MODULES NAFSTB2,LIB=(NATvrs.MOD) print/work file server stub
. ALL OTHER
. MODULES
INCLUDE-MODULES NATLAST,LIB=(NATvrs.MOD)
MODIFY-SYMBOL-ATTRIBUTES ADD-MODE=*ANY,RES-MODE=*ANY,READ-ONLY=*NO
MODIFY-SYMBOL-VISIBILITY -
SYMBOL-NAME = *ALL, -
SYMBOL-TYPE = *DEFINITIONS, -
VISIBLE = *NO(KEEP-RESOLUTION=*YES)
SAVE-LLM LIB=NATURAL.USER.MOD
END

```

where *vrs* represents the relevant product version of Natural for Mainframes.

Example of parameters for the Natural batch server:

```

AUTO=ON,
STACK=(LOGON DFSERVER),
RPC=(
SERVER=ON,
SRVNODE='10.20.91.202:3860:TCP',SRVNAME=DFSRV1,
RPCSIZE=128,MAXBUFF=30,
TRACE=2
),
RCA=BROKER,RCALIAS=(BROKER,BKIMBTIA),
MADIO=0,MAXCL=0,MT=0,MENU=OFF,
PRINT=((10),AM=STD),WORK=((1-10),AM=STD)

```

Example of parameters for the Natural RPC server client:

```

STACK=(LOGON DFCLIENT),
RPC=(
DFS=(DFSRV1,BKR043,,,NOSERVDIR),
RPCSIZE=128,MAXBUFF=30
),
RCA=BROKER,RCALIAS=(BROKER,BKIMBTIA),
MADIO=0,MAXCL=0,MT=0,ETID=' '

```

The Natural RPC batch server requires the file named P10 for the output of server messages. If the print file/work file server is used, this file has to be defined using the FILE instruction in the job control for the print file/work file server, unless it is defined in the job control of the Natural RPC batch server.

Example of a start job:

```
/.SERVER LOGON
/SYSFILE SYSOUT=SERVER.OUT
/SYSFILE SYSLST=SERVER.LIST
/FILE NAT $vrs$ .EDIT.WORKFILE, LINK=CMEDIT
/FILE NAT $vrs$ .SERVER.ROLLFILE, LINK=PAMNAT, SHARUPD=YES
/FILE SERVER.MSG, LINK=P10           Is required for the server messages
/FILE ADA $vrs$ .MOD, LINK=DDLIB
/FILE ADAPARM, LINK=DDLNKPAR
/FILE EXX $vrs$ .LIB, LINK=BLSLIB01      Broker
/FILE EXX $vrs$ .LIB, LINK=ETBLIB       Load library
/SYSFILE SYSDTA=SERVERPARMS
/EXEC BATCH.SERVER
/LOGOFF N
```

where vrs represents the relevant product version of Natural for Mainframes, Adabas for Mainframes or EntireX Communicator.

For information on how to generate and start the EntireX Broker, refer to the EntireX Communicator documentation.

VI Natural in Batch Mode

This part contains considerations that apply when running Natural in batch mode.

- Natural in Batch Mode under z/OS** Provides special considerations that refer to Natural in batch mode under the operating system z/OS.
- Natural in Batch Mode under z/VSE** Provides special considerations that refer to Natural in batch mode under the operating system z/VSE.
- Natural in Batch Mode under BS2000** Provides special considerations that refer to Natural in batch mode under the operating system BS2000.
- Natural in Batch Mode (All Environments)** Contains general considerations that apply when running Natural in batch: Adabas data sets, sort data sets, subtasking session support for batch environments.

See also *Batch Mode* in the section *Profile Parameters Grouped by Category (Parameter Reference documentation)* for an overview of the Natural profile parameters that apply if Natural is used in batch mode.

25 Natural in Batch Mode under z/OS

▪ Natural z/OS Batch Interface	168
▪ Driver Parameters for z/OS Batch	168
▪ Data Sets Used by Natural in z/OS Batch Mode	168

This document contains special considerations that refer to Natural in batch mode under the operating system z/OS.

For considerations that refer to Natural in batch mode generally, see also:

- [Adabas Data Sets](#)
- [Sort Data Sets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

Natural z/OS Batch Interface

The Natural z/OS batch interface consists of the NATOS object module which is linked to the Natural nucleus during the installation procedure for base Natural as described in the *Installation for z/OS* documentation.

You can customize the Natural z/OS batch interface to meet your requirements by changing the parameter settings in the NTOSP macro in the Natural parameter module during the appropriate installation step.

NATOS is fully reentrant and can run above the 16 MB line. Multiple Natural sessions can be started in parallel within one batch region; see [Subtasking Session Support for Batch Environments](#).

Driver Parameters for z/OS Batch

For information on the driver parameters that are available for z/OS in batch mode, refer to the description of profile parameter OSP or parameter macro NTOSP in the *Parameter Reference* documentation.

Data Sets Used by Natural in z/OS Batch Mode

The following data sets are required if certain functions are used during a Natural z/OS batch mode session:

Data Set	Explanation
CMEDIT	Software AG Editor Work File
CMHCOPY	Hardcopy Print Output
CMOBJIN	Input for Natural INPUT Statements
CMPLOG	Dynamic Profile Parameter Report Output
CMPRINT	Primary Report Output
CMPRMIN	Dynamic Profile Parameter Input
CMPRT nn	Additional Reports 01-31
CMSYNIN	Primary Command Input
CMTRACE	External Trace Output
NATRJE	Job Submit Output
STEPLIB	Load Library for External Modules
CMWKF nn	Work Files 01-32

These data sets are described below.

For sequential data output sets, the default DCB RECFM/LRECL information is as follows:

RECFM=FBA and LRECL=133

CMEDIT - Software AG Editor Work File

The Software AG Editor work file VSAM data set is required if a local or global Software AG editor buffer pool is to be used.

If not defined in the JCL, the name of the Editor work file specified by subparameter DSNAME of profile parameter EDBP or parameter macro NTEDBP is used by Natural to do the dynamic allocation for the Editor work file.

Alternatively, profile parameter EDPSIZE can be used to run with an auxiliary editor buffer pool, which doesn't require an editor work file. For more information about the installation of the Software AG editor, refer to *Installing the Software AG Editor on z/OS* in the *Installation for z/OS* documentation.

CMHCOPY - Optional Report Output for Hardcopy

The default name of the hardcopy print output data set is CMHCOPY. It can be changed by one of the following:

- the subparameter DEST of profile parameter PRINT for Print File 0,
- the profile parameter HCDEST, which is an equivalent of PRINT=((0),DEST=...),
- the setting of the system variable *HARDCOPY during the session,
- the terminal command %H during the session.

The subparameters of the profile parameter PRINT for Print File 0 can be used to change the default values for the hardcopy data set. The default data set name CMHCOPY implies CLOSE=FIN for the hardcopy print data set, that is, after the data set has been opened for output, any subsequent change of the hardcopy print output data set name will not be honored. If a different name is defined at open time, the hardcopy data set will be closed according to subparameter CLOSE of profile parameter PRINT for Print File 0.

During the session, the hardcopy data set can be released and reallocated (before open or after close) by the by dynamic allocation (via application programming interface USR2021N, see *SYSEXT - Natural Application Programming Interfaces*).

CMOBJIN - Input for Natural INPUT Statements

This data set can be used to read data by the Natural INPUT statement rather than from the primary input data set CMSYNIN.

The usage of CMOBJIN is controlled by the profile parameter OBJIN. The input record data length for Natural is determined by profile parameter SL. The maximum record length (LRECL) supported is 255. The record format (RECFM) can be fixed or variable.

CMPLOG - Dynamic Profile Parameter Report Output

If profile parameter PLOG=ON is set and data set CMPLOG is available, the evaluated dynamic profile parameters are written to this data set during session initialization. If data set CMPLOG is not available, the evaluated dynamic profile parameters are written to CMPRINT.

CMPRINT - Primary Report Output

CMPRINT is used for the primary output report resulting from DISPLAY, PRINT and WRITE statements in a Natural program.

The record format (RECFM) for CMPRINT is FBA. If no DCB information for LRECL is available from the data set or from the JCL, LRECL=133 will be used as default. If no DCB information for BLKSIZE is available from the data set or from the JCL, BLKSIZE will be 10 times the value of LRECL as default.

If not defined in JCL, CMPRINT will be allocated dynamically as

```
//CMPRINT DD SYSOUT=*
```

when the first record is to be written.

CMPRMIN - Dynamic Parameter Data Set

CMPRMIN can be used as a dynamic parameter data set to overcome the length restriction for the character string in the job control PARM keyword of the EXEC statement.

If available, this file is read during session initialization to get the dynamic profile parameters.

All input records from CMPRMIN are concatenated into one parameter string. Only the first 72 positions of each CMPRMIN record are significant. Trailing blanks at the end of each record are truncated; if the last non-blank character is a comma, all trailing blanks are truncated, otherwise just one blank is left as delimiter; no commas are inserted.

Additional dynamic parameters can be supplied using the job control PARM keyword. They are concatenated at the end of the parameter string which was built from the input of CMPRMIN, that is, these can be used to overwrite the parameters from CMPRMIN.

CMPRTnn - Additional Reports 01 - 31

These data sets can be used by Natural print file statements like WRITE (*nn*). If no DCB information (for example, RECFM, LRECL, BLKSIZE) is available, the defaults are defined by the PRINT profile parameter or the NTPRINT macro in the Natural parameter module. The print file names can be overwritten by subparameter DEST.

CMSYNIN - Primary Command Input

This data set is used to read command input and data requested by the Natural INPUT statement. The latter is controlled by the profile parameter OBJIN (see also CMOBJIN).

The input record data length for Natural is determined by profile parameter SL. The maximum record length (LRECL) supported is 255. The record format (RECFM) can be fixed or variable.

CMTRACE - Optional Report Output for Natural Tracing

If profile parameter ETRACE=ON is set or the equivalent terminal command %TRE+ was issued, any Natural trace output during the session is written to the CMTRACE data set. To define the Natural components that are to be traced, the profile parameter TRACE is required.

If data set CMTRACE is not available, it will be allocated dynamically as

```
//CMTRACE DD SYSOUT=*
```

when the first trace record is to be written.

NATRJE - Job Submit Output

This data set is used for the Natural job submitting utility. If it is not defined, it will be allocated dynamically as

```
//NATRJE DD SYSOUT=(A,INTRDR)
```

when the first job is submitted.

STEPLIB - Load Library for External Modules

STEPLIB is the default load library name for loading external modules, for example:

- the environment-independent nucleus (profile parameter NUCNAME),
- a separate Adabas link routine module (profile parameter ADANAME),
- the session back-end program (profile parameter PROGRAM),
- any external subprograms not linked to the [Natural parameter module](#).

The load library name can be changed by profile parameter LIBNAM. The specified load library name must be defined by a DD statement in the JCL.

CMWKFnn - Work Files 01-32

These data sets can be used by Natural work file statements like `READ WORK nn` and `WRITE WORK nn`.

If no DCB information (`RECFM`, `LRECL`, `BLKSIZE`, etc.) is available in the JCL or in the VTOC entry for the data set, the defaults are defined by the `WORK` profile parameter or the `NTWORK` macro in the **Natural parameter module**.

The work file data set names can be overwritten by subparameter `DEST`.

26 Natural in Batch Mode under z/VSE

▪ Natural z/VSE Batch Interface	176
▪ Driver Parameters for z/VSE Batch	176
▪ Natural Data Sets Used under a z/VSE Batch Mode Session	176
▪ NATVSE Print and Work File Support for z/VSE Library Members	182
▪ NATVSE Print File Support for Direct POWER SPOOL Access	183
▪ NATVSE Dynamic Work File Allocation (DYNALLOC) Support	184
▪ Debugging Facilities for Natural under z/VSE	188
▪ NATVSE Attention Interrupts	191

This document contains special considerations that refer to Natural in batch mode under the operating system z/VSE.

For considerations that refer to Natural in batch mode generally, see also:

- [Adabas Data Sets](#)
- [Sort Data Sets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

Natural z/VSE Batch Interface

The Natural z/VSE batch interface consists of the object modules NATVSE, NATVSEL and NATLEOPT. They are linked to the Natural nucleus during the installation procedure for base Natural as described in the *Installation for z/VSE* documentation.

You can customize the Natural z/VSE batch interface to meet your requirements by changing the parameter settings in the NTVSEP macro in the Natural parameter module during the appropriate installation step.

The batch interface must run below the 16 MB line. Multiple sessions can be started in parallel within one batch region; see [Subtasking Session Support for Batch Mode Environments](#).

Driver Parameters for z/VSE Batch

For information on the driver parameters that are available for z/VSE in batch mode, refer to the description of profile parameter VSEP or parameter macro NTVSEP in the *Parameter Reference* documentation.

Natural Data Sets Used under a z/VSE Batch Mode Session

The following data sets are required if certain functions are used during a Natural z/VSE batch mode session:

Data Set	Explanation
CMEDIT	Software AG Editor Work File
CMHCOPY	Hardcopy Print Output
CMOBJIN	Input for Natural INPUT Statements
CMPLOG	Dynamic Profile Parameter Report Output
CMPRINT	Primary Report Output
CMPRMIN	Dynamic Profile Parameter Input
CMPRTnn	Additional Reports 01-31
CMSYNIN	Primary Command Input
CMTRACE	External Trace Output
CMWKFnn	Work Files 01-32

These data sets are described below.

CMEDIT - Software AG Editor Work File

The Software AG Editor work file VSAM data set is required if a local or global Software AG editor buffer pool is to be used.

If not defined in the JCL, the name of the Editor work file specified by subparameter `DSNAME` of profile parameter `EDBP` or parameter macro `NTEDBP` is used by Natural to do the dynamic allocation for the Editor work file.

Alternatively, profile parameter `EDPSIZE` can be used to run with an auxiliary editor buffer pool, which does not require an editor work file. For more information about the installation of the Software AG editor, see *Installing the Software AG Editor on z/VSE* in the *Installation for z/VSE* documentation.

CMHCOPY - Optional Report Output for Hardcopy

The default name of the hardcopy print output data set is `CMHCOPY`. It can be changed by one of the following:

- the subparameter `DEST` of profile parameter `PRINT` for Print File 0,
- the profile parameter `HCDEST`, which is an equivalent of `PRINT=((0),DEST=...)`,
- the setting of the system variable `*HARDCOPY` during the session,
- the terminal command `%H` during the session.

The subparameters of the profile parameter `PRINT` for Print File 0 can be used to change the default values for the hardcopy data set. The default data set name `CMHCOPY` implies `CLOSE=FIN` for the hardcopy print data set, that is, after the data set has been opened for output, any subsequent change of the hardcopy print output data set name will not be honored. If a different name is

defined at open time, the hardcopy data set will be closed according to subparameter `CLOSE` of profile parameter `PRINT` for Print File 0.

By default, the `CMHCOPY` file is assigned to `SYSLST` and is processed via the macro `DTFPR`.

If appropriate label information is supplied for the file name `CMHCOPY`, the print output may also be routed to disk or tape by using the z/VSE macro `DTFSD` or `DTFMT` with:

```
RECFORM=UNDEF, BLKSIZE=133
```

When routed to a z/VSE library, the record format is `fix`, the record length is 80 and the default member type is `PRINT`.

CMOBJIN - Input for Natural INPUT Statements

`CMOBJIN` is used for data intended to be read by Natural `INPUT` statements. This type of data can alternatively be placed in the `CMSYNIN` input stream immediately following the relevant source program or the relevant `RUN` or `EXEC` command.

When the setting for the profile parameter `OBJIN` is `N`, Natural reads input from `CMSYNIN`. When `OBJIN` is set to `Y`, Natural reads input from `CMOBJIN`. When `OBJIN` is set to `R`, Natural determines which option has been selected for a particular session depending upon the presence or absence of a `CMOBJIN` label information.

By default, the `CMOBJIN` input file is assigned to `SYSIPT`. By using the profile parameter `READER`, it can be assigned to `SYSRDR`.

Alternatively, a sequential disk or labeled tape file may be used rather than a real/logical (`POWER`) reader file. In that case, you must supply appropriate label information for file name `CMOBJIN`.

Supported file formats are:

```
DTFSD/DTFMT: RECFORM=FIXUNB, RECSIZE=81
DTFSD/DTFMT: RECFORM=FIXUNB/FIXBLK, RECSIZE=80
LIBR:         RECFORM=FIX, RECSIZE=80 , default member type CARD
```

You must supply appropriate label information; for assignment, you have to use file names `CMSYNIN` and /or `CMOBJIN`.

CMPLOG - Optional Report Output for Dynamic Parameters

If profile parameter `PLOG=ON` is set and data set `CMPLOG` is available, the evaluated dynamic profile parameters are written to this data set during session initialization. If data set `CMPLOG` is not available, the evaluated dynamic profile parameters are written to `CMPRINT`.

By default, the `CMPLOG` file is assigned to `SYSLST` and is processed with the macro `DTFPR`.

If appropriate label information is supplied for the file name `CMPLOG`, the print output may also be routed to disk or tape by using the z/VSE macros `DTFSD` or `DTFMT` with:

```
RECFORM=UNDEF ,BLKSIZE=133
```

When routed to a z/VSE library, the record format is `fix`, the record length is 80 and the default member type is `PRINT`.

CMPRINT - Primary Report Output

`CMPRINT` is used for the primary output report resulting from `DISPLAY`, `PRINT` and `WRITE` statements in a Natural program.

By default, the `CMPRINT` file is assigned to `SYSLST` and is processed via the macro `DTFPR`.

If appropriate label information is supplied for the file name `CMPRINT`, the print output may also be routed to disk or tape by using the z/VSE macro `DTFSD` or `DTFMT` with:

```
RECFORM=UNDEF ,BLKSIZE=133
```

When routed to a z/VSE library, the record format is `fix`, the record length is 80 and the default file type is `PRINT`.

CMPRMIN - Dynamic Parameter Data Set

`CMPRMIN` can be used as a dynamic parameter data set to overcome the length restriction for the character string in the job control `PARM` keyword of the `EXEC` statement.

If available, this file is read during session initialization to get the dynamic profile parameters.

All input records from `CMPRMIN` are concatenated into one parameter string. Only the first 72 positions of each `CMPRMIN` record are significant. Trailing blanks at the end of each record are truncated; if the last non-blank character is a comma, all trailing blanks are truncated, else just one blank is left for delimiter; no commas are inserted.

Additional dynamic parameters can be supplied using the job control `PARM` keyword: if the `PARM` keyword contains a dynamic parameter string, these profile parameters are concatenated at the end of the parameter string which was built from the input of `CMPRMIN`, i.e. these can be used to overwrite the parameters from `CMPRMIN`. If the `PARM` keyword is specified as `SYSRDR` or `SYSIPT`,

Natural retrieves additional profile parameters from `SYSRDR` or `SYSIPT` respectively as a logical extension of the `CMPRMIN` data set, i.e. the same rules apply.

`CMPRMIN` is a sequential disk or a labeled tape data set. Supported file formats are:

```
DTFSD/DTFMT: RECFORM=FIXUNB,RECSIZE=81
DTFSD/DTFMT: RECFORM=FIXUNB/FIXBLK,RECSIZE=80
LIBR:         RECFORM=FIX,RECSIZE=80 , default member type CARD
```

CMPR T_{nn} - Additional Reports

`CMPR T_{nn}` is used for each additional report referenced by any Natural program compiled or executed during the session. " nn " must be a two-digit decimal number in the range 01-31 corresponding to the report number used in a `DISPLAY`, `PRINT` or `WRITE` statement.

Instead of `CMPR T_{nn}` , another file name may be used by setting the `DEST` subparameter of profile parameter `PRINT` to an appropriate value, for example:

```
PRINT=(( $nn$ ),...,DEST=PRNTFIL)
```

When supplying label information with file name `CMPR T_{nn}` , the print output can be written to a disk or tape. Natural treats this print file like an unblocked fixed-length work file. When "printing" to disk or tape, the same logic as for work files applies (see [below](#)).

When mapped to a z/VSE library member, the record format is `fix`, the record length is 80 and the default file type for these files is `PRINT`.

CMSYNIN - Primary Input

`CMSYNIN` is used for the primary input file that contains Natural commands, Natural source programs, and (optionally) data to be read by `INPUT` statements during the execution of Natural programs.

By default, the `CMSYNIN` input file is assigned to `SYSRDR`. By using the profile parameter `READER`, it may be assigned to `SYSIPT`.

Alternatively, a sequential disk or labeled tape file may be used rather than a real/logical (`POWER`) reader file. In that case, you must supply appropriate label information for file name `CMSYNIN`.

Supported file formats are:

```
DTFSD/DTFMT: RECFORM=FIXUNB,RECSIZE=81
DTFSD/DTFMT: RECFORM=FIXUNB/FIXBLK,RECSIZE=80
LIBR:         RECFORM=FIX,RECSIZE=80 , default member type CARD
```

CMTRACE - Optional Report Output for Natural Tracing

If profile parameter `ETRACE` is set to "ON" or the equivalent terminal command `%TRE+` was issued, any Natural trace output during the session is written to the `CMTRACE` data set. To define the Natural components that are to be traced, the profile parameter `TRACE` is required.

By default, the `CMTRACE` file is assigned to `SYSLST` and is processed via the macro `DTFPR`.

If appropriate label information is supplied for the file name `CMTRACE`, the print output may also be routed to disk or tape by using the z/VSE macro `DTFSD` or `DTFMT` with:

```
RECFORM=UNDEF, BLKSIZE=133
```

When routed to a z/VSE library, the record format is `fix`, the record length is 80 and the default member type is `PRINT`.

CMWKFnn - Work Files 01-32

`CMWKFnn` is used for each Natural work file referenced by any Natural program compiled or executed during the session. `nn` must be a two-digit decimal number in the range 01 - 32 corresponding to the number used in a `READ WORK FILE` or `WRITE WORK FILE` statement.

Instead of `CMWKFnn`, another file name may be used by setting the `DEST` subparameter of profile parameter `WORK` to an appropriate value.

If the subparameter `FILMNGR` of profile parameter `VSEP` or macro `NTVSEP` is set to `YES`, and there is label information for a work file, or if `OFF` or `NOTM` is specified for the `LABEL` subparameter of profile parameter `WORK` for an unlabeled work file, Natural knows the file is available. Otherwise, the Natural work-file logical-unit number must be assigned to the correct device type.

When mapped to a z/VSE library member, the record format is `fix`, the record length is 80 and the default member type for these files is `WORK`.

If a Natural printer or work file is assigned `IGN`, all I/O requests for these files are treated as dummy and no Natural error is generated. However, if there is no assignment or the printer/work file is assigned `UA`, any attempt to use this file is treated as an error.

NATVSE Print and Work File Support for z/VSE Library Members

NATVSE supports access to z/VSE library members for input and/or output for all Natural data sets. When a z/VSE library member is accessed, only “card image format” is supported, that is, a record length of 80 bytes.

The access to a z/VSE library member is triggered via the file ID of an associated DLBL statement. A special string (see subparameter LIBRID of profile parameter VSEP or macro NTVSEP) at the start of the file ID field in the DLBL statement signals that the Natural data set actually is a z/VSE library member which is specified in the remainder of the file ID field.

The following specifications are possible:

<i>C=chain</i>	Specifies a library concatenation chain defined in JCL.
<i>S=library.sublib</i>	Specifies a specific sublibrary in a specific library.
<i>M=mbrname.mbrtype</i>	Specifies a library member name and its type.

The following rules apply:

- All these possible specifications are optional.
- Each parameter may be specified only once.
- The parameters are separated by one or more commas or blanks.
- Chain (C=) and sublibrary (S=) specifications are optional, but mutually exclusive when specified.
- If neither a chain (C=) nor a sublibrary (S=) is specified, a default of C=SOURCE is taken.
- If a library member (M=) is not specified, a default of M=*filename.type* is taken, where *filename* is the file name of the DLBL statement and *type* indicates the Natural file class, namely WORK for Natural work files, PRINT for Natural print files and CARD for the Natural input files CMPRMIN, CMSYNIN and CMOBJIN (the relevant default member type for every Natural data set is mentioned below).
- An asterisk specified for any sub-parameter of the library member specification signals the default to be taken; hence a specification of M=*. * has the same effect as omitting this parameter.
- Omitting the member type subparameter also means the default to be taken.

Example:

```
// LIBDEF PROC,SEARCH=(...)
// LIBDEF SOURCE,SEARCH=(...)
// DLBL CMWKF01,'LIBR:M=FILE1.TEST S=SAGLIB.USRLIB'
// DLBL CMWKF02,'LIBR:          S=SAGLIB.USRLIB'         -> M=CMWKF02.WORK
// DLBL CMWKF03,'LIBR: M=TEST   C=PROC'                 -> M=TEST.WORK
// DLBL CMPRT04,'LIBR:M=* .LISTING,S=SAGLIB.USRLIB'     -> M=CMPRT04.LISTING
// DLBL CMPRT05,'LIBR:'                                 -> M=CMPRT05.PRINT,C=SOURCE
// DLBL CMPRT06,'LIBR:M=WORK'                           -> M=WORK.PRINT,C=SOURCE
// DLBL CMWKF07,'LIBR:          M=* .DATA'              -> M=CMWKF07.DATA,C=SOURCE
// DLBL CMPRMIN,'LIBR:M=* .*'                            -> M=CMPRMIN.CARD,C=SOURCE
```



Notes:

1. When a chain is specified or defaulted for an output file, the output is written into the first sublibrary specified in the chain.
2. If a member with the same name and type already exists in a target sublibrary of a Natural output file, this member is replaced unconditionally.
3. The file ID field of a DLBL statement is just 44 characters in length, which is not enough to specify all (sub)parameters in their full length. Therefore it is recommended to take advantage of the defaults. Regarding the member name, there is also the option to specify the file name via the DEST subparameter of the Natural profile parameter PRINT or WORK.

NATVSE Print File Support for Direct POWER SPOOL Access

Natural supports up to 35 print files per session, namely the system print files [CMPRINT](#), [CMTRACE](#), [CMPLOG](#) and [CMHCOPY](#), plus “application” print files 1 to 31. POWER on the other side just supports up to 8 logical printers per VSE partition. This means, it is not possible to create more than 8 separate reports in parallel from a Natural program in one partition. This bottleneck becomes worse in multi-tasking or Natural server environments, where up to 32 Natural tasks may run in parallel in one VSE partition. That limitation can be overcome by accessing POWER SPOOL directly; that is, instead of LIOCS services, NATVSE uses “PUTSPOOL” services to create reports in the POWER list queue.

The creation of a report directly in POWER SPOOL is triggered via the file ID of an associated DLBL statement or a corresponding specification in a Natural DEFINE PRINTER statement. A special string (see subparameter SPOOLID of profile parameter VSEP or macro NTVSEP) at the start of the file ID field in the DLBL statement signals that the Natural print file is not created by LIOCS services, but by “PUTSPOOL” services.

For such print files, the keyword subparameters NAME, FORMS, DISP, COPIES, CLASS, PRTY of the profile parameter PRINT or the macro NTPRINT in the [Natural parameter module](#) are sensitive; they can also be set via a DEFINE PRINTER statement. When a report name is not specified, the print file name (DEST=) is the default. For print files, where DEFINE PRINTER and the (NT)PRINT

profile do not apply, that is, CMPRINT, CMTRACE and CMPLOG, the POWER list attributes can be specified in the associated DLBL statement via keyword parameters, with the following meanings:

C	Class
D	Disposition
F	Forms
N	Name
O	Copies
P	Prty

Example:

```
// DLBL CMPLOG, 'SPL2PWR N=PARMSLOG,C=K,D=L,O=3,F=GREY'
// DLBL CMPRT01, 'SPL2PWR'
// ASSGN SYS041,UA
// ASSGN SYS042,UA
...
PRINT=((1-2),AM=STD)
PRINT=((1),NAME=REPORT1,CLASS=K,DISP=L,COPIES=3,FORMS=BLUE)
...
EDT
DEFINE PRINTER (2) OUTPUT '*' /* Prepare for DYNALLOC
DEFINE PRINTER (2) OUTPUT 'DSN=SPL2PWR'
NAME 'MYREPORT' CLASS 'K' DISP 'L' COPIES 3 FORMS 'PINK'
WRITE (1) 'HELLO'
WRITE (2) 'HELLO'
END
.E
RUN
```

NATVSE Dynamic Work File Allocation (DYNALLOC) Support

Natural under z/VSE offers functionality to define work files dynamically, that is, these files need not be predefined in JCL. This means that Natural under z/VSE adds labels into the partition's temporary labels area for work files defined using a `DEFINE WORK FILE` statement.

In this respect Natural under z/VSE does not modify existing label information. All file labels dynamically added by a Natural session are deleted at session termination.

The following topics are covered below:

- [Prerequisites](#)
- [DEFINE WORK FILE Keyword Parameters](#)
- [Rules for Using the DEFINE WORK FILE Keyword Parameters](#)

- Samples

Prerequisites

A disk file manager is required, as it is not feasible to have fix file extent information within Natural application programs, particularly when these programs are executed in parallel in the same partition or in several partitions. This is not a restriction, as at least VSAM/SAM is available under z/VSE.

For dynamic allocation support by Natural under z/VSE, the following keyword subparameters of Natural profile parameter `WORK` have been made sensitive:

- `BLOCKS` (Number of Storage Blocks)
- `DISP` (File Open Mode)

DEFINE WORK FILE Keyword Parameters

The following keyword parameters are available for dynamic work files under VSE:

Keyword Parameter	Purpose
<code>CAT=</code> <i>catalog</i>	Triggers the usage for VSAM/SAM for the dynamic work file, where <i>catalog</i> is the 1 to 7 characters VSAM catalog file name. As this parameter is mandatory, if you want to use VSAM/SAM, you have to specify a VSAM job catalog explicitly (<code>CAT=IJSYSUC</code>).
<code>VOL=</code> <i>volser</i>	If specified a <code>// EXTENT</code> information is generated with that 1 to 5 characters volume serial number; its content depends on the preceding <code>DLBL</code> information, see below.
<code>DSN=</code> <i>fileid</i>	Is the DSN to set the file ID (optional).

Rules for Using the DEFINE WORK FILE Keyword Parameters

Potential `CAT` or `VOL` parameters have to come first in the `DEFINE WORK FILE` string, as the end of the parameter value can easily be found; the `DSN=` parameter must be specified as the last keyword parameter.

In other words, if `CAT` or `VOL` parameters are specified, and the `DSN` keyword parameter is not specified, all data in the `DEFINE WORK FILE` parameter string behind the last keyword parameter is considered as file ID to be set.

For VSAM/SAM, NATVSE dynamically adds the following label information:

```
// DLBL xyyyyyz, 'file-id', 0, VSAM, CAT=catalog, +
  RECORDS=n1, RECSIZE=n2, DISP=(dsp1, dsp2)
// EXTENT ,volser optional
```

where:

xx is the partition's SYSLOG ID, for example BG, F4, etc.

yyyy *yyyy* is the edited z/VSE two-byte hexadecimal task number (to allow Natural subtasks in the same partition).

z is the Natural work file number: "1" through "9" for files 1 to 9, "A" through "W" for files 10 to 32.

n1 is the value specified by the keyword subparameter BLOCKS of profile parameter WORK.

n2 is the value specified by the keyword subparameter BLKSIZE of profile parameter WORK.

dsp1, *dsp2* is the value specified by the keyword subparameter DISP of profile parameter WORK.

catalog is the VSAM catalog which has to be set using a DEFINE WORK FILE statement.

volser is the volume serial number on which the file is allocated.

Note that the EXTENT card is only generated when *volser* has been set in a DEFINE WORK FILE statement in the Natural application.

For other disk file management systems, for example CA-DYNAM/D, NATVSE adds dynamically the following label information:

```
// DLBL xyyyyyz, 'file-id', 0
// EXTENTsysnnn, volser, , , 1, n1 optional
```

where:

xx is the partition's SYSLOG ID, for example BG, F4, etc.

yyyy is the edited z/VSE two-byte hexadecimal task number (to allow Natural subtasks in the same partition).

z is the Natural work file number: 1 through 9 for files 1 to 9, A through W for files 10 to 32.

n1 is the value specified by the keyword subparameter BLOCKS of profile parameter WORK.

sysnnn is the value specified by the keyword subparameter SYSNR of profile parameter WORK.

volser is the volume serial number on which the file is allocated.

Note that the EXTENT card is only generated when *volser* has been set in a DEFINE WORK FILE statement in the Natural application. If *n1* is zero, extent information (start track/block and number of tracks/blocks) is omitted.

For Natural as a server the file name setup has the format:

```
xyyyyyz
```

where:

`xyyyyy` is the server session number in hexadecimal format (edited) with the very first character forced alphabetic by translation of 0 through F into A to P.

`z` is the Natural work file number, as for normal Natural under z/VSE batch operation.

Files to be dynamically allocated must have a Natural file name of '*' set in keyword subparameter `DEST='*'` of profile parameter `WORK` or a statement definition of `DEFINE WORK FILE '*'` to enable Natural under z/VSE to create new file names as described above.

The file identification to be used also has to be set using a `DEFINE WORK FILE` statement.

The regular z/VSE restrictions for file IDs apply.

In a Natural multitasking or server environment, it is recommended to provide some unique information in the file ID to prevent "equal file" conditions.

Samples

Natural parameters:

```
WORK=((1-6),AM=1,DEST='*',BLOCKS=100),WORK=((2),
DISP=(OLD,DELETE)),WORK=((6),BLOCKS=0)
```

Natural work file definition within application:

```
DEFINE WORK 1 'CAT=IJSYSUC,VSAM.SAM.FILE'
DEFINE WORK 2 'CAT=IJSYSCT,DSN=ANOTHER FILE'
DEFINE WORK 3 'CAT=IJSYSUC,VOL=DOSRES,ONE MORE FILE'
DEFINE WORK 4 '==.CATALOGED.FILE'
DEFINE WORK 5 'VOL=POOL01,DSN=FILE WITH EXTENT INFO'
DEFINE WORK 6 'VOL=DOSRES,ANY FILE'
```

z/VSE labels generated:

```
// DLBL xxyyyy1,'VSAM.SAM.FILE',0,VSAM,CAT=IJSYSUC,+
RECORDS=100,RECSIZE=4628,DISP=(NEW,KEEP)
// DLBL xxyyyy2,'ANOTHER FILE',0,VSAM,CAT=IJSYSCT,+
RECORDS=100,RECSIZE=4628,DISP=(OLD,DELETE)
// DLBL xxyyyy3,'ONE NORE FILE',0,VSAM,CAT=IJSYSUC,+
RECORDS=100,RECSIZE=4628,DISP=(NEW,KEEP)
// EXTENT ,DOSRES
// DLBL xxyyyy4,'==.CATALOGED.FILE',0
// DLBL xxyyyy5,'FILE WITH EXTENT INFO',0
```

```
// EXTENT SYS005,P00L01,,,1,100
// DLBL xxyyyy6,'ANY FILE',0
// EXTENT SYS006,DOSRES
```

Debugging Facilities for Natural under z/VSE

The Natural z/VSE batch mode interface contains some debugging facilities which can help you to track down problems.

These facilities are controlled by the `UPSI` settings in the JCL.

Additionally, the `UPSI` settings may also be specified as a Natural profile parameter (`UPSI=1XXXXXXX`, for example). This is useful if the `UPSI` settings in the JCL have produced side effects in the sense that they have a different meaning for other programs such as for front-end Natural or for programs called by Natural.

There may be the following `UPSI` settings:

UPSI Setting	Meaning
UPSI 1xxxxxxx	Dump Flag
UPSI x1xxxxxx	Trace Flag
UPSI xx1xxxxx	Print Output Identification Flag
UPSI xxx1xxxx	Storage Freeze Flag
UPSI xxxxxx1x	Abend Exit Flag

These settings are described below. In addition, a sample job is given to show you how to obtain [documentation for debugging](#).

UPSI 1xxxxxxx - Dump Flag

When Natural encounters a problem, the corresponding job usually cancels without a dump, unless an abend actually occurred. When this `UPSI` flag is set, a dump is always created at the end of the job when an error occurs, that is, when the Natural session termination message is other than NAT9995.

UPSI x1xxxxx - Trace Flag

When this flag is set, snapshots are taken of the register save area at some strategic points in Natural.



Note: Depending on the product sample output, setting this flag can lead to large output.

On entry of all NATVSE service routines, the name of this routine and the general registers 0 to 15 (GRG) are displayed.



Note: You can identify the caller from Register 14.

On exit of all NATVSE service routines, the name of this routine, the current general registers (GRG) and Registers 0 to 15 of the currently assigned save area (CSA) are displayed.



Notes:

1. The contents of the CSA are returned to the caller of the service routine, except the Register 15 return code which is taken from the general registers.
2. The contents of the HSA are returned to the caller, which means that this save area contains the return code in Register 15 if a return code was set at all.

Whenever the GRG registers are set, the debugging trace program tries to determine the name of the calling routine and the offset of the call from the beginning of the routine.

The `SYSnnn` number for the debugging trace print output is `SYS040`, as long as this `SYSnnn` number is assigned to a printer device; otherwise `SYSLST` is used. This is of particular interest if debugging trace output and other Natural print output are to be separated; to do so, assign `SYS040` appropriately and supply a `POWER * $$ LST` statement for this logical print unit.

UPSI xx1xxxx - Print Output Identification Flag

When this flag is set, the VSE (sub)task ID is inserted into the last four bytes of a print record to help identify the task which printed the record.



Important: When this flag is set, potential print data may become overwritten. Overwritten data is indicated by `'...'` preceding the VSE (sub)task ID.

UPSI xxx1xxxx - Storage Freeze Flag

On normal or abnormal session termination, Natural, by default, releases all its resources including storage. Despite the setting of `UPSI 1`, a dump may be useless, because all relevant storage has already been released during Natural termination. When this flag is set, no `GETVIS` storage acquired earlier is ever released within this job; this applies to all external subroutine programs called by Natural including the Natural nucleus (if not linked to `NATVSE`) and `RCA=ON` subproducts.

 **Caution:** This flag should be handled carefully, because more partition `GETVIS` storage is used, but jobs may still cancel due to failed `GETVIS` requests if the operating system storage requests cannot be satisfied.

UPSI xxxxxx1x - Abend Exit Flag

This flag may be helpful in the case of recurrent abends.

In batch mode, Natural usually has a check abend exit for active programs (`STXIT PC`) to recover from program checks (`NAT095n` error messages). When `DU=ON` is specified, this exit creates a snap dump and passes control to Natural for a clean session termination.

When this flag is set, the Natural session runs without any abend exit for active programs, which means that all program checks are handled directly by the operating system.

If this flag is set, the dump flag, the storage freeze flag, the session abend flag and the formatted dump-only flag are ignored.

Obtaining Documentation for Debugging

If a problem has to be analyzed, any information which might be relevant is important, in particular, the executed `JCS` and the corresponding console log.

The following sample job is intended to show you how to obtain comprehensive documentation:

```
// JOB sample job
// OPTION LOG,PARTDUMP to see JCL on printer
/* Library Definitions: labels and LIBDEFs
...
/* ADARUN Parameter Input Definition
// ASSGN SYS000,SYSRDR
/* Natural Work File Definitions
// DLBL CMWKFnn,'...',... disk work file
// EXTENT SYSnnn,volser,,nn,mm
// ASSGN SYSnnn,DISK,VOL=volser,SHR
// TLBL CMWKFnn,'...',... labeled tape work file
// ASSGN SYSnnn,uuu assignment to tape unit
/* Natural Print File Definitions
// ASSGN SYSnnn,uuu assignment to print UR unit
// DLBL CMPRTnn,'...',... print file on disk
```

```
// EXTENT SYSnnn,volser,,nn,mm
// ASSGN SYSnnn,DISK,VOL=volser,SHR
// TLBL CMPRTnn,'...',... print file on labeled tape
// ASSGN SYSnnn,uu assignment to tape unit
/* Debugging Options
// ASSGN SYS040,SYSLST debugging trace unit
// UPSI 1xxx00xx flags as discussed above
// EXEC NATURAL,SIZE=...
... dynamic parameters
/* end of dynamic parameters
... ADARUN parameters
/* end of ADARUN parameters
... Natural input
/* end of Natural input
// EXEC LISTLOG print console messages
/& end of job
```

NATVSE Attention Interrupts

The Natural z/VSE batch mode interface (NATVSE) supports attention interrupts via the console command `MSG xx`, where `xx` is the z/VSE partition ID a console operator can force on a NAT1016 attention interrupt event.

This special functionality is controlled by the Natural profile parameter `ATTN`.

27

Natural in Batch Mode under BS2000

■ Files and System Files Used by Natural in BS2000 Batch Mode	194
■ Keyword Parameters	196
■ BS2000 Job Variables	205

This document contains special considerations that refer to Natural in batch mode under the operating system BS2000.

See also *Natural under BS2000 Batch Mode Error Messages*.

For considerations that refer to Natural in batch mode generally, see:

- [Adabas Data Sets](#)
- [Sort Data Sets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

Files and System Files Used by Natural in BS2000 Batch Mode

The following optional sequential files or system files are used during a Natural under BS2000 batch mode session:

Link Name	System File	Explanation
CMPRMIN		Dynamic Parameter Data Set
	SYSIPT	Dynamic Parameter Input
	SYSDTA	Dynamic Parameter Input
	SYSDTA	Primary Input and Input for Natural INPUT Statements
	SYSLST	Primary Report Output
	SYSOUT	Primary Report Output
	SYSLST nn	Optional Report Output for Natural Tracing or Additional Reports, nn is the report number
Pnn		Additional Reports, nn is the report number
Wnn		Natural Work Files, nn is the work file number

CMPRMIN - Dynamic Parameter File

CMPRMIN can be used as dynamic parameter file if the system files SYSIPT or SYSDTA shall not be used or are not available to Natural. The parameter file must be of FCBTYPE SAM.

All input records from CMPRMIN are concatenated into one parameter string. Trailing blanks at the end of each record are truncated; no commas are inserted.

For further information on reading dynamic parameters, see the keyword parameter DYNPAR for macro NAMBS2 (see DYNPAR=FILE).

SYSIPT - Dynamic Parameter System File

The system file `SYSIPT` can be used as dynamic parameter file.

All input records from `SYSIPT` are concatenated into one parameter string. Only the first 72 positions of each `SYSIPT` record are significant. Trailing blanks at the end of each record are truncated; no commas are inserted.

For further information on reading dynamic parameters, see the keyword parameter `DYNPAR` for macro `NAMBS2`.

SYSDTA - Dynamic Parameter System File

The system file `SYSDTA` can be used as dynamic parameter file.

All input records from `SYSDTA` are concatenated into one parameter string. Trailing blanks at the end of each record are truncated; no commas are inserted.



Note: If `SYSDTA` is assigned to `SYSCMD`, the parameter input has to be closed off by an `/EOF` command to separate it from succeeding primary input data.

For further information on reading dynamic parameters, see the keyword parameter `DYNPAR` for macro `NAMBS2` (see `DYNPAR=FILE`).

SYSDTA - Primary Input

The system file `SYSDTA` is used as the primary input file that contains Natural commands, Natural source programs, and (optionally) data to be read by `INPUT` statements during the execution of Natural programs.

The number of characters actually processed per line is determined by the current setting of the profile/session parameter `SL`. This setting applies for both source statement and execution time input data. This enables identification or sequence numbers to be placed in the rightmost columns of every record, if desired.

SYSOUT, SYSLST - Primary Report Output

The system files `SYSOUT` or `SYSLST` are used for the primary output report, resulting from `DISPLAY`, `PRINT` and `WRITE` statements in a Natural program.

The actually used system file depends on the value for the keyword parameter `WRITE` in the assembly of the reentrant part of the Natural batch driver.

The system files `SYSOUT` or `SYLST` are also used as optional report output for dynamic parameters. If the profile parameter `PLOG` is set to `ON`, all dynamic profile parameters are written to the same destination as the primary report output.

SYSLSTnn - Optional Report Output for Natural Tracing

If profile parameter `ETRACE` is set to `ON`, all trace output is written to this file during the session.

Depending on the value for the keyword parameter `TRACE` in the assembly of the reentrant part of the Natural batch driver, one of the alternate `SYSLST` system files `SYLST01 - SYSLST99` is used as destination for the trace records.

Pnn - Additional Reports 01-31

`Pnn` is used for each additional report referenced by any Natural program compiled or executed during the session. `nn` must be a two-digit decimal number in the range 01-31, corresponding to the report number used in a `DISPLAY`, `PRINT` and `WRITE` statement.

Instead of `Pnn`, any other link names may be used by setting the keyword subparameter `DEST` of profile parameter `PRINT` to an appropriate value, for example:

```
PRINT=((nn),...,DEST=PRNTnn)
```

Wnn - Natural Work Files 01-32

`Wnn` is used for each Natural work file referenced by any Natural program compiled or executed during the session. `nn` must be a two-digit decimal number in the range 01 - 32, corresponding to the number used in a `READ WORK FILE` or `WRITE WORK FILE` statement.

Instead of `Wnn`, any other link names may be used by setting the subparameter `DEST` of profile parameter `WORK` to an appropriate value, for example:

```
WORK=((nn),...,DEST=WRKnn)
```

Keyword Parameters

The Natural BS2000 batch mode driver is generated by assembling the macro `NAMBS2`. For the control of conditional assembly of the driver modules, the following keyword parameters are available:

`ADACOM` | `ADDBUFF` | `APPLNAM` | `CODE` | `DELETE` | `DYNPAR` | `ILCS` | `JV` | `LF` | `LINK` | `LINK2/LINK3/LINK4`
| `NUCNAME` | `PARMOD` | `REQMLOC` | `SYSDTA` | `TERM` | `TIMESTAMP` | `TRACE` | `USERID` | `WRITE`

ADACOM

This parameter applies to the generation of the environment-dependent nucleus. It determines which Adabas link module is to be used.

Possible values:

ADACOM=ADAUSER	The module ADAUSER is linked to the environment-dependent nucleus.
ADACOM=ADABAS	The module ADAUSER is linked to the environment-dependent nucleus.
ADACOM=ADALNK	The module ADALNK is linked to the environment-dependent nucleus or the modules ADALNK and SSFB2C are linked to the environment-dependent nucleus. This is the default.

In any case a resolve to the Adabas module library has to be given in the linkage step for the environment-dependent nucleus.

ADDBUFF

This parameter applies to the generation of the non-reentrant part.

It determines the additional number of pages for the terminal I/O buffer.

Possible values:

ADDBUFF= <i>n</i>	<i>n</i> specifies the number of pages in 4 KB units. Range of values: 1 to 8.
	There is no default.

APPLNAM

This parameter applies to the generation of the non-reentrant part.

Possible values:

APPLNAM= <i>name</i>	<i>name</i> is the name (maximum 8 characters) of the Natural batch application. This name is part of the serialization ID when the Natural batch task is initialized.
APPLNAM=NATBS2	This is the default.

CODE

This parameter applies to the generation of both the non-reentrant part and the reentrant part.

It determines which part of the Natural BS2000 interface is to be generated.

Possible values:

CODE=FRONT	Indicates the generation/assembly of the non-reentrant part. This is the default.
CODE=RENT	Indicates the generation/assembly of the reentrant part.

DELETE

This parameter applies to the generation of the reentrant part.

Possible values:

DELETE=ON	The setting of the profile parameter DELETE in the Natural parameter module module determines whether dynamically loaded non-Natural programs are unloaded at the end of the Natural program in which they are loaded or whether they are unloaded when command mode is entered. This is the default.
DELETE=OFF	A dynamically loaded non-Natural program once loaded is kept for the duration of the whole Natural session.

DYNPAR

This parameter applies to the generation of the non-reentrant part.

Possible values:

DYNPAR=NO	No dynamic parameters are read. This is the default.
DYNPAR=SYSDTA	The dynamic parameters are read from SYSDTA. If SYSDTA is assigned to SYSCMD, at least an /EOF card must follow the /EXEC Natural card. Example:

	<pre> /LOGON /SYSFILE SYSDTA=(SYSCMD) /EXEC NATBAT /EOF * Null dynamic parameters LOGON SYSEXTP L * * FIN /LOGOFF </pre>
DYNPAR=SYSIPT	The dynamic parameters are read from SYSIPT.
DYNPAR=FILE	<p>The dynamic parameters are read from a sequential file. The input of this SAM file is interpreted as one single text string, which means that the individual entries must be separated from each other by a comma, even at the end of a line. Such a parameter file must be defined with a FILE command by using the LINK parameter CMPRMIN.</p> <p>Example:</p> <pre> /FILE NAT.PARAMS, LINK=CMPRMIN </pre>

ILCS

This parameter applies to the generation of the reentrant part.

Possible values:

ILCS=CRTE	<p>3GL subprograms are invoked with common runtime environment convention. For this to be possible, the ILCS initialization routine ITOSL# must be linked to the Natural environment-dependent nucleus (see the <i>Installation</i> documentation):</p> <pre> INCLUDE ITOSL#,SYSLNK.CRTE.010 RESOLVE,SYSLNK.CRTE.010 </pre>
ILCS=YES	<p>3GL subprograms are invoked with enhanced ILCS linkage convention. For this to be possible, the ILCS initialization routine ITOINITS must be linked to the Natural environment-dependent nucleus (see the <i>Installation</i> documentation):</p> <pre> INCLUDE ITOINITS,SYSLNK.ILCS RESOLVE,SYSLNK.ILCS </pre>
ILCS=NO	<p>Standard processing applies.</p> <p>This is the default.</p>

JV

This parameter applies to the generation of the non-reentrant part.

Possible values:

JV=ON	The condition code created when the Natural session is terminated is passed to a job variable if one has been declared with the link name *NATB2JV.
JV=OFF	If your BS2000 installation does not include the BS2000 Job Variables subsystem, this parameter must be set to OFF; otherwise assembly errors in the NAMBS2 compilation occur. This is the default.

LF

This parameter applies to the generation of the non-reentrant part.

With this parameter, you specify the control character to be used for line advance when printing on the local printer.

Possible values:

LF=X 'zz'	zz: line advance control character in hexadecimal format.
LF=X '25'	This is the default.

LINK

This parameter applies to the generation of the non-reentrant part.

Possible values:

LINK= <i>name</i> (LINK= <i>name,name,...</i>)	The <i>name(s)</i> of programs and modules that are called from Natural programs and linked with the non-reentrant part must be specified with this parameter. Conversely, the programs and modules whose names are specified must be linked with the non-reentrant part, otherwise the application is put into status SYSTEMERROR and all users are rejected with an error message.
	There is no default.

A "TABLE" macro call is performed for the specified programs and modules, which enters their load addresses into the dynamic loader's link table. It is therefore not necessary to dynamically load these programs when they are called by Natural programs. For dynamically loaded programs, only the load library needs to be defined in the [Natural parameter module](#).

Example:

```
LINK=PROG1
LINK=(PROG1,PROG2,MODUL111)
```

LINK2/LINK3/LINK4

These parameters apply to the generation of the non-reentrant part.

The parameters LINK2, LINK3 and LINK4 are an extension of the LINK parameter. Since an operand definition cannot be longer than 127 characters (including parentheses), these parameters are provided for cases where the operand of parameter LINK would be too long.

Possible values:

LINK n = <i>name</i> LINK n =(<i>name</i> , <i>name</i> ,...)	<i>n</i> : 2, 3 or 4. The rest of the parameter syntax is analogous to that of LINK.
	There is no default.

Examples:

```
NAMBS2 LINK=(PROG1,PROG2,...),
LINK2=(PROG54,...)
NAMBS2 LINK=(PROG1,PROG2,PROG3,PROG4)
```

NUCNAME

This parameter applies to the generation of the non-reentrant part.

With this parameter, you specify the name of the bounded, reentrant Natural module. You must use this name for the Natural pool and load information in macro ADDON (macro ADDON assembles BS2STUB).

Possible values:

NUCNAME= <i>name</i>	<i>name</i> is the name of the bounded, reentrant Natural module
NUCNAME=NB2RENT	This is the default.

PARMOD

This parameter applies to the generation of both the non-reentrant part and the reentrant part.

Possible values:

PARMOD=(<i>nn</i> , <i>loc</i>)	<i>nn</i> : 24/31 <i>loc</i> : BELOW/ABOVE
PARMOD=(31, ABOVE)	This is the default.

The first part of this parameter (*nn*) is used to define an addressing mode (24-bit or 31-bit mode) for the Natural BS2000 application.

31-bit mode is required if the Natural buffer pool, the reentrant part of the Natural BS2000 application, Adabas or the Adabas Fast Path pool is located above 16 MB.

The second part of this parameter (*loc*) is used to define the location of the Natural environment-dependent nucleus (see the *Installation* documentation). If you load the environment-dependent nucleus, this must be defined in the environment-dependent nucleus link procedure as follows:

```
LOADPT=*XS
```

or

```
LOADPT=X'address'
```

Example:

```
/EXEC TSOLINK  
PROG NATvrs, FILENAM=NATvrs, LOADPT=*XS, ...  
TRAITS RMODE=ANY, AMODE=31  
INCLUDE ...  
/* PARMOD=(nn, loc) MUST BE IDENTICAL IN THE NON-REENTRANT AND REENTRANT PARTS
```

vrs represents the relevant product version (see also *Version* in the *Glossary*).

REQMLOC

This parameter applies to the generation of both the non-reentrant part and reentrant part in 31-bit mode (PARMOD=31).

This parameter determines where the requested Natural work areas are to be allocated by the system using request memory.

Possible values:

REQMLOC=BELOW	All areas are requested below 16 MB.
REQMLOC=ABOVE	All areas are requested above 16 MB.
REQMLOC=RES	All areas are requested depending on the location of the reentrant part. This is the default.

The REQMLOC parameter corresponds to the LOC parameter of the BS2000 system macro REQM.

SYSDTA

This parameter applies for the generation of the non-reentrant part.

Possible values:

SYSDTA=PRIMARY	After reading of dynamic parameters from SYSDTA, SYSDTA is set to SYSFILE SYSDTA=(PRIMARY). This is the default.
SYSDTA=SYSCMD	After reading of dynamic parameters from SYSDTA, SYSDTA is set to SYSFILE SYSDTA=(SYSCMD).

TERM

This parameter applies to the generation of the non-reentrant part.

Possible values:

TERM=PRGR	The Natural batch application will be terminated. This is the default.
TERM=STEP	The system additionally executes the next SET -JOB -STEP command.

TIMESTMP

This parameter applies to the generation of the non-reentrant part.

It determines the time base for all system variables and time stamps derived from the machine time.

Possible values:

TIMESTMP=UTC	Time base is UTC (former GMT). This is the default.
TIMESTMP=LOCAL	Time base is the local machine time

TRACE

This parameter applies to the generation of the reentrant part.

With this parameter, you specify the number of a trace file and the maximum length of a trace print record.

Possible values:

TRACE= <i>nn</i> , <i>ll</i>	<i>nn</i> is the number for the SYSLST <i>nn</i> trace file in the range of 01 to 99. <i>ll</i> is the maximal length in characters of a trace print record in the range of 71 to 132.
TRACE=99, 71	This is the default.

If any external Natural trace function is active, the trace records will be written to SYSLST*nn*. In this case, the Natural batch mode driver creates the following trace file:

Example:

```
NATURAL.TRACE.BTCH.TTTT,SPACE=(30,3)
SYSFILE SYSLSTnn=Natural.TRACE.BTCH.TTTT
/* TTTT is the task sequence number
```

Before the Natural batch mode session is terminated, the trace file will be closed as follows:

```
SYSFILE SYSLSTnn=(PRIMARY)
```

USERID

This parameter applies to the generation of the non-reentrant part.

Possible values:

USERID=SYSTEM or USERID=YES	The Natural user ID is created by using the BS2000 user ID.
USERID=USER or USERID=NO	The Natural user ID is created by using the job name; that is, the / .JOBNAME of the LOGON command. If no BS2000 job name has been specified with the LOGON command, the Natural user ID is created as with USERID=SYSTEM or YES.
USERID=USER	This is the default.

WRITE

This parameter applies to the generation of both the non-reentrant and the reentrant part.

This parameter defines the destination of the output produced by Natural.

Possible values:

WRITE=SYSOUT	Natural output is written to SYSOUT.
WRITE=SYSLST	Natural output is written to SYSLST.
	This is the default value.

BS2000 Job Variables

The Natural batch mode driver uses the BS2000 facility “Job Variables” to pass return codes to the user or to subsequent jobs (steps). The return codes are created either by Natural itself (in the range 1 to 31) or by the Natural application if a `TERMINATE` statement is used with the condition-code option (the range to be used is 32 to 256).

The job variable which is to contain the return code has to be declared using the link name `*NATB2JV`. The support of job variables depends on the setting of the `SET` parameter `&JV` in the Natural BS2000 batch mode driver `NATBS2`.

Example:

```
/LOGON
/DCLJV NATBJV, LINK=*NATB2JV
/EXEC NATnnnB
*TERMCC
/LOGOFF
```

To assign Return Code 36 to `NATBJV`, the Natural program `TERMCC` could be coded as follows:

```
ASSIGN CC(N8) = 36
TERMINATE CC
END
```


28

Natural in Batch Mode (All Environments)

▪ Adabas Data Sets	208
▪ Sort Data Sets	208
▪ Subtasking Session Support for Batch Mode Environments	208

This document contains general considerations that apply when running Natural in batch mode.

Adabas Data Sets

Adabas data sets must be specified only in single-user mode. They are identical to those required for the execution of any normal application program using Adabas. See the relevant Adabas documentation for detailed information on Adabas data sets.

Sort Data Sets

Sort data sets must be specified if a Natural program containing a `SORT` statement is to be executed during the Natural session.

The requirements are identical to those for execution of a normal COBOL or PL/1 application program that invokes the operating system sort program and can vary according to the sort program in use.

Natural does not require the intermediate data sets `SORTIN` and `SORTOUT`, but communicates with the sort program via the `E15` and `E35` user-exit routine interfaces.

Subtasking Session Support for Batch Mode Environments

- [Purpose](#)
- [Prerequisites](#)
- [Functionality](#)
- [Starting a Natural Session](#)
- [Starting a Subtask](#)
- [Accessing the User Parameter Area](#)

Purpose

With subtasking support, you can run multiple Natural batch mode sessions within one address space. This allows parallel processing within one address space, rather than executing subsequent job steps, and can increase throughput dramatically.

Typically, client/server applications and products would take advantage of this functionality, for example, the Natural remote procedure call. Multiple server subtasks can be started to communicate with remote clients.

Prerequisites

If you wish to restart the Natural nucleus, it must be linked as a reentrant module (linkage editor option `RENT`).

The Adabas link routine (`ADALNK`) must be generated with reentrancy support.

Functionality

You start a subtask by issuing a `CALL` statement from a Natural program. The new Natural session (“subtask”) is started with an extended front-end parameter list. This list contains up to three parameter sets:

- dynamic Natural profile parameters,
- startup parameters,
- user parameters.

Variable names for standard I/O data sets (for example `CMPRINT`) and other parameters for the batch mode interface startup can be passed from the starting program in the startup parameter area. Standard I/O data sets can be undefined or dummy data sets; they can be owned by one session or shared by multiple sessions.

Furthermore, a `CALL` interface is provided for reading the user parameter area with a Natural program.

Starting a Natural Session

Extended Parameter List

The Natural batch mode interface without extended parameter list gets initial control from the operating system using standard linkage call. Register 1 points to an address with high-order bit on as the last address indicator. This address points to a halfword field containing the length of the following parameter area.

The extended parameter list contains up to three parameter addresses. This is indicated by the high-order bit in the last address which can be the first, second or third address. All parameter addresses point to a halfword field containing the parameter length of the following parameter area. Zero length indicates that there is no parameter area.

- The first parameter area contains the dynamic profile parameters for the Natural session.
- The second contains special startup parameters for the initialization of the batch mode interface.
- The third contains a user parameter area which can be accessed during the Natural session.

Startup Parameter Area

When multiple batch mode Natural (sub)tasks are running in the same region, by default these sessions access the very same Natural standard I/O data sets (such as `CMPRINT`, `CMSYNIN`, etc),

as there are no Natural profile parameters available to set these file names. Also by default the Natural system variables *INIT-ID and *INIT-USER are identical because of their definition for batch mode.

In order to provide unique standard I/O data set names and unique IDs for Natural subtask sessions the startup parameters in the extended parameter list can be used to overwrite the Natural system defaults. The Startup Parameter area is a table of pairs of 8-character fields:

- The first entry contains the 8-byte keyword to be replaced,
- the second entry contains the 8-byte replacement value.

Keywords and replacement values must be padded with trailing blanks, if necessary.

The following keywords are valid:

CMHCOPY	Permanent hardcopy destination
CMSYNIN	Command input data set name
CMOBJIN	Object input data set name
CMPRINT	Standard output data set name
CMPRMIN	Dynamic parameter input data set name
CMPLDG	Dynamic parameter output data set name
CMTRACE	Trace output data set name
INITID	Job step name (system variable *INIT-ID)
MSGCLASS	Spool class for dynamic allocation of CMPRINT and CMTRACE (z/OS only)
NATRJE	Job submission data set name (z/OS only)
STEPLIB	Program load library name (see also profile parameter LIBNAM, Name of Load Library, z/OS only)
SUBPOOL	z/OS storage subpool (0 - 127, right justified)
USERID	Initial user identification (system variable *INIT-USER)

The usage of these entries is optional and no particular sequence is required. A blank value for a data set means that this data set is not available or is empty.

Note concerning z/VSE:

By default, all print output (that is, the one resulting from CMPRINT, CMHCOPY, CMTRACE and CMPLOG) is routed to SYSLST. An overwrite specification for these files starting with SYS is considered a z/VSE system number overwrite. Possible format is SYS nnn where nnn is a three-digit number in the range from 000 to 099; if you specify an invalid number nnn , it is ignored.

User Parameter Area

The format of the user parameter area is free. It can be accessed from any Natural program by a special CALL interface see [Accessing the User Parameter Area](#).

Starting a Subtask

The following call interface is supplied to be used by Natural programs to start a subtask in the same address space.

PGMNAME	Natural nucleus name getting control (mandatory). To restart with the same nucleus, an asterisk can be specified as the first character. The actual nucleus name is passed back in this field.
NATPARML	Natural dynamic parameter area
STRPARML	Startup parameter area
USRPARML	User parameter area

All parameter areas must start with the length of the following parameters. The following example illustrates the usage of CMTASK.

Example:

```

DEFINE DATA LOCAL
01 PGMNAME (A8) INIT <'*'>
01 PARM1
02 NATPARML (I2) INIT <30>
02 NATPARMS (A30) INIT <'INTENS=1,IM=D,STACK=MYPROG'>
01 PARM2
02 STRPARML (I2) INIT <32>
02 STRPARAM1 (A16) INIT <'CMPRINT SYSPRINT'>
02 STRPARAM2 (A16) INIT <'CMPRMIN MYPARMS'>
01 PARM3
02 USRPARML (I2) INIT <80>
02 USRPARMS (A80) INIT <'special user parameters'>
END-DEFINE
CALL 'CMTASK' PGMNAME NATPARML STRPARML USRPARML
END

```

A sample program, ASYNBAT, can be found in library SYSEXTP.

Accessing the User Parameter Area

The user parameter area passed during startup can be read from any Natural program with the following CALL statement:

```
CALL 'CMUPARM' USRPARML USRPARMS
```

USRPARML is the length (I2) of the USRPARMS area (before the call) and the length of the data returned (after the call). USRPARMS is the parameter data area.

If the length of the data to be returned is greater than the area length, the data is truncated to the area length. The following return codes are possible:

0	Data successfully moved
4	Data moved but truncated
8	No data available
12	Length value not positive
16	Insufficient number of parameters

A sample program, `GETUPARM`, can be found in library `SYSEXTP`.

VII

Natural Buffer Pools

This part contains information about the various storage management functions that are available to a Natural administrator under the operating systems z/OS, z/VSE and BS2000.

Natural Buffer Pool - General

Natural Global Buffer Pool under z/OS

Natural Global Buffer Pool under z/VSE

Common GBP Operating Functions under z/OS and z/VSE

Natural Global Buffer Pool under BS2000

For a functional overview of the Natural buffer pool, see *Natural Buffer Pool* in the *Natural System Architecture* documentation.

For an overview of the Natural profile parameters that affect the Natural buffer pools, see *Storage Management* in the section *Profile Parameters Grouped by Category* in the *Parameter Reference* documentation.

29 Natural Buffer Pool - General

- Natural Buffer Pool Principle of Operation 216
- Buffer-Pool Monitoring and Maintenance 221
- Natural Global Buffer Pool 224

The buffer pool is a storage area into which Natural programs are placed in preparation for their execution. Programs are moved into and out of the buffer pool as Natural users request Natural objects. Conceptually, it serves a function similar to that of an operating system in loading programs in and out of a reentrant area. The Natural buffer pool is an integral part of Natural in all supported environments.

Natural Buffer Pool Principle of Operation

Natural generates reentrant Natural object code. A compiled program is loaded into the buffer pool and executed from the buffer pool. Thus, it is possible that a single copy of a Natural program can be executed by more than one user at the same time.

This section covers the following topics:

- [Objects in the Buffer Pool](#)
- [Directory Entries](#)
- [Text Pool](#)
- [Buffer Pool Hash Table](#)
- [Buffer Pool Initialization](#)
- [Buffer Pool Search Methods](#)
- [Local Buffer Pool](#)
- [Global Buffer Pool](#)
- [Buffer Pool Cache](#)

Objects in the Buffer Pool

Objects in the buffer pool can be programs, subprograms, maps and global data areas. Global data areas are placed in the buffer pool only for compilation. In this case, two objects with the same name are loaded in the buffer pool: the GDA itself and the corresponding symbol table.

Directory Entries

When a Natural object is loaded into the buffer pool, a control block called a directory entry is allocated to this object.

A directory entry contains such information as the name of the object, what library it belongs to, what database ID and Natural system file number the object was retrieved from, and some statistical information (for example, the number of users who are concurrently executing the program at a given point in time).

When a user executes a program, Natural checks the directory entries to see if the program has already been loaded into the buffer pool. If it is not already in the buffer pool, a copy of the program is retrieved from the appropriate Natural system file and loaded into the buffer pool.

When an object is loaded in the buffer pool, one or more other Natural objects which are currently not being executed may be deleted from the buffer pool in order to make room for the newly loaded object. When the new object is loaded, a new directory entry is created in order to identify this object.

When an object is deleted from the system file, it will also be deleted from the buffer pool as soon as it is no longer being used. When an object is newly cataloged or stowed, its old version will also be deleted from the buffer pool as soon as it is no longer being used; when it is requested for execution again, the new version will then be loaded from the system file into the buffer pool.

Text Pool

The actual object code of a program that is loaded into the buffer pool is placed into an area called the text pool and must be allocated as a contiguous piece of memory within this text pool. This text pool is divided into a number of 4 KB buffers. This is an arbitrary size and can be changed at the Natural administrator's discretion. When an object is loaded, one or more text buffers that are contiguous to each other are allocated to store the object code of the object.

Buffer Pool Hash Table

This section applies to buffer pools of `TYPE=NAT` only.

To speed up the search time for looking up an object in the buffer pool directory, a hash table is used. The number of entries in the hash table is twice the number of directory entries, rounded up to the next prime number. This will ensure that only half of the table is filled at any point in time and that the probability of collisions is near zero. As a consequence, the average number of tests to find an existing object in the hash table is theoretically less than 2.

The hash criterion is the eight character long program name. If more than one program name is hashed to the same location in the hash table, an overflow technique resolves the collisions.

The storage required for the hash table is roughly 16 bytes per text block. Thus, the available storage in the text pool is reduced by between 1.6% (1 KB text blocks) and 0.1% (16 KB text blocks).

Buffer Pool Initialization

In case of a global buffer pool the initialization occurs during start of the global buffer pool.

In case of a local buffer pool the initialization time varies depending on the environment.

- In batch mode, TSO and TIAM, the initialization occurs when you begin the execution of the Natural session.
- In a TP monitor environment, the initialization generally occurs when the first user invokes Natural under this TP monitor. Under Com-plete and CICS, it is also possible to initialize the local buffer pool when the TP monitor is started.

Buffer Pool Search Methods

As mentioned earlier and explained below, there are different search methods for allocating space in the buffer pool.

➤ To select a search method, use

- The Natural profile parameters `BPMETH` and `BPI`.
Or the macro `NTBPI` in the Natural parameter module.
Or the function parameter `METHOD` of the global buffer pool.

For a description of these parameters and the macro `NTBPI` refer to the *Natural Parameter Reference* documentation.

Below is information on the search methods:

- `METHOD=S`
- `METHOD=N`
- **Choosing Search Methods**

METHOD=S

`METHOD=S` indicates that a selection process is used as search algorithm for allocating storage in the buffer pool in order to obtain the space required to accomplish a new load.

The selection process used is a combination of search Algorithms 1 and Algorithm 2:

■ **Algorithm 1**

Search Algorithm 1 attempts to find storage in the buffer pool that is either free space or space occupied by an unused (active but not used) object.

If free space of the exact object size required is found, the selection process ends immediately. Otherwise, the search continues by browsing the buffer pool from top to bottom and comparing the entries in the buffer pool for best size fit. Additionally, in the case of unused objects, the search also considers the last attached time of the object, that is, the time the object was last referenced at a load or locate.

When the selection process has finished, either free space or the space of an unused object with a size greater than or equal to the size requested is selected. The rule of precedence that applies to the search is: free space is taken first and space of unused objects next. In the case of unused objects, the oldest objects are removed first.

If the selection process of Algorithm 1 was not successful, Algorithm 2 is invoked.

■ Algorithm 2

Search Algorithm 2 starts if Algorithm 1 fails. Algorithm 2 starts searching from a position in the buffer pool which is passed by Algorithm 1 and attempts to combine two or more entities (free storage and/or space occupied by unused objects) in order to obtain the necessary storage for a new load. However, the age of the object is not taken into account.

Algorithm 2 continues processing to the bottom of the text record section and, if necessary, wraps around to the top of the text record section to make one final pass from top to bottom. If space is still unavailable, Algorithm 2 fails, the object cannot be loaded and Natural issues a corresponding error message.

METHOD=N

METHOD=N indicates that the next available free or unused space is used in order to obtain the space required to accomplish a new load. Unused space is space that is occupied by an active but not used object.

The search for the next available space starts from a pointer that moves through the buffer pool in a wrap-around fashion. Any next available buffer pool entries that are free or contain unused objects can be used and possibly chained together to obtain the amount of storage requested.

If the bottom of the buffer pool is reached during an allocation request, the pointer is wrapped around to the top of the buffer pool and one final search is performed through the buffer pool from top to bottom. If the bottom of the buffer pool is reached again and the object cannot be loaded, the load fails and Natural issues a corresponding error message.

METHOD=N can especially be considered for large buffer pools in combination with the **buffer pool cache** function. For details, see also Choosing Search Methods below.

Choosing Search Methods

By default, METHOD=S is used. The advantage of this method is, that a diligent search is performed to allocate space, taking into account the size and the age of objects and guaranteeing that the most dispensable unused objects are removed from the buffer pool to provide space for a new load.

A disadvantage of METHOD=S can be the high CPU time that is consumed by the selection process when browsing the buffer pool from top to bottom.

The advantage of METHOD=N is the short selection process and, usually, little browsing that require less CPU time for allocating space. This can be significant to large buffer pools.

The disadvantage of METHOD=N is that objects are selected less carefully for removal from the buffer pool. To avoid an increase in Adabas I/Os for reloading removed objects, we recommend that you use METHOD=N in combination with the **buffer pool cache** function.

Local Buffer Pool

Using Natural as supplied on the installation tape, you are running a *local* buffer pool. This is a buffer pool area that is allocated in the same partition (or region or address space) of the particular environment in use.

For example, in a batch or TSO environment, each user has his/her own local buffer pool. In a TP monitor environment such as Com-plete, CICS or IMS TM, there is one buffer pool per TP monitor from which all TP users execute.

Global Buffer Pool

In a z/OS environment, a global buffer pool is allocated from CSA or ECSA storage. In such an environment, all TSO users, batch users and TP monitor users could be executing from one common global area.

In a z/VSE environment, a global buffer pool is allocated from System GETVIS Area (below or above). In such an environment, all batch users and TP monitor users could be executing from one common global area.

In a BS2000 environment, a global buffer pool is a common memory pool, see [Natural Global Buffer Pool under BS2000](#).

For further information on the global buffer pool, see [Natural Global Buffer Pool](#).

Buffer Pool Cache

This section applies to global buffer pools of `TYPE=NAT` and local buffer pools of `TYPE=NAT` or `TYPE=SWAP`.

The buffer pool cache is available in conjunction with global and local buffer pools. It is used only for Natural objects (programs, subprograms, maps, etc.), whereas it is not used for example for objects generated by Natural for DL/I.

When a buffer pool is not large enough to take up all objects requested by the different users, special overload strategies are used to replace existing objects with requested objects. The number of overload situations has a direct relation to the efficiency of the buffer pool. The best and most efficient way to reduce the disliked overloads, hence to improve the buffer pool performance, is simply to increase its size.

However, this choice is not applicable at most customer sites, due to a lack of available storage in the primary address space and/or the z/OS (E)CSA, z/VSE system GETVIS area or BS2000 Common Memory Pool.

In order to improve the situation described above, a buffer pool cache is used. The main purpose of this mechanism is to prevent a loss of all objects which were deleted from the buffer pool due to “short-on-buffer-pool-storage” situations. This means, that an object delete results in a “swap

out to buffer pool cache". The intended benefit of this feature is a reduction of database calls used for object loading and consequently a performance improvement.

Note for Global Buffer Pools:

The buffer pool cache area is allocated in a data space. When a data space is created for a buffer pool (profile parameter `BPCSIZE` specified for z/OS or z/VSE, or `DATA` parameter specified for BS2000), the ownership is assigned to the creator task. If this task terminates, the system automatically deletes the data space. Therefore, the creator task will stay alive in this case, regardless of whether `RESIDENT=Y` has been set or not.

Note for Local Buffer Pools: (z/OS and z/VSE only, not for Complete and not for IMS TM)

The buffer pool cache is allocated in a data space or in a memory object "above the bar", that is, in 64-bit memory (z/OS only). When a data space or memory object is created for a buffer pool (see profile parameters `BPCSIZE` and `BPC64`), the ownership is assigned to the creator task. If this task terminates, the system automatically deletes the data space or the memory object.

Buffer-Pool Monitoring and Maintenance

The Natural utility `SYSBPM` (described in the Natural *Utilities* documentation) provides statistical information on the current status of the buffer pool. `SYSBPM` also allows you to adjust the buffer pool to your requirements.

The following topics are covered below:

- [Preload List](#)
- [Blacklist](#)
- [Propagation of Buffer-Pool Changes](#)
- [Performance Considerations](#)

Preload List

A preload list is a list of objects that will be loaded into the buffer pool and remain there as resident. When a user requests such an object for execution, it is always already in the buffer pool and need not be loaded from the system file.

This may improve performance, may avoid buffer pool fragmentation, or may be useful to ensure that central error transactions are always available, even if the database containing the system file is not active.

At the start of the Natural session, Natural checks which of the objects on the preload list are already in the buffer pool. Those which are not will then be loaded from the system file into the buffer pool. This checking and loading is also performed whenever the buffer pool is connected, re-connected and re-initialized using the `SYSBPM` utility. If a global buffer pool is re-initialized by a `REFRESH` command, no checking takes place for existing Natural sessions. That is, as long as no

new Natural session is started that accesses this buffer pool, the objects on the preload list are not loaded.

The load of the preload list is not serialized. That means, if multiple Natural sessions start concurrently, each session tries to load all objects named in the preload list into the buffer pool. This may lead to duplicate entries of the same Natural object in the buffer pool (see also hint below).

A preload list is identified by name, and is attached to a specific buffer pool by specifying its name as startup parameter (for a global buffer pool) or in the `NTBPI` macro (for a local buffer pool). Thus, a different preload list may be defined for each buffer pool; or the same preload list may be used for different buffer pools.

If the specified preload list cannot be located, or if objects contained on the preload list cannot be loaded, Natural will issue a corresponding warning message at session initialization. In either case, the preloading will be repeated for each subsequent session initialization.

As the objects on the preload list are the first to be loaded, they are located at the beginning of the buffer pool (except if the initial preloading could not load all objects, in which case the objects may be located anywhere in the buffer pool).

To maintain preload lists, you use the `SYSBPM` utility, see *SYSBPM - Preload List Maintenance* in the Natural *Utilities* documentation.



Tip: To avoid problems with the load of the objects on a preload list by user sessions the following procedure is recommended:

■ **For a global buffer pool:**

Immediately after the allocation or refresh of the global buffer pool, start a batch Natural session that accesses the global buffer pool and that executes a `FIN`.

■ **For a local buffer pool under CICS and Com-plete:**

During startup of the TP system, start an asynchronous Natural session that access the local buffer pool, and put a `FIN` command on the Natural stack. Ensure that this Natural session references the name of the preload list in its `NTBPI` macro.

Blacklist

To prevent a Natural object from being executed, you can put it on a so-called “blacklist”: the object can then not be loaded into the buffer pool; and if it is already in the buffer pool, it cannot be executed. A user requesting such an object to be executed will then receive an appropriate error message.

You can put not only individual objects on the blacklist, but also entire libraries.

Examples

- The blacklist may be useful, when you upgrade a Natural application and do not wish users to continue to work with that application until you have finished the upgrade. Without the

blacklist, a user might execute a new module which in turn would invoke an old module - which might lead to an abnormal termination of the Natural session. With the blacklist, the user can will receive a message that the requested object can currently not be executed, and can then continue his/her Natural session.

- Performance aspects may be another reason for using the blacklist to prevent certain resource-consuming objects from being executed in a specific environment.
- You may use the blacklist to prevent the execution of test programs in a production environment.

To maintain the blacklist, you use the `SYSBPM` utility. See *SYSBPM - Blacklist Maintenance* in the Natural *Utilities* documentation.

Propagation of Buffer-Pool Changes



Note: Under z/OS, the propagation of buffer-pool changes is always restricted to the Natural subsystem in which the change has occurred (for details on the Natural subsystem, see [Natural Subsystem \(z/OS\)](#) or [Natural Subsystem \(z/VSE\)](#)). Thus, “all global buffer pools” in this context means “all global buffer pools within the same subsystem”.

In some environments, it is important that changes which occur in one (local or global) buffer pool are also propagated to all other global buffer pools - that is, the same changes are also automatically made in the other global buffer pool - so as to ensure the consistency of the buffer pools and the Natural applications being used. This is particularly important in a z/OS Parallel Sysplex environment.

For example, if a Natural program is newly cataloged in one z/OS image, the propagation will cause the program to be deleted from all other global buffer pools in the z/OS Parallel Sysplex environment, so that its new version has to be loaded from the system file when the program is to be executed again.

The following changes are propagated:

- an object is deleted from the buffer pool,
- the buffer pool's blacklist is modified,
- the buffer pool is re-initialized.

Changes can be propagated to all other global buffer pools within the current z/OS image, or within the entire z/OS Parallel Sysplex environment, or all other global buffer pools of the same name within the z/OS Parallel Sysplex environment.

The propagation does not affect those objects in another global buffer pool which are defined as resident in that buffer pool.

The propagation is activated and its scope controlled by the Natural profile parameter `BPPROP`.



Note: As the propagation is performed asynchronously and an object is only deleted from a buffer pool when it is not longer being used, it may take some time until the current version of an object is available in all buffer pools.

Propagation to other *local* buffer pools is not possible.

Performance Considerations

For general advice on performance-related issues regarding the buffer pool and the BP cache, see *Performance Considerations* in the section *SYSBPM Utility - Buffer Pool Management* of the *Natural Utilities* documentation.

Natural Global Buffer Pool

The Natural global buffer pool is an optional Natural component.

It is available for the following operating systems

- z/OS (refer to *Global Buffer Pool under z/OS*)
- z/VSE (refer to *Global Buffer Pool under z/VSE*)
- BS2000 (refer to *Global Buffer Pool under BS2000*).

Profile Parameters Used

The following Natural profile parameters are used to establish a global buffer pool:

BPNAME	Specifies the name of the global buffer pool (see BPNAME). BPNAME=' ' (blank) is used to establish a connection to the <i>local</i> buffer pool.
SUBSID	Specifies the ID of the Natural subsystem to be used (see profile parameter SUBSID; applies only under z/OS and z/VSE).

During Natural startup, Natural attempts to locate the global buffer pool using these parameters.

Buffer Pool Opening / Closing Procedure

With the `NTBPI` macro of the Natural parameter module or the corresponding profile parameter `BPI`, you can define more than one buffer pool.

At session initialization, Natural attempts to establish a connection to the first buffer pool defined. If this fails, Natural attempts to establish a connection to the second buffer pool defined. If that fails, too, it tries the next buffer pool defined, etc. Whenever an attempt to establish a connection to a buffer pool fails, Natural will issue a corresponding message.

The same procedure applies when a buffer pool is stopped: if you close the currently connected buffer pool while a Natural session is still active, Natural attempts to connect to another buffer pool (in the order in which they are defined) and continue the session. Thus, it is possible for the Natural administrator to close a global buffer pool without having to terminate all active Natural sessions.

30

Natural Global Buffer Pool under z/OS

▪ Using a Natural Global Buffer Pool	228
▪ Prerequisites	228
▪ Operating the Natural Global Buffer Pool	229
▪ Sample NATGBPvr Execution Jobs	231
▪ Localization	232
▪ Messages	233

This document describes purpose and usage of a Natural global buffer pool (GBP) under the operating system z/OS.

Certain parts of the Natural global buffer pool are identical under z/OS and z/VSE. These parts are concentrated in a separate section (see [Common GBP Operating Functions under z/OS and z/VSE](#)) which covers the following topics:

- [Global Buffer Pool Operating Functions](#)
- [Global Buffer Pool Function Parameters](#)
- [Examples of NATBUFFER Specifications](#)

Using a Natural Global Buffer Pool

Purpose

The Natural global buffer pool is a segment of storage assigned from the z/OS extended common system area (ECSA) above 16 MB (or from CSA storage below, if requested), used by Natural to load and execute Natural programs.

Benefits

Using a global buffer pool, multiple Natural sessions under different TP monitors (multiple copies of CICS, TSO, IMS TM, etc.) and/or in multiple batch sessions share the same area - thus requiring less storage than would be required for a local buffer pool in each environment.

Prerequisites

The following prerequisites must be met if you want to use a global buffer pool:

1. The module `NATGBPvr` must have been linked into an Authorized Program Facility (APF) library; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
2. The global buffer pool must have been started; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.

Operating the Natural Global Buffer Pool

The global buffer pool is operated using the program `NATGBPvr` which must be executed from within an Authorized Program Facility (APF) library.

The following topics are covered below:

- Allocation of the Natural GBP
- Setting up the Natural GBP
- Starting the Natural GBP Operating Program
- Stopping the Natural GBP Operating Program



Note: In the following document, *vr*s or *vr* represents the relevant version of the product. For information on product versions, see *Version* in the *Glossary*.

Allocation of the Natural GBP

If the z/OS parameter `ALLOWUSERKEYCSA(YES)` has explicitly been specified in `SYS1.PARMLIB(DIAGxx)`, a Natural global buffer pool is allocated in user key, so that Natural sessions accessing a global buffer pool have write permission for that buffer pool.

If `ALLOWUSERKEYCSA(NO)` is in effect, a Natural global buffer pool is allocated in system key; therefore, Natural sessions accessing a global buffer pool do not have any write permission for that buffer pool. These Natural sessions call the Authorized Services Manager (ASM) to perform all buffer pool functions. As a consequence, installation of the ASM is mandatory. The ASM is not only called to load a Natural object into the buffer pool but also to maintain the use count of a Natural object if the execution of this Natural object is started or terminated. The calls to the Authorized Services Manager will increase Natural's resource consumption. The overhead is hard to predict and depends on the application profile (ratio of program calls to program execution time).

Setting up the Natural GBP

The functions provided by the operating program `NATGBPvr` are activated in that they are

- specified in a parameter card (`PARM=`),
- read from a file (see below),
- or supplied by the `MODIFY` operator command.

`NATGBPvr` expects the first command in the parameter field (`PARM=`) of the `EXEC` statement.

You may enter:

- one of the **functions** (described in the section *Common GBP Operating Functions under z/OS and z/VSE*)
- or a reference to an input file with `CF=dd-name`, where `dd-name` represents a DD name defined in the JCL. Only “card image” files are supported, that is, `RECFM=F`, `LRECL=80`, and only the first 72 bytes of the input record are honored. Every record included from the input file represents a command. Blank records or records prefixed with an asterisk (*) are ignored. A file is processed until End-Of-File (EOF). Example: `PARM='CF=SYSIN1'`

If the parameter field is not supplied or blank, the commands will be read from file `SYSIN` by default.

It is only possible to enter one function at a time at the console or one function per line using the command file, otherwise an error message will be returned.

Each command received, from the parameter card, from file input or from operator console input is shown on the operator console.

Starting the Natural GBP Operating Program

To start the program `NATGBPvr`, either start a started task or submit a job, which executes `NATGBPvr`.

 **Important:** To ensure that the global buffer pool is retained after a system failure, the global buffer pool should be started automatically during machine IPL.

Stopping the Natural GBP Operating Program

After all commands are processed, the program `NATGBPvr` terminates, unless

- `RESIDENT=Y` was specified
- or a buffer pool with a cache was created.

`NATGBPvr` will return one of the following condition codes:

Condition Code	Explanation
0	All functions executed successfully.
20	An error has occurred; see the message log for details. See also the <code>CC</code> function parameter of the global buffer pool.

Sample NATGBPvr Execution Jobs

The following examples show sample batch jobs for creating and terminating a global buffer pool.

In the following examples, the notation *vr*s or *vr* represents the relevant product version. For information on product versions, see *Version* in the *Glossary*.

Example 1:

```
//GBPSTART JOB
//*
//* Starts a global buffer pool with the name NATvrGBP, a size of 1 MB and
//* a text block size of 4 KB. The global buffer pool is allocated above 16 MB.
//* The subsystem used is NATv.
//* After the allocation, the job GBPSTART terminates.
//*
//STEP EXEC PGM=NATGBPvr,PARM='BPN=NATvrGBP,N=(1M)'
//SETPLIB DD DISP=SHR,DSN=USER.APF.LINKLIB
```

Example 2:

```
//GBPRES JOB
//*
//* Starts a global buffer pool with the name GBP, a default size of
//* 100 KB and a text block size of 1 KB. The global buffer pool is allocated
//* below 16 MB. The subsystem used is SAGS.
//* After the allocation, the job GBPRES will wait for further commands.
//* Further commands may be entered using the MODIFY operator command:
//* F GBPRES,command-string
//*
//STEP EXEC PGM=NATGBPvr,PARM='BPN=GBP,N=(,BL,1),S=SAGS,R=Y'
```

Example 3:

```
//GBPSTOP
//*
//* Stops the global buffer pool GPB if it contains no active objects. If it
//* does contain active objects, the operator console will prompt for a reply.
//* Depending on the reply, the shutdown will be forced (Y) or aborted (N).
//* The subsystem used is NATv.
//*
//STEP EXEC PGM=NATGBPvr,PARM='FSHUT,BPN=GPB'
```

Example 4:

```
//GBPSTRT2
/* Read commands from SYSIN1:
/*
/* Start two global buffer pools (subsystem ID Nvrs) with names
/*   NATGBP1 - size=1024KB and a cache with size 2048KB, and
/*   NATGBP2 - size=2048KB without cache.
/* Display all buffer pools of subsystem ID Nvrs.
/*
/* Note: The job does not terminate by itself, but stays resident and waits
/*       for operator commands because it owns the data space allocated for
/*       buffer pool NATGBP1.
/*
/* To shut down the buffer pools, send the operator command MODIFY with
/* parameter CF=SYSIN2 to execute the corresponding FSHUTs.
/*
//STEP EXEC PGM=NATGBPvr,PARM='CF=SYSIN1'
//SYSIN1 DD *
CREATE,BPN=NATGBP1,S=Nvrs,N=(1M),BPC=2M
CREATE,BPN=NATGBP2,S=Nvrs,N=(2M)
SHOWBP S=Nvrs
//SYSIN2 DD *
FSHUT,BPN=NATGBP1,S=Nvrs
FSHUT,BPN=NATGBP2,S=Nvrs
SHOWBP S=Nvrs
/*
```

Localization

The module NATGBPTX is delivered in source form. It contains all error messages in English in mixed case. The messages can be translated into other languages as required. In this case, the “new” NATGBPTX source module has to be assembled and the module NATGBPvr has to be relinked.

To issue the global buffer pool messages including their variable parts in upper case, the global buffer pool parameter module NATGBPRM has to be assembled with the UCTRAN parameter set to YES, and the module NATGBPvr has to be relinked.

To relink the module NATGBPvr, use the following JCL:

```
//SYSLIN DD *  
SETCODE AC(1)  
SETOPT PARM(REUS=RENT)  
INCLUDE NATLIB(NATGBPMG)  
INCLUDE SMALIB(NATGBPRM)  
INCLUDE SMALIB(NATGBPTX)  
INCLUDE NATLIB(NATBPMGR)  
NAME NATGBPvr(R)  
/*
```

Messages

Refer to *Natural Global Buffer Pool Manager Messages* in the *Natural Messages and Codes* documentation

31 Natural Global Buffer Pool under z/VSE

▪ Using a Natural Global Buffer Pool	236
▪ Prerequisites	236
▪ Operating the Natural Global Buffer Pool	237
▪ Sample NATGBPvr Execution Jobs	238
▪ Localization	240
▪ Messages	240

This document describes purpose and usage of a Natural global buffer pool (GBP) under the operating system z/VSE.

Certain parts of the Natural global buffer pool are identical under z/VSE and z/OS. These parts are concentrated in a separate section (see [Common GBP Operating Functions under z/OS and z/VSE](#)) which covers the following topics:

- [Global Buffer Pool Operating Functions](#)
- [Global Buffer Pool Function Parameters](#)
- [Examples of NATBUFFER Specifications](#)

Using a Natural Global Buffer Pool

Purpose

The Natural global buffer pool is a segment of storage assigned from the z/VSE system GETVIS storage above 16 MB (or from storage below, if requested), used by Natural to load and execute Natural programs. The Natural global buffer pool is allocated in storage key 9, so that all participating partitions have write-access to it.

Benefits

Using a global buffer pool, multiple Natural sessions under different TP monitors (multiple copies of CICS, Com-plete, etc.) and/or in multiple batch sessions share the same area - thus requiring less storage than would be required for a local buffer pool in each environment.

Prerequisites

1. The global buffer pool manager module `NATGBPvr` must have been linked, and the start and stop jobs must have been created. See the corresponding step in the section *Installing Natural on z/VSE* in the *Installation for z/VSE* documentation.
2. A Natural global buffer pool under z/VSE requires the subsystem storage protection facility of an ESA/390 or compatible processor. Consequently, it also requires a minimum operating system level of z/VSE Version 2 Release 4 for support of this hardware feature.

Operating the Natural Global Buffer Pool

The global buffer pool is operated by the program `NATGBPvr` which is contained in and executed from the Natural load library.

The following topics are covered below:

- [Setting Up the Natural GBP](#)
- [Starting the Natural GBP Operating Program](#)
- [Stopping the Natural GBP Operating Program](#)

Setting Up the Natural GBP

The functions available from the operating program `NATGBPvr` are activated in that they are

- provided by a parameter card (`PARM=`),
- read from a file (see below)
- or supplied by the operator (`AR` command `MSGxx` with `xx` being the z/VSE partition ID) unless `NATGBPvr` has not been terminated.

`NATGBPvr` expects the first command in the parameter field (`PARM=`) of the `EXEC` job control statement.

You may enter:

- one of the **functions** described in the section *Common GBP Operating Functions under z/OS and z/VSE*
- or a reference to an input file with `CF=d1b1-name`, where `d1b1-name` represents a DLBL name defined in the JCL or the z/VSE (partition) standard labels. Only “card image” files are supported, that is, `RECFM=F`, `LRECL=80`, and only the first 72 bytes of the input record are honored. Every record included from the input file represents a command. Blank records or records prefixed with an asterisk “*” are ignored. An asterisk (*) for `d1b1-name` indicates to the module `NATGBPvr` that input has to be read from `SYSIPT`. A file is processed until End-Of-File (EOF).

Example: `PARM='CF=SYSIN1'`

If the parameter field is not supplied or is blank, the commands will be read from `SYSIPT` by default.

It is only possible to enter one function at a time at the console or one function per line using the command file, otherwise an error message will be returned.

Each command received from the parameter card, from file input or from operator console input is shown on the operator console and is logged to `SYSLST`.

Starting the Natural GBP Operating Program

To start the program `NATGBPvr`, submit a job that executes `NATGBPvr`.

 **Important:** To ensure that the global buffer pool is retained after a system failure, the global buffer pool should be started automatically during machine IPL.

Stopping the Natural GBP Operating Program

After all commands are processed, the operating program `NATGBPvr` terminates unless

- `RESIDENT=Y` was specified or
- a buffer pool with a cache was created.

`NATGBPvr` will return one of the following condition codes:

Condition Code	Explanation
0	All functions executed successfully.
20	An error has occurred; see the message log for details.

Sample NATGBPvr Execution Jobs

The following examples show sample batch jobs for creating and terminating a global buffer pool.

In the following examples, the notation `vr`s or `vr` represents the relevant product version. For further information on product versions, see *Version* in the *Glossary*.

Example 1:

```
// JOB GBPSTART
/*
/* Starts a global buffer pool with the name NATvrGBP, a size of 1 MB and
/* a text block size of 4 KB. The global buffer pool is allocated above 16 MB.
/* The subsystem used is NATv.
/* After the allocation, the job GBPSTART terminates.
/*
// LIBDEF PHASE,SEARCH=SAGLIB.NATLIB
// EXEC NATGBPvr,SIZE=NATGBPvr,PARM='BPN=NATvrGBP,N=(1000)'
/*
// EXEC LISTLOG
/ &
```

Example 2:

```
// JOB GBPRES
/*
/* Starts a global buffer pool with the name GBP, a default size of
/* 100 KB and a text block size of 1 KB. The global buffer pool is allocated
/* below 16 MB. The subsystem used is SAGS.
/* After the allocation, the job GBPRES will wait for further commands.
/* Further commands may be entered using AR command MSG partition-id:
/* the job GBPRES will then prompt for console input.
/*
// LIBDEF PHASE,SEARCH=SAGLIB.NATLIB
// EXEC NATGBPvr,SIZE=NATGBPvr,PARM='BPN=GBP,N=(,BL,1),S=SAGS,R=Y'
/*
// EXEC LISTLOG
/ &
```

Example 3:

```
// JOB GBPSTOP
/*
/* Stops the global buffer pool GPB if it contains no active objects. If it
/* does contain active objects, the operator console will prompt for a reply.
/* Depending on the reply, the shutdown will be forced (Y) or aborted (N).
/* The subsystem used is NATv.
/*
// LIBDEF PHASE,SEARCH=SAGLIB.NATLIB
// EXEC NATGBPvr,SIZE=NATGBPvr,PARM='FSHUT,BPN=GBP'
/*
// EXEC LISTLOG
/ &
```

Example 4:

```
// JOB GBPSTRT2
/* Read commands from SYSIPT:
/*
/* Start two global buffer pools (subsystem ID Nvrs) with names
/* NATGBP1 - size=1024KB and a cache with size 2048KB, and
/* NATGBP2 - size=2048KB without cache.
/* Display all buffer pools of subsystem ID Nvrs.
/*
/* Note: The job does not terminate by itself, but stays resident and waits
/* for operator commands because it owns the data space allocated for
/* buffer pool NATGBP1.
/*
/* If the buffer pools should shut down, wake up the sleeping job by the MSG ←
/* partition ID
/* and enter the parameter "CF=*" to execute the corresponding FSHUTs.
/*
```

```
// LIBDEF PHASE,SEARCH=SAGLIB.NATLIB
// EXEC NATGBPvr,SIZE=NATGBPvr
CREATE,BPN=NATGBP1,S=Nvrs,N=(1M),BPC=2M
CREATE,BPN=NATGBP2,S=Nvrs,N=(2M)
SHOWBP S=Nvrs
/*
FSHUT,BPN=NATGBP1,S=Nvrs
FSHUT,BPN=NATGBP2,S=Nvrs
SHOWBP S=Nvrs
/*
```

Localization

The module NATGBPTX is delivered in source form. It contains all error messages in English in mixed case. The messages can be translated into other languages as required. In this case, the “new” NATGBPTX source module has to be assembled and NATGBPvr has to be relinked.

To issue the global buffer pool messages including their variable parts in upper case, the global buffer pool parameter module NATGBPRM has to be assembled with the UCTRAN parameter set to YES, and NATGBPvr has to be relinked.

To relink NATGBPvr, use the following JCL:

```
// OPTION CATAL,LIST
ACTION NOAUTO,SMAP
PHASE NATGBPvr,*
INCLUDE NATGBPMG
INCLUDE NATGBPRM
INCLUDE NATGBPTX
INCLUDE NATBPMGR
ENTRY CMSTART
/*
```

Messages

Refer to *Natural Global Buffer Pool Manager Messages* in the *Natural Messages and Codes* documentation

32 Common Natural GBP Operating Functions under z/OS and z/VSE

- Global Buffer Pool Manager Parameter Module 242
- Global Buffer Pool Operating Functions 242
- Global Buffer Pool Function Parameters 244
- Examples of NATBUFFER Specifications 250

This document provides a summary of those operating functions of the Natural global buffer pool which are identical under z/OS and z/VSE.

Global Buffer Pool Manager Parameter Module

The global buffer pool parameter module NATGBPRM is used to set global processing options which apply to all functions and buffer pools. The global buffer pool parameter module is delivered in source and object form with all defaults set.

The following parameter is available:

- UCTRAN - Lower/Mixed Case Support

UCTRAN - Lower/Mixed Case Support

This parameter enables or disables the lower/mixed case support for the global buffer pool messages.

UCTRAN=NO	Lower/mixed case support is fully enabled. This is the default value.
UCTRAN=YES	All global buffer pool messages are issued in upper case.

Global Buffer Pool Operating Functions

The following functions are available:

- ADDCACHE - Allocate Cache for an Existing Global Buffer Pool
- CREATE - Create Global Buffer Pool
- DELCACHE - Release Cache of a Global Buffer Pool
- FSHUT - Shut Down Global Buffer Pool
- GLOBALS - Show Global Parameter Settings
- LISTCACHE - List All Global Buffer Pool Caches Owned by Job
- NOP - No-Operation
- REFRESH - Re-initialize Global Buffer Pool
- SHOWBP - Show Existing Buffer Pools
- TERMINATE - Terminate GBP Operating Program
- ZAPS - Display Zaps Applied to GBP



Note: If no function is specified, **CREATE** is assumed when the profile parameter BPNAME is specified, otherwise **NOP** is assumed.

ADDCACHE - Allocate Cache for an Existing Global Buffer Pool

This function adds cache storage to an existing global buffer pool.

CREATE - Create Global Buffer Pool

This function creates a global buffer pool with the specified parameters.

DELCCACHE - Release Cache of a Global Buffer Pool

This function removes the cache storage of a global buffer pool without shutting down the buffer pool itself.

FSHUT - Shut Down Global Buffer Pool

The global buffer pool is shut down, and the storage area is released.

If there are no active objects in the buffer pool, FSHUT is executed immediately.

If there are still active objects in the buffer pool, this will be indicated to the operator. Depending on the setting of the parameter **CONFIRM**, the operator is asked for a confirmation or FSHUT is executed immediately.

GLOBALS - Show Global Parameter Settings

This function shows all global parameter settings, that is, parameters which do not only apply to the statement for which they have been specified.

In addition, the storage key of the global buffer pool(s) is shown.

LISTCACHE - List All Global Buffer Pool Caches Owned by Job

This function lists all global buffer pool caches currently owned by the job.

NOP - No-Operation

This function code particularly can be used to set global parameters.

REFRESH - Re-initialize Global Buffer Pool

With the `REFRESH` command it is possible to re-initialize an already active buffer pool. As no storage allocation takes place, the buffer pool size and location (above or below 16 MB) remain unchanged. However, it is possible to change the text-block size (see `NATBUFFER` parameter).

You should use this function only if the Current Use Count (see *Fields for Buffer Pool Objects* in *SYSBPM Directory Information*) is equal to zero (see warning below) or if the buffer pool has been destroyed.



Caution: If you re-initialize the buffer pool while Natural objects are being executed by active sessions in this buffer pool, the results of the active sessions are unpredictable and Natural may even abend.

SHOWBP - Show Existing Buffer Pools

Displays all buffer pools currently existing.

TERMINATE - Terminate GBP Operating Program

The GBP operating program is terminated. This termination does *not* affect any active global buffer pool.

ZAPS - Display Zaps Applied to GBP

Displays all Zaps applied to the global buffer pool operating program.

Global Buffer Pool Function Parameters

The functions of the Natural GBP operating program can be controlled with the aid of parameters. These parameters can be specified in any sequence. They can be abbreviated so that they are still unique.



Note: If you like to start multiple global buffer pools with an associated cache, you are recommended to use a single job or (under z/OS only) a single started task and to supply the different `CREATE` commands in an input data set. See [Example 4](#) in the section *Natural Global Buffer Pool under z/OS* or [Example 4](#) in the section *Natural Global Buffer Pool under z/VSE*.

The following parameters are available:

- `BPNAME` - Name of Global Buffer Pool
- `BPLIST` - Name of Preload List
- `BPCSIZE` - Buffer Pool Cache Size
- `CC` - Count Condition Code

- CONFIRM - FSHUT Confirmation
- IDLE - Wait Time before Check
- METHOD - Search Algorithm for Allocating Space in Buffer Pool
- NATBUFFER - Buffer Size, Mode, Text Block Size
- RESIDENT - Behavior after Function Execution
- SUBSID - Natural Subsystem ID
- TYPE - Type of Buffer Pool

BPNAME - Name of Global Buffer Pool

BPNAME=*value* is required (except for the `TERMINATE` function). It specifies the name of the global buffer pool to be created.

Value	Explanation
8 bytes	The name of the global buffer pool. Note: If the specified name is shorter than 8 bytes, blanks will be appended to it.
*	For the functions <code>DELCACHE</code> and <code>FSHUT</code> , you may supply a value of "*" to process all buffer pools for the specified Natural subsystem.

BPLIST - Name of Preload List

BPLIST=*value* specifies the name of the preload list.

Value	Explanation
8 bytes	The name of the preload list. Note: If the specified name is shorter than 8 bytes, blanks will be appended to it.

BPCSIZE - Buffer Pool Cache Size

BPCSIZE=*value* specifies the amount of storage (in KB) used to allocate a data space for the buffer pool cache.

Value	Explanation
100 - 2097148	The amount of storage (in KB) used to allocate a data space for the buffer pool cache.



Notes:

1. The cache size can also be specified in units of MB or GB, for example, by specifying 10M for 10 MB.
2. If the `BPCSIZE` parameter is omitted (or set to zero), the buffer pool is not supplied with a cache.
3. A cache is only supported for buffer pools of `TYPE=NAT`.

CC - Count Condition Code

`CC=value` determines whether a condition code is ignored when returned by a command executed by the global buffer pool manager.

Value	Explanation
Y	The condition code returned after command execution is counted for the condition code of the NATGBP <i>vr</i> job. This is the default value.
N	The condition code returned after command execution is ignored. This may lead to a job response code of zero although the command execution failed.

 **Note:** This parameter is valid for all commands.

Example of Command Execution:

The global buffer pool QA82GBP is stopped and restarted with the following command sequence:

```
FSHUT BPN=QA82GBP,S=QA82,CONFIRM=N,CC=N
CREATE BPN=QA82GBP,S=QA82,N=(1024),M=S,BPC=4096,I=60
```

The FSHUT command usually returns a condition code of 20 when it executes and the buffer pool is not active. However, with `CC=N` set, any condition code is ignored. In this case, a job response code greater than zero is only returned if the following CREATE command fails.

CONFIRM - FSHUT Confirmation

`CONFIRM=value` controls the FSHUT behavior if there are still active objects in the buffer pool.

Value	Explanation
Y	A confirmation for the FSHUT function is required from the operator. The operator can decide to abort or to force the FSHUT function. This is the default value.
N	FSHUT is forced without interaction with the operator.

 **Note:** This parameter is only valid for the FSHUT command it has been specified with, that is, CONFIRM has to be specified with each FSHUT parameter, and it does not apply to subsequent FSHUT commands.

IDLE - Wait Time before Check

IDLE=*value* is ignored when the task does not own a buffer pool cache.

Value	Explanation
Numeric	The number of seconds to elapse before the GBP operating program checks for each buffer pool cache if its associated buffer pool is still active; if not, that buffer pool cache is released; when the last buffer pool cache owned by the task has been released, the task terminates, unless RESIDENT=Y has been specified.
60	This is the default value.

**Notes:**

1. IDLE is a “global” parameter. Once specified, IDLE will also apply to subsequent commands, without your having to specify it again.
2. Under z/OS, the GBP operating program also checks the specified IDLE time value against the job's timeout value: the specified IDLE time value internally may reduce IDLE to prevent timeout abends (S322).

METHOD - Search Algorithm for Allocating Space in Buffer Pool

METHOD=*value* controls which algorithm is to be used for allocating storage in the Natural buffer pool.

Value	Explanation
S	Indicates that a selection process is to be used for allocating storage. The selection process consists of browsing the whole buffer pool directory and comparing different entries in order to find a most suitable entry. This method was formerly known as algorithm 1+2. This is the default value.
N	Indicates that the next available unused or free space is to be used. The search for the next available space is done from a pointer to directory entries which moves in a wrap-around fashion. This method may be used in combination with a buffer pool cache.



Note: This parameter is only valid for the **CREATE** function. If you want to change the allocation method, restart the buffer pool.

NATBUFFER - Buffer Size, Mode, Text Block Size

NATBUFFER=(*size,mode,tsize*) specifies the size and the mode of the buffer pool, and the text block size.

Syntax	Value	Explanation
NATBUFFER=(<i>size,mode,tsize</i>)	<i>size</i>	<p>is the amount of storage (in KB) to be allocated.</p> <p>For the Natural buffer pool (TYPE=NAT), the default and minimum possible size is 256 KB.</p> <p>For the other buffer pools, the default and minimum possible size is 100 KB.</p> <p>The specified amount of storage is always rounded up to a multiple of 4 KB.</p> <p>The pool size can also be specified in units of MB or GB, e.g. by specifying 10M for 10 MB.</p> <p>Next to the storage specified by <i>size</i>, one page (4 KB) of write protected storage will be allocated for administrative purposes.</p>
	<i>mode</i>	<p>determines if the global buffer pool is to be allocated above or below 16 MB.</p> <p>Possible values are: XA = above (default), BL = below.</p>
	<i>tsize</i>	<p>determines the text block size (in KB).</p> <p>Possible values are: 1, 2, 4, 8, 12, and 16. The default value is 4.</p>
<p><i>size, mode</i> and <i>tsize</i> have to be specified in the sequence shown above.</p>		



Note: If NATBUFFER is not specified, the default values will be used. See also [Examples of NATBUFFER Specifications](#).

RESIDENT - Behavior after Function Execution

RESIDENT=*value* specifies the behavior of the GBP operating program after the specified function has been executed. The following values are possible:

Value	Explanation
Y	The GBP operating program will remain active after executing the specified function and await further commands. Once specified, <code>RESIDENT=Y</code> will also apply to subsequent commands, without your having to specify it again. (To stop the GBP operating program, you use the TERMINATE function.)
N	The GBP operating program will terminate after executing the specified function, if no further command is available. If the task owns a buffer pool cache, <code>RESIDENT=N</code> is ignored and the task is not terminated.
A	<p>The GBP operating program automatically decides how to behave after having processed all commands. It will terminate if</p> <ul style="list-style-type: none"> ■ no further command is available and ■ no buffer pool with an associated cache exists that was created by this task. <p>In other words: If no buffer pool cache is owned by the task, <code>RESIDENT=A</code> works in the same way as <code>RESIDENT=N</code>. When the task owns a buffer pool cache, <code>RESIDENT=A</code> works the same way as <code>RESIDENT=Y</code>, but switches automatically to <code>RESIDENT=N</code>, when the last buffer pool whose associated buffer pool cache was owned by this task has terminated.</p> <p>This is the default setting.</p>



Note: `RESIDENT` is a “global” parameter. Once specified, `RESIDENT` will also apply to subsequent commands until explicitly specified/overwritten.

SUBSID - Natural Subsystem ID

`SUBSID=value` specifies the ID of the Natural subsystem.

Value	Explanation
4 bytes	<p>The 4-byte ID of the Natural subsystem.</p> <p>Once specified, <code>SUBSID</code> will also apply to subsequent commands, without your having to specify it again.</p> <p>The default value is <code>NAT v</code>, where <code>v</code> is the first digit of the current Natural version.</p>
<code>NAT v</code>	This is the default value. <code>v</code> is the first digit of the current Natural version.



Notes:

1. `SUBSID` is a “global” parameter, that is, once specified, `SUBSID` will also apply to subsequent commands until explicitly specified/overwritten.
2. For the functions [DELCACHE](#), [FSHUT](#) and [SHOWBP](#), you may supply a value of "*" to process all buffer pools for the specified Natural subsystem.
3. For further information on the Natural subsystem, see [Natural Subsystem \(z/OS\)](#) or [Natural Subsystem \(z/VSE\)](#).

TYPE - Type of Buffer Pool

TYPE=*value* specifies the type of the buffer pool. Possible values are:

Value	Explanation
NAT	Natural buffer pool (this is the default).
SORT	Sort buffer pool.
DLI	DL/I buffer pool.
EDIT	Editor buffer pool.
MON	Monitor buffer pool.
RNM	Review Natural Monitor buffer pool.

Examples of NATBUFFER Specifications

The following examples refer to the `NATBUFFER` parameter which is used to set buffer size, mode and text block size, the parameter name being abbreviated (N).

Example 1: To allocate a global buffer pool above 16 MB, with a size of 1 MB and a text block size of 1 KB, you specify:

```
N=(1000,,1)
```

or

```
N=(1M,,1)
```

Example 2: To allocate a global buffer pool above 16 MB, with a size of 10 MB and a text block size of 4 KB, you specify:

```
N=(10000)
```

or

```
N=(10M)
```

Example 3: To allocate a global buffer pool above 16 MB, with a size of 256 KB and a text block size of 4 KB, you specify:

N=(, ,)

This is equivalent to omitting the `NATBUFFER` parameter altogether, as it causes the default values to apply.

33

Natural Global Buffer Pool under BS2000

- Using a Natural Global Buffer Pool under BS2000 254
- Establishing the Global Buffer Pool under BS2000 254
- Administering the Global Buffer Pool under BS2000 255

This document describes purpose and usage of a Natural global buffer pool (GBP) under the operating system BS2000.

In the examples below, the notation *vrs* or *vr* represents the relevant product version. For further information on product versions, see *Version* in the *Glossary*.

Using a Natural Global Buffer Pool under BS2000

The Natural global buffer pool is a common memory pool that can be used with BS2000 Version 10.0 and above.

On XS31 computers, it can be located either below 16 MB or in the extended address space above 16 MB. On non-XS31 computers, it can be located in the user address space below Class 4 storage (whose size depends on how the operating system was generated).

The global buffer pool can be used by several Natural under TIAM, Natural under *openUTM* and batch applications simultaneously. It is possible to have more than one global buffer pool per operating system.

The global buffer pool has to be activated before the first Natural application is started. It can remain active as long as the operating system is active, even after the last Natural session has been terminated. This means that the global buffer pool's contents are still available when a new session is started and need not be loaded into the buffer pool again.

Establishing the Global Buffer Pool under BS2000

The global buffer pool is established by executing a batch job which starts the program **CMPSTART**. The global buffer pool's name, size, virtual address, etc. are determined by parameters specified in this job.

Example of CMPSTART Job:

```
/SYSFILE SYSOUT.LST.BPvrsGA
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (CMPSTART,$NATvrs.NATvrs.BS2.MOD)
NAME=BPvrsGA,TYPE=NAT,POSI=ABOVE,SIZE=2MB,ADDR=260,PFIX=NO,SCOP=GLOBAL
/SYSFILE SYSDTA=(PRIMARY)
```

If the parameter values are invalid or do not match the BS2000 environment, the buffer pool task is terminated with an error message. The error message contains the reason for the termination and (if applicable) the SVC return code. All error messages are output on *SYSOUT*. In the case of grave errors, they are also displayed on the operator console.

Administering the Global Buffer Pool under BS2000

Once the global buffer pool is active, it is administered via the operator console.

The following BS2000 console commands are available:

Command	Function
/INTR <i>tsn</i> , DPRM	Displays the current parameters settings and the start time of the global buffer pool.
/INTR <i>tsn</i> , SHUT /INTR <i>tsn</i> , STOP	Terminates the buffer pool task normally.
/INTR <i>tsn</i> , DUMP	Terminates the buffer pool task abnormally and produces a dump.

Where *tsn* is the TSN of the buffer pool task.

The termination of the buffer pool task does not necessarily mean the termination of the global buffer pool, as the common memory pool remains active until the end of the last Natural application.

You can also terminate global buffer pools by executing the `CMPEND` program, for example:

```
/PROC C
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (CMPEND,NATvrs.MOD)
name /* name of the global buffer pool
/SYSFILE SYSDTA=(PRIMARY)
/ENDP
```


VIII

Message Buffer Pool

34 Message Buffer Pool

- Purpose 260
- Prerequisites 260
- Operating the Message Buffer Pool 260
- Sample NATMBPvr Execution Jobs 262
- Message Buffer Pool Operating Functions 263
- Function Parameters 264
- Messages 265

This part describes the use of the message buffer pool.



Note: The message buffer pool is available under z/OS.

Purpose

The message buffer pool is a cache memory which is used to store the Natural system messages and the user texts.

Before an error message is output, Natural first checks whether the corresponding message text is available in the message buffer pool. If so, this text is output. Otherwise, the error message would be read from the database, and would be stored in the message buffer pool.

The message buffer pool is available only as a global buffer pool. Its use is optional, and is controlled by the Natural profile parameter `BPI` or the corresponding macro `NTBPI`. When used, the message buffer pool is allocated in a data space.

Prerequisites

The following prerequisites must be met if you want to use the message buffer pool:

1. The module `NATMBP` *vr* must have been linked into an Authorized Program Facility (APF) library; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
2. The message buffer pool must have been created and started; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
3. The keyword subparameter `TYPE` of profile parameter `BPI` or macro `NTBPI` must be set to `MSG`.

Operating the Message Buffer Pool

The message buffer pool is operated by the program `NATMBP` *vr* which must be executed from within an Authorized Program Facility (APF) library.

The following topics are covered below:

- [Setting up the Message Buffer Pool](#)
- [Starting the Message Buffer Pool Operating Program](#)

- [Stopping the Message Buffer Pool Operating Program](#)



Note: In the following document, *vrs* or *vr* represents the relevant version of the product. For information on product versions, see *Version* in the *Glossary*.

Setting up the Message Buffer Pool

The **functions** available from NATMBP *vr* (see also [Function Parameters](#)) are activated in that they are

- provided by a parameter card (PARM=),
- read from a file (see below),
- or supplied by the MODIFY operator command unless NATMBP *vr* has not been terminated.

NATMBP *vr* expects the first command in the parameter field (PARM=) of the EXEC statement.

You may enter:

- one of the functions described in the section [Common Message Buffer Pool Operating Functions](#),
- or a reference to an input file with CF=<dd-name>, where <dd-name> represents a DD name defined in the JCL.

Only “card image” files are supported; that is, RECFM=F, LRECL=80, and only the first 72 bytes of the input record are honored.

Every record included from the input file represents a command.

Blank records or records prefixed with an asterisk (*) are ignored.

A file is processed until End-Of-File (EOF).

Example: PARM='CF=SYSIN1'

If the parameter field is not supplied or blank, the commands will be read from file SYSIN by default.

It is only possible to enter one function at a time at the console, or one function per line using the command file, otherwise an error message will be returned.

Each command received from parameter card, from file input or from operator console input is displayed on the operator console.

Starting the Message Buffer Pool Operating Program

To start the program NATMBP vr , either start a started task or submit a job which executes NATMBP vr .

Stopping the Message Buffer Pool Operating Program

The program NATMBP vr is stopped by using the TERMINATE function (see [Common Message Buffer Pool Operating Functions](#)) or, in case of emergency, by using the CANCEL operating program.

Sample NATMBP vr Execution Jobs

The following examples show sample batch jobs for creating and terminating a global buffer pool.

- [Example 1](#)
- [Example 2](#)
- [Example 3](#)



Note: In the following examples, v , vrs or vr represents the relevant version of the product. For information on product versions, see *Version* in the *Glossary*.

Example 1

```
//MBPSTART JOB
//*
//* Starts a message buffer pool with the name NAT $vr$ MBP and
//* a size of 10 MB.
//* The subsystem used is NAT $v$ .
//*
//STEP EXEC PGM=NATMBP $vr$ ,PARM='BP=NAT $vr$ MBP,SI=10'
//SETPLIB DD DISP=SHR,DSN=USER.APF.LINKLIB
```

Example 2

```
//MBPRES JOB
//*
//* Starts a message buffer pool with the name MBP and a default size of
//* 100 MB. The subsystem used is SAGS.
//*
//STEP EXEC PGM=NATMBP $vr$ ,PARM='BP=MBP,S=SAGS'
```

Example 3

```
//MBPSTRT2
/* Read commands from SYSIN1:
/*
/* Start 2 message buffer pools (subsystem ID Nvrs) with name
/*   NATMBP1 - size=1000MB
/*   NATMBP2 - size=2000MB
/* If the buffer pools should shut down, send operator command MODIFY with
/* parameter "CF=SYSIN2" to execute the corresponding FSHUTs.
/*
//STEP EXEC PGM=NATMBPvr,PARM='CF=SYSIN1'
//SYSIN1 DD *
CREATE,BP=NATMBP1,S=Nvrs,SI=1000M
CREATE,BP=NATMBP2,S=Nvrs,SI=2000M
SHOWBP S=Nvrs
//SYSIN2 DD *
FSHUT,BP=NATMBP1,S=Nvrs
FSHUT,BP=NATMBP2,S=Nvrs
/*
```

Message Buffer Pool Operating Functions

The following functions are available:

- **CREATE** - Create a Message Buffer Pool
- **FSHUT** - Shut Down Message Buffer Pool
- **TERMINATE** - Terminate Message Buffer Pool Operating Program
- **ZAPS** - Display Zaps Applied to Message Buffer Pool



Note: The function names can be abbreviated. It is sufficient to use the first character only, for example T for TERMINATE.

CREATE - Create a Message Buffer Pool

This function creates a message buffer pool with the specified [parameters](#).

FSHUT - Shut Down Message Buffer Pool

The message buffer pool with the specified **parameters** is deallocated.

TERMINATE - Terminate Message Buffer Pool Operating Program

The message buffer pool operating program is terminated. Prior to that, all active message buffer pools are deallocated.

ZAPS - Display Zaps Applied to Message Buffer Pool

Displays all Zaps applied to the message buffer pool operating program.

Function Parameters

The functions of the message buffer pool operating program can be controlled with the aid of parameters. These parameters can be specified in any sequence. They can be abbreviated.

The following parameters are available:

BPNAME **Name of message buffer pool.**

BPLIST **Name of the preload list (optional).**

SUBSID **Natural subsystem ID.**

SIZE **Size of the message buffer pool.**



Note: The underlined part of the parameter name marks the shortest possible abbreviation.

BPNAME - Name of Message Buffer Pool

BPNAME=*value* specifies the name of the message buffer pool to be created.

Value	Explanation
8 bytes	The name of the message buffer pool. Note: If the specified name is shorter than 8 bytes, blanks will be appended to it.
MTBP	This is the default value.

BPLIST - Name of Preload List

BPLIST=*value* specifies the name of the optional preload list.

Value	Explanation
8 bytes	The name of the preload list. Note: If the specified name is shorter than 8 bytes, blanks will be appended to it. There is no default value.

SUBSID - Natural Subsystem ID

SUBSID=*value* specifies the ID of the Natural subsystem.

Value	Explanation
4 bytes	The ID of the Natural subsystem. Note: If the specified name is shorter than 8 bytes, blanks will be appended to it.
NAT <i>v</i>	This is the default value, where <i>v</i> is the first digit of the current Natural version.

SIZE - Size of Message Buffer Pool

SIZE=*value* specifies the size of the message buffer pool.

Value	Explanation
1 - 2000 MB	The size of the message buffer pool.
100	This is the default value.

Messages

Refer to *Message Buffer Pool Messages* in the *Messages and Codes* documentation.

IX

Optimize Monitor Buffer Pool

35

Optimize Monitor Buffer Pool

- Purpose 270
- Prerequisites 270
- Starting the Optimize Monitor Buffer Pool 271
- Operator Commands 271
- Messages 272

The Software AG product ETS Optimize for Infrastructure enables you to monitor all Software AG component resources in real time. A global buffer pool, in the following called Optimize Monitor Buffer Pool, is available in Natural to provide system and operational data for monitoring all Natural components running in one LPAR (z/OS), on one machine (z/VSE) or on one host (BS2000).

This part describes the use of the Optimize Monitor Buffer Pool.

Purpose

All Optimize for Infrastructure Key Performance Indicator (KPI) data is collected in an Optimize Monitor Buffer Pool per LPAR (z/OS), machine (z/VSE) or host (BS2000). This buffer pool actually is a data space.

It consists of the following parts:

- **Pool Header**

The pool header holds control information and statistical data.

- **KPI Pool**

The KPI pool holds all instances with their KPI data slots.

- **Session Data Pool**

This is the pool where all sessions put their session data per dialog step.

The session data pool is written to in a wrap-around manner. Session data is aggregated by a back-ground task (aggregation daemon) into the associated KPI slots.

Prerequisites

The following prerequisites must be met if you want to use the Optimize Monitor Buffer Pool:

1. The Optimize Monitor Buffer Pool must have been created and started; see the corresponding step in *Installing Natural on z/OS* in the *Installation* documentation.
2. The profile parameter 04I must be set to ON.

Starting the Optimize Monitor Buffer Pool

You start the Optimize Monitor Buffer Pool either as a batch job or as a started task by executing module `NAT04Ivr` (where *vr* represents the relevant product version).

➤ To start the Optimize Monitor Buffer Pool

- Specify the following parameters as `PARM` in the `JCL EXEC` statement (z/OS and z/VSE) or in the `SYSDTA` logical system file (BS2000):

```
bp-size,subpool-size,daemon-idle-time,filled-threshold
```

All parameters are positional and must be separated by a comma. They are explained in the table below:

<i>bp-size</i>	Total Optimize Monitor Buffer Pool size in KB. Can also be specified in units of MB or GB by adding a trailing M or G at the end of the size value.	
<i>subpool-size</i>	Size of KPI sub-pool.	
<i>daemon-idle-time</i>	Aggregation daemon idle time in seconds. Valid Values:	
	1 - 60	This is the time interval the aggregation daemon will regularly wake up and aggregate the session data into KPI data.
	30	This is the default value.
<i>filled-threshold</i>	Session data pool "filled" threshold, when the aggregation daemon is triggered for immediate wake-up. Valid values:	
	20 - 80	"Filled" threshold in percent.
	50	This is the default value.

Operator Commands

Notation:

The operator commands can be specified in any length from minimum (capital letter) up to full command name length.

End

Terminates the Optimize Monitor Buffer Pool.

In z/OS, the Optimize Monitor Buffer Pool can also be terminated via the operator command `STOP`.

Halt

Stops the aggregation daemon.

Start

Restarts the aggregation daemon.

When restarted, the aggregation daemon starts from session data pool start.

Idle=nn

Sets the aggregation daemon idle time. *nn* = value in seconds.

Trigger=nn

Sets the aggregation daemon trigger threshold. *nn* = value in percent

Messages

Refer to *Optimize Monitor Buffer Pool Messages* in the *Messages and Codes* documentation.

X Natural Swap Pool

This part provides information on the Natural swap pool which is available when you are using either of the following TP monitors:

- **CICS**
(where the Natural swap pool is optional)
- **openUTM**
(where the Natural swap pool is necessary)

The behavior and the functionality of the Natural swap pool is to a large extent identical in these environments. However, differences or TP-monitor-specific features exist. These are marked accordingly in the following texts.

Purpose of a Natural Swap Pool

Natural Swap Pool Operation

Swap Pool Initialization

Dynamic Swap-Pool Reorganization

Defining the Natural Swap Pool

Natural User Area Size Considerations

Swap Pool Data Space

Global Restartable Swap Pool under *openUTM*

Terminating the Global Swap Pool

Related Topics:

- *Natural Swap Pool under CICS* in the *TP Monitor Interfaces* documentation
- *Using the Natural Swap Pool under CICS* in the *TP Monitor Interfaces* documentation
- *Natural Swap Pool under openUTM* in the *TP Monitor Interfaces* documentation
- *Error Messages from the Natural Swap Pool Manager Valid under CICS and openUTM* in the *Messages and Codes* documentation

36 Purpose of a Natural Swap Pool

- Purpose of a Natural Swap Pool 276
- Benefits of Using a Natural Swap Pool 276
- Swap Pool Structure 276

This document describes the purpose, benefits and structure of a Natural swap pool.

Purpose of a Natural Swap Pool

A Natural user work area is required for each online Natural user. The size of this work area is determined by the parameter `MAXSIZE` in the macro `NTSWPRM`.) The user work area must be in the computer's main storage whenever the user initiates any form of dialog transaction.

In order to reduce the frequency with which the user work area is rolled out to the swap file (or roll facility under CICS) and rolled in again, it is possible to set up a Natural swap pool.

For more details, refer to [Natural Swap Pool Theory of Operation](#).

Benefits of Using a Natural Swap Pool

The user work areas are held in the Natural swap pool in compressed form as much as possible. The amount by which disk swapping is reduced depends upon the size of the swap pool, the size of each compressed Natural user work area and the number of online users.

If the user work areas of all the online users can be kept resident in the swap pool, no disk swapping takes place.

TP Monitor	Comment
CICS	The size, name and cache size of the swap pool are specified using profile parameter <code>BPI</code> or the corresponding macro <code>NTBPI</code> in the Natural parameter module , that is, the <code>(NT)BPI</code> settings in effect for the Natural session initializing the Natural CICS environment are taken.
<i>openUTM</i>	The size of the Natural swap pool is specified with the keyword parameter <code>SIZE</code> in the macro <code>ADDON</code> or by program <code>CMPSTART</code> (see also the keyword parameters <code>DATA</code> and <code>DESA</code> for the generation of swap pool data space).

Swap Pool Structure

The physical swap pool is made up of the following parts:

- Main directory
- Logical swap pools with
 - Subdirectories
 - Swap pool slots

Swap Pool Main Directory

The swap pool main directory refers to the entire swap pool. Up to 15 logical swap pools can be defined.

Subdirectories

Each logical swap pool has its own subdirectory.

Swap Pool Slots

In the swap pool slots, the Natural user work areas are held in compressed form.

For the first initialization of the swap pool, the number of logical swap pools and the size of their slots can be defined with the parameter `SWPSLSZ` in the macro `NTSWPRM` to generate the swap pool parameter module.

Logical Swap Pools

Each logical swap pool contains a subdirectory and a guest table.

Each swap pool directory entry used is chained to its predecessor entry and successor entry. This is also true for the entries in the guest table. In this way, the most recent and the oldest swap pool users/guests are always known.

To define a guest in a logical swap pool, proceed as in the following example:

There is a swap pool with three logical swap pools (LSPs).

- LSP 1 has a slot size of 62 KB.
- LSP 2 has a slot size of 72 KB.
- LSP 3 has a slot size of 82 KB.

The size of the compressed Natural user work area is 60 KB and therefore, this user work area should be compressed into a slot of the logical swap pool 1. If LSP 1 is currently full (which is the case in the above example) and LSP 2 contains a free slot, the user work area will be compressed into LSP 2; if it is full, and LSP 3 contains a free slot, the user work area will be compressed into LSP 3. A user work area in LSP 2 or 3 is a guest in these LSPs because its own LSP was full.

TP Monitor	Comment
CICS	The Natural swap pool is optional under CICS. Due to CICS command-level overhead, swapping into or from the swap pool is faster than expensive roll I/Os. Nevertheless, if virtual storage is a bottleneck, the installation of a swap pool may lead to performance degradations due to paging overhead; see also <i>Natural Swap Pool under CICS</i> and <i>Using the Natural Swap Pool under CICS</i> in the <i>Natural TP Monitor Interfaces</i> documentation.
<i>openUTM</i>	The Natural swap pool is necessary under <i>openUTM</i> . See also <i>Natural Swap Pool under openUTM</i>

37

Natural Swap Pool Operation

- Users are On their Way to Natural - No Session Start 280
- Users are Returning from Natural 280

Users are On their Way to Natural - No Session Start

If the user's work area is held in the swap pool, the corresponding slot is read and decompressed into the Natural user thread. The corresponding swap pool directory entry is unlinked from the directory chain and declared as a free entry. If it was a guest, the guest table will be updated.

If the user's work area is not held in the swap pool, it is read and decompressed from the Data Space or from the swap file (or roll facility under CICS) into the Natural user thread.

Natural is activated.

Users are Returning from Natural

Natural checks whether the compressed length of the user work area exceeds the highest slot size of the logical swap pools.

If it exceeds the highest slot size, the user work area is compressed and written asynchronously to the swap file (or rolled to the roll facility, which is associated with the session under CICS).

If it does not exceed the highest slot size, Natural finds out whether there is a free slot in the user work area's own swap pool:

- If there is a free slot, the user work area is compressed into this slot. The corresponding directory entry is linked into the directory chain as latest entry.
- If there is no free slot, Natural finds out whether there are guests in the user work area's own logical swap pool.

If there are one or more guests, a slot is made available: The oldest guest-table entry is unlinked from the guest table and the until then second oldest is made oldest guest. The adequate directory entry is unlinked from the directory chain.

If there are no guests, a slot is made available: The oldest directory entry is unlinked from the directory chain and the until then second oldest is made oldest.

- If ESA Data Space is generated and there is a free slot available, this slot will be used before a thread will be rolled out into a swap file.

The compressed user area of the unlinked user is transferred to the write buffer and written asynchronously to the swap file (or rolled synchronously to the roll facility, which is associated with the session under CICS). The current user's work area is compressed into the slot which has become available. The corresponding directory entry is linked to the directory chain as latest entry.

The statistics tables for swap pool reorganization and slot size calculation are updated.

38

Natural Swap Pool Initialization

- Swap Pool Initialization Control 282
- Swap Pool Initialization Parameters 283

This document describes how to control the initialization of a Natural swap pool and contains an overview of the keyword parameters available for initialization in the macro `NTSWPRM`.

Swap Pool Initialization Control

The parameter `SWPINIT` in the macro `NTSWPRM` controls the initialization of the swap pool.

If You Set `SWPINIT=AUTO`

- The swap pool manager tries to read the swap pool initialization data with the swap pool name as key from the Natural system file `FNAT` or `FUSER` (see keyword parameters `SWPFILE` of macro `NTSWPRM`). If it finds data, they are used and the corresponding parameters in the macro `NTSWPRM` are ignored. If it does not find data, the operand(s) of the keyword parameter `SWPSLSZ` in the macro `NTSWPRM` will be used for initializing the swap pool.
- If the parameter `SWPSLSZ` contains only one slot size definition, the swap pool is initialized with one logical swap pool. In the specified time interval (see parameter `SWPTIM1` in the macro `NTSWPRM`), the swap pool manager controls whether the swap pool needs to be reorganized or optimized (see also the section *Dynamic Swap-Pool Reorganization*). If the swap pool was reorganized, the newly calculated initialization data for the swap pool are stored in the Natural system file for the next initialization. If the swap pool's reorganization has resulted in more than one logical swap pool, there will be no further dynamic swap pool reorganization.
- Dynamic swap pool reorganization is not possible when the swap pool contains more than one logical swap pool.
- Further swap pool optimizations can be explicitly initialized with the following Natural `SYSTP` utility functions:
 - Slot Size Calculation,
 - Swap Pool Parameter Service (modification of the swap pool initialization data in the Natural system file),
 - Deactivate the Swap Pool and Activate the Swap Pool.
- The maximum number of logical swap pools for dynamically reorganizing or optimizing the swap pool can be defined in the operand of the keyword parameter `SWPLSWP` in macro `NTSWPRM`.

If You Set SWPINIT=

- No swap pool initialization data in the Natural system file will be read or stored. The operand(s) of the keyword parameter `SWPSLSZ` in the macro `NTSWPRM` will be used for initializing the swap pool.
- The rules for dynamically reorganizing or optimizing the swap pool are the same as described under `SWPINIT=AUTO` above, except that no initialization data will be stored in the Natural system file.

Swap Pool Initialization Parameters

The following is an overview of the keyword parameters that are available for initialization in the macro `NTSWPRM`.

Parameter	Explanation
SWPSLSZ	Defines the number of logical swap pools, their slot sizes and the numerical relation between slot number and logical swap pools.
SWPINIT	Defines the access to the swap pool initialization data through the Natural system file.
SWPLSWP	Defines the maximum number of logical swap pools for reorganizing swap pools dynamically.
SWPSDIF	Defines the even-numbered minimum difference between the slot sizes of the different logical swap pools. This value will be controlled during slot-size calculation and dynamic swap-pool reorganization.

The following TP-monitor-specific requirements apply:

- **Under openUTM:**
The size of the swap pool must be specified in the operand of keyword parameter `SIZE` for macro `ADDON` or program `CMPSTART`.
- **Under CICS:**
The size, name and cache size of the swap pool are specified using profile parameter `BPI` or the corresponding macro `NTBPI` in the [Natural parameter module](#), that is, the `(NT)BPI` settings in effect for the Natural session initializing the Natural CICS environment are taken.

39

Dynamic Swap-Pool Reorganization

- Requirements for Dynamic Swap-Pool Reorganization 286
- Statistics Tables 286
- Swap-Pool-Reorganization Plus Table 286
- Swap-Pool-Reorganization Minus Table 287
- Parameters for Swap-Pool Reorganization 287
- Checking for the Necessity of Swap-Pool Reorganization 288
- Flow of Dynamic Swap-Pool Reorganization 288
- Start of Dynamic Swap-Pool Reorganization 289

This document describes the prerequisites, process, control and start of a dynamic swap pool reorganization.

Requirements for Dynamic Swap-Pool Reorganization

Dynamic swap pool reorganization is only possible when the physical swap pool contains only one logical swap pool. In this case, the swap pool slots are all of the same size. If necessary, the number of logical swap pools and the slot sizes can be adjusted to meet the requirements. Slot sizes are adjusted by reorganizing the swap pool dynamically.

Statistics Tables

The statistical area of the swap pool directory contains two statistics tables which are used for swap pool reorganization:

- swap-pool-reorganization plus table
- swap-pool-reorganization minus table

Swap-Pool-Reorganization Plus Table

The swap-pool-reorganization plus table contains information on the Natural user areas which could not be placed into the swap pool because their compressed length exceeded the swap-pool slot size.

The table contains 11 entries:

- The first 9 entries count the number of user areas whose length exceeded the slot size by 1 to 9 units.
- The 10th entry counts the number of user areas whose length exceeded the slot size by more than 9 units.
- The 11th entry contains the average length of those user areas counted by the 10th entry.

Swap-Pool-Reorganization Minus Table

The swap-pool-reorganization minus table contains information on the Natural user areas whose compressed length was **smaller** than the swap-pool slot size.

The table contains 11 entries:

- The first 9 entries count the number of user areas whose length was smaller than the slot size by 1 to 9 “units”.
- The 10th entry counts the number of user areas whose length was smaller than the slot size by more than 9 units.
- The 11th entry contains the average length of those user areas counted by the 10th entry.

The size of a “unit” is defined with the keyword parameter `SWPFACT`.

Parameters for Swap-Pool Reorganization

Dynamic swap-pool reorganization is controlled via the following keyword parameters in the macro `NTSWPRM`.

Parameter	Specifies
SWPSLSZ	the slot size for the first initialization of the swap pool. The default size is 62 KB.
SWPTFIX	if the slot size is to be fixed or dynamic. With fixed slot size, there is no dynamic swap pool reorganization. If the slot size is defined as not fixed, the swap pool is dynamically reorganized when necessary (this is the default).
SWPTIM1	the time interval at which a check is to be performed to ascertain whether a swap pool reorganization is necessary. By default, the check is performed every 30 minutes.
SWPTIM2	the time to elapse after the check for the necessity of a swap pool reorganization is performed and before the reorganization is to be started. By default, a reorganization is started 2 minutes after a check has proved a reorganization to be necessary.
SWPUSER	the rate of compressed user threads (in percent) which are too long for the actual SWP slot length. If this value is reached and the physical SWP contains only one logical swap pool, an SWP reorganization will be announced.
SWPFACT	the factor for a “unit” in the swap pool reorganization plus table and minus table.

There is no need to change the default values for any of these parameters (unless you feel that slot size optimization is not performed efficiently enough).

For testing and optimizing, you can dynamically change the values for these parameters online using the Natural Swap Pool Manager, which is part of the Natural utility `SYSTP`.

Checking for the Necessity of Swap-Pool Reorganization

The check is based on:

- the overall number of dialog steps during the time between two checks;
- the percentage defined with the `SWPUSER` parameter;
- the maximum number of logical swap pools defined with the `SWPLSWP` parameter;
- the minimum difference of slot sizes for different logical swap pools;
- the values of the swap-pool reorganization plus and minus tables (these tables are influenced by the setting of the `SWPFACT` parameter);
- the total size of the physical swap pool.

The number of necessary logical swap pools with the corresponding slot sizes will be computed if the number of user areas whose compressed length was greater or smaller (by at least one unit) than the current slot size is more than n percent of the number of dialog steps (n being the value of the `SWPUSER` parameter).

When the swap pool is reorganized, the new logical swap pools are used. If the physical swap pool contains more than one logical swap pool after the reorganization, there will be no further dynamic swap-pool reorganization.

Flow of Dynamic Swap-Pool Reorganization

Natural will only check whether the swap pool needs to be reorganized if the physical swap pool contains no more than one logical swap pool.

Once the time specified with the `SWPTIM1` parameter has elapsed, a check is performed to determine whether a swap-pool reorganization is necessary.

- If swap-pool reorganization is not necessary, the timer set with the `SWPTIM1` parameter (time interval between checks) is activated again.
- If swap-pool reorganization is found necessary, the timer set with the `SWPTIM2` parameter (time interval between end of check and start of reorganization) is activated: no further user areas can be placed in the swap pool; user areas held in the swap pool can still be used and read into the user thread. Once this second time interval has elapsed, swap-pool reorganization is started.

Start of Dynamic Swap-Pool Reorganization

After the time specified with the `SWPTIM2` parameter has elapsed, the swap pool is reorganized while the current online session continues:

1. The compressed user areas which are still held in the swap pool are written to the swap file (or roll facility under CICS).
2. The contents of the swap-pool-reorganization statistics tables are written to `SYSLST` and then deleted from the tables.
3. The swap-pool is re-initialized with the newly computed values.
4. The timer set with the `SWPTIM1` parameter (time interval between checks) is activated again.

The Natural swap-pool manager, which is part of the Natural utility `SYSTP` (see the Natural *Utilities* documentation), can be used to obtain information on swap pool statistical data, sizes of Natural buffers and user threads.

40

Defining the Natural Swap Pool

- Environment-Specific Requirements 292
- Keyword Parameters of Macro NTSWPRM 292

This document describes the TP monitor environment-specific requirements that apply and the keyword parameters can be used to define the Natural swap pool.

Environment-Specific Requirements

The following environment-specific requirements apply:

- **Under openUTM:**

The Natural swap pool is defined by specifying macro `NTSWPRM` for assembling the Natural swap-pool parameter module.

- **Under CICS:**

The Natural swap pool is defined by specifying `NTSWPRM` in the `NCISPCB` environment definition module or in the **Natural parameter module**. If the `NTSWPRM` macro is specified both in the `NCISPCB` environment definition module and in the **Natural parameter module**, the specification in `NCISPCB` takes precedence over the specification in the Natural parameter module.

Keyword Parameters of Macro NTSWPRM

The following keyword parameters can be used to define the Natural swap pool details:

`DSPCONT` | `DSPLIFE` | `LABEL` | `MAXLOCK` | `MAXSIZE` | `NOVPA` | `NOVPW` | `SWPCOPT` | `SWPFACT` | `SWPFILE` | `SWPINIT` | `SWPLSWP` | `SWPPWRD` | `SWPSDIF` | `SWPSLSZ` | `SWPTFIX` | `SWPTIM1` | `SWPTIM2` | `SWPUSER` | `WAITMS` | `WRITMS` |

DSPCONT - Minutes for Data Space Slot Control

This parameter defines the time (in minutes) after which data space control takes place when the ESA Data Space area is full. When this time has elapsed, the slots in the Data Space are checked for whether their threads' life time has expired. If so, the compressed Natural user thread of each affected slot is rolled out into the roll file.

<code>DSPCONT=nnn</code>	<code>nnn</code> must be in the range from 1 to 480.
<code>DSPCONT=10</code>	The default value is 10 (minutes).

DSPLIFE - Life Time in Minutes for a Thread in the ESA Data Space

This parameter defines the life time for a compressed Natural user thread in a slot of the ESA Data Space. When the data space slots control logic becomes active, the thread is rolled out if its life time has elapsed. The life time of a thread starts when the thread is written to the ESA Data Space.

DSPLIFE= <i>nn</i>	<i>nn</i> must be in the range from 1 to 60.
DSPLIFE=5	The default value is 5 (minutes).

LABEL - Name of Swap-pool Parameter Module

This parameter defines the CSECT name of the swap-pool parameter module.

LABEL= <i>nnnnnnnn</i>	The name <i>nnnnnnnn</i> may be 8 characters at maximum.
LABEL=NATSWPRM	The default setting is the name of macro NTSWPRM.

MAXLOCK - Maximum Lock Count

This parameter sets the timeout limit for users of the swap pool. If the limit is exceeded, the swap pool is considered to be destroyed and permanently locked, which is indicated by the error message NUS0180.

MAXSIZE= <i>nnn</i>	<i>nnn</i> must be in the range from 10 to 999.
MAXSIZE=10	This is the default setting.

MAXSIZE - Size of Natural User Threads

This parameter defines the size *nnn* of the Natural user threads in KB. For information on how to determine this size, see [Using the MAXSIZE Parameter](#).

MAXSIZE= <i>nnn</i>	<i>nnn</i> must be in the range from 64 to 32768.
MAXSIZE=400	The default setting is 400 (KB).

Under CICS, this parameter specification is ignored, because the Natural CICS Interface will automatically take the size of the largest thread for this parameter.

NOVPA - Number of Waits for Completed Asynchronous Write

This parameter determines the number of waits for a completed asynchronous write.

NOVPA= <i>nnn</i>	<i>nnn</i> must be in the range from 1 to 999.
NOVPA=20	The default value is 20 (waits).

NOVPW - Number of Waits for Unlocked Swap Pool

This parameter determines the number of waits for an unlocked swap pool.

NOVPW= <i>nnn</i>	<i>nnn</i> must be in the range from 1 to 999.
NOVPW=15	The default value is 15 (waits).

SWPCOPT - Optimizing Size Use of Swap Pool Cache Slot

This parameter determines whether to adjust the size of the swap pool cache slot to optimize storage utilization of the roll file slot.

SWPCOPT=Y	The slot size of the swap pool cache is set to half the size of the roll file slot. SWPCOPT=Y is only accepted if the roll file slot is at least the size of the maximum thread size (MAXSIZE). This is always the case under CICS since the roll file slot is internally set to the maximum thread size.
SWPCOPT=N	The slot size of the swap pool cache is set to the slot size of the roll file. This is the default setting.

SWPFACT - Size of Unit in Reorganization Tables

The factor *n* you specify with this parameter determines the size of a “unit” in the swap-pool reorganization plus tables and minus tables.

SWPFACT= <i>n</i>	Possible values for <i>n</i> are 0 to 4. <i>n</i> determines the size of a “unit” as follows:	
	Value	Corresponds to
	0	2 KB
	1	4 KB
	2	8 KB
	3	16 KB
4	32 KB	
SWPFACT=1	The default setting is 4 KB.	

These tables are used to calculate slot sizes, to dynamically reorganize the swap pool and to get swap-pool statistics see [Dynamic Swap-Pool Reorganization](#).

SWPFILE - Location of Swap Pool Initialization Data

This parameter defines whether the swap-pool initialization data are stored in the Natural system file FNAT or FUSER when the function SWPINIT=AUTO is used.

SWPFILE=FNAT/FUSER	File definition for the swap pool initialization data.
SWPFILE=FNAT	The default value is FNAT.

Note:

The FNAT default setting is for compatibility reasons. However, we recommend that you set this parameter to FUSER if you want to achieve the following:

- Ensure that the FNAT system file does not contain user-specific swap pool changes you defined with the SYSTP utility.
- Share an FUSER system file between different Natural versions. This is impossible with the FNAT system file.

SWPINIT - Access to Swap-Pool Initialization Data

This parameter specifies the access to the swap-pool initialization data through the Natural system file.

SWPINIT=	Blank, as described above under <i>Swap Pool Initialization</i> , see If You Set SWPINIT= .
SWPINIT=AUTO	This is the default setting. The swap-pool initialization data are to be read from/stored in the Natural system file. See also <i>Swap Pool Initialization</i> , If You Set SWPINIT=AUTO .

For more information on how to use this parameter, see [Swap Pool Initialization](#).

SWPLSWP - Number of Logical Swap Pools

This parameter defines the maximum number n of logical swap pools to be used.

SWPLSWP= n	Possible values for n are 0 to 15.
SWPLSWP=0	See Note 3 below.

Notes:

1. The minimum size of a logical swap pool is 64 KB.
2. The value defined must not be smaller than the number of slot sizes defined in the parameter [SWPSLSZ](#).

3. If the default value 0 is used, the swap-pool manager will compute the maximum number of logical swap pools.
4. This parameter will be ignored if the swap-pool initialization data could be read from the Natural system file.

SWPPWRD - Administration Password

With this parameter, you specify the password for the administration of the swap-pool reorganization control data and the Buffer Usage Statistics in the swap-pool manager subsystem of the Natural utility SYSTP.

SWPPWRD= <i>password</i>	The <i>password</i> can be up to 4 characters long.
SWPPWRD=ADMI	This is the default value.

SWPSDIF - Minimum Difference of Slot Sizes

With this parameter, you specify the minimum difference of the slot sizes in the logical swap pools.

SWPSDIF= <i>nn</i>	<i>nn</i> must be an even number and specifies the number of kilobytes (KB). <i>nn</i> must be in the range from 2 to 98.
SWPSDIF=8	The default value is 8 KB.



Note: This parameter will be ignored if the swap-pool initialization data could be read from the Natural system file.

SWPSLSZ - Number of Logical Swap Pools, Slot Sizes

This parameter determines the number of logical swap pools, the slot sizes and the relation of slot numbers between the different logical swap pools.

This parameter will be ignored if the swap-pool initialization data could be read from the Natural system file.

Under *openUTM*:

Make sure that the size specified for a logical swap pool does not exceed the size specified with the ROLLTSZ keyword subparameter of the NATUTM macro (see the *TP Monitor Interfaces* documentation).

Possible values of SWPSLSZ are:

SWPSLSZ=(<i>nn</i> , <i>f</i> (, <i>nn</i> , <i>f</i> ...)) SWPSLSZ=(<i>nn</i> (, <i>nn</i> ...), <i>f</i> (, <i>f</i> ...)) SWPSLSZ=(<i>nn</i> (, <i>nn</i> ...))	<i>nn</i>	Determines the slot size of a logical swap pool in kilobytes (must be an even number). <i>nn</i> must be in the range from 40 to 4096.
	<i>f</i>	Determines the relation in terms of a numerical factor between the slot numbers of the different logical swap pools. <i>f</i> must be in the range from 1 to 9.
SWPSLSZ=(62,1)	The default slot size is 62 KB. The default relation is 1.	

Examples:

```

SWPLSZ=(44,1,62,2)
/* SWAP POOL SIZE IS 2048 KB
/* THERE WILL BE TWO LOGICAL SWAP POOLS, RELATION BETWEEN THEM IS 1:2
/* 1 LOGICAL SWAP POOL WITH 12 (1) 44-KB SLOTS
/* 1 LOGICAL SWAP POOL WITH 24 (2) 62-KB SLOTS<

SWPLSZ=(64,80,96)
/* SWAP POOL SIZE IS 8 MB
/* THERE WILL BE THREE LOGICAL SWAP POOLS, RELATION BETWEEN THEM IS 1:1:1
/* 1 LOGICAL SWAP POOL WITH 34 (1) 64-KB SLOTS
/* 1 LOGICAL SWAP POOL WITH 34 (1) 80-KB SLOTS
/* 1 LOGICAL SWAP POOL WITH 34 (1) 96-KB SLOTS

```

SWPTFIX - Fixed Slot Size

This parameter determines if the size of the swap pool slots is to be fixed or not. Possible values are:

SWPTFIX=Y	The slot size defined with the SWPSLSZ parameter (see above) is taken as a fixed size and no reorganization of the swap pool takes place.
SWPTFIX=N	This is the default value. The slot size defined with the SWPSLSZ parameter (see above) is not taken as a fixed size and the swap pool is reorganized when necessary; that is, the size of the slots is dynamically adjusted to meet the actual requirements.



Note: This parameter will be ignored if the physical swap pool contains more than one logical swap pool.

SWPTIM1 - Time Interval for Reorganization Check

With this parameter, you specify the time interval *nnn* at which a check is to be performed to ascertain whether a swap-pool reorganization is necessary. Possible values are:

SWPTIM1= <i>nnn</i>	<i>nnn</i> must be in the range from 1 to 540 (minutes).
SWPTIM1=(<i>nnn</i> , RESET)	The contents of the swap-pool-reorganization statistics tables are deleted after the check (normally, they are only deleted after a swap-pool reorganization).
SWPTIM1=30	The default value is 30 (minutes).

For details on how the check and a possible swap pool reorganization are performed, see [Dynamic Swap-Pool Reorganization](#).

 **Important:** If the parameter SWPTFIX is set to Y or if the physical swap pool contains more than one logical swap pool, the SWPTIM1 parameter does not apply.

SWPTIM2 - Lapse of Time Before Start of Reorganization

With this parameter, you can specify the time *nn* to elapse after the check for the necessity of a swap-pool reorganization is performed and before the actual reorganization is to be started.

SWPTIM2= <i>nn</i>	<i>nn</i> must be in the range from 1 to 99 (minutes)
SWPTIM2=2	The default value is 2 (minutes).

During this time, no further user areas can be placed in the swap pool, while user areas still held in the swap pool can still be used and read in the Natural user thread.

For details on how the check and a possible swap-pool reorganization are performed, see [Dynamic Swap-Pool Reorganization](#).

If the parameter SWPTFIX is set to Y or if the physical swap pool contains more than one logical swap pool, the SWPTIM2 parameter does not apply.

SWPUSER - Condition for Swap Pool Reorganization

With this parameter you can define which condition has to be met for a swap-pool reorganization to take place.

SWPUSER= <i>nn</i>	<i>nn</i> must be in the range from 1 to 99.
SWPUSER=20	The default value is 20 (percent).

You can define a percentage value *nn* which determines the percentage of dialog steps of all users where the length of the compressed user areas was 1 or more units larger (or 1 or more units smaller) than the current slot size. If a check establishes that this percentage is reached, a swap-pool reorganization takes place.

For details on how the check is performed, see [Dynamic Swap-Pool Reorganization](#).

If the parameter SWPTFIX is set to Y or if the physical swap pool contains more than one logical swap pool, the SWPUSER parameter does not apply.

WAITMS - Wait Time for Unlocked Swap Pool

This parameter determines the number of milliseconds for one wait of an unlocked swap pool.

WAITMS= <i>nnn</i>	<i>nnn</i> must be in the range from 1 to 999.
WAITMS=5	The default value is 5 (milliseconds).

WRITMS - Wait Time for Completed Asynchronous Write

This parameter determines the number of milliseconds for one wait of a completed asynchronous write.

WRITMS= <i>nnn</i>	<i>nnn</i> must be in the range from 1 to 999.
WRITMS=10	The default value is 10 (milliseconds).

41 Natural User Area Size Considerations

- Using the MAXSIZE Parameter 302
- Defining the Size of the Individual Natural Buffers 302
- Possible Error Messages 302
- Displaying the Aggregate Size of All Buffers 303
- Calculating the Maximum Size 303

This document describes how to manage the size of the Natural user area and the size of the individual Natural buffers.

Using the MAXSIZE Parameter

The overall size of the Natural user area is determined by the `MAXSIZE` parameter in the swap-pool parameter module. Therefore the `MAXSIZE` must be set large enough to contain the aggregate size of all buffers that are required by Natural and also by possibly used subsystems (Connect, TRS, etc.). The buffer requirements of Natural and subsystems are met by the TP driver. When a Natural application is started, a user thread with a size of `MAXSIZE` is created. This is done by a physical request memory to the operating system.

The buffer requests of Natural to the TP driver cause only “logical” `GETMAINS`; that is, the Natural user thread is then divided into “logical” units: the Natural buffers.

Defining the Size of the Individual Natural Buffers

The size of the individual Natural buffers is either explicitly defined in the **Natural parameter module** (with the parameters `ESIZE` (size of user-buffer extension area), `CSIZE` (size of Connect buffer area), etc.) or is implicitly determined by the definitions of the parameters `PS` (page size for Natural reports), `LS` (line size), etc.

The maximum sizes of the Natural buffers can be displayed with the function *Buffer Usage Statistics* of the Natural utility `SYSTP`. `SYSTP` also offers functions for ascertaining the overall maximum Natural buffer sizes used for all users of a specific application.

Possible Error Messages

When the Natural error message `NOT ENOUGH MEMORY or BUFFER SIZES EXCEED MAXSIZE` appears, this indicates that the `MAXSIZE` parameter value has not been defined large enough.

Displaying the Aggregate Size of All Buffers

The aggregate size of all buffers requested by Natural (that is, the amount of `MAXSIZE` actually used by the users of an application) can be obtained via the *Natural Swap Information* function of the `SYSTP` utility.

Calculating the Maximum Size

A standard way of calculating the `MAXSIZE` is:

Add all explicitly defined buffer sizes (for example, `ESIZE`) and 40 KB (the sum of the internal Natural buffer sizes).

This gives you roughly the required size for `MAXSIZE`.

42 Swap Pool Data Space

- Using ESA Data Space in Addition 306
- ESA Data Space Slot Size Adjustment 306

This document describes how to extend the Natural Swap Pool capacity by generating ESA Data Space.

Using ESA Data Space in Addition

To achieve a further reduction of the swap I/O operations, you can use the keyword parameters `DATA` and `DESA` of the `CMPSTART` program to extend the Natural Swap Pool capacity by generating ESA Data Space. This Data Space will be available to store compressed Natural user threads whenever the Swap Pool runs out of space.

When this Data Space has been also consumed, a check occurs whether it is necessary to write user threads from the Data Space to the roll file, because their life time has ended (see the keyword parameters `DSPCONT` and `DSPLIFE` of macro `NTSWPRM`).

If there is no free storage space in the Data Space, the swap pool logic will cause the oldest inactive user thread to be written from the swap pool to the roll file.

ESA Data Space Slot Size Adjustment

The generated ESA Data Space is divided into slots of equal size.

- If you are using the TP monitor `openUTM`, you can define the slot size by appropriately setting the keyword parameter `ROLLTSZ` of the `NATUTM` macro.
- If you are using the TP monitor `CICS`, the Data Space slot size will automatically take the size of the longest thread.
- You can optimize use of the roll file slot by setting the keyword parameter `SWPCOPT=Y` (`NTSWPRM` macro) to adjust the Data Space slot size.

The size, name and cache size of the swap pool are specified using profile parameter `BPI` or the corresponding macro `NTBPI` in the **Natural parameter module**, that is, the `(NT)BPI` settings in effect for the Natural session initializing the Natural CICS environment are taken.

43 Global Restartable Swap Pool under openUTM

- Purpose of a Natural Global Swap Pool under openUTM 308
- Installing a Natural Global Swap Pool under openUTM 308
- Starting a Natural Global Swap Pool under openUTM 309
- Displaying Information about the Global Swap Pool 309

This document describes how to install and operate a Natural global swap pool in a Natural under *openUTM* environment.

Purpose of a Natural Global Swap Pool under openUTM

If all tasks of a Natural under *openUTM* application are terminated abnormally, the contents of a local Natural swap pool are deleted. Consequently, when a task is started again, a new swap pool is initialized and all users affected by the abnormal termination must start their Natural sessions again.

To avoid this situation, a global (that is, restartable) swap pool can be used: after an abnormal termination of the Natural under *openUTM* application, when the users log on to the application again, the last screen displayed before the termination is sent again and the users can resume their session at the point where they were interrupted.

Installing a Natural Global Swap Pool under openUTM

The following prerequisites are required for the installation of a global swap pool:

If a global swap pool is to be used, a global buffer pool must also be used. Before the restart of a Natural under *openUTM* application, the global buffer pool must have been initialized; that is, at least one user must have used this buffer pool by normally starting a new Natural session.

If a new global buffer pool is started before an abnormally terminated Natural under *openUTM* application is restarted, a new global swap pool must also be started. However, if a new global swap pool is started, a new global buffer pool need not be started as well.

The relation between the swap pool and the swap file is as follows: When the first *openUTM* task uses a newly started swap pool, the swap file is opened with `OPEN 'OUTIN'`, which means that the contents of the swap file are deleted. When a subsequent *openUTM* task uses an already used (initialized) swap pool, the swap file is opened with `OPEN 'INOUT'`, which means that the contents of the swap file can still be used.

Starting a Natural Global Swap Pool under openUTM

A Natural global swap pool must be started with program `CMPSTART`. It can be used from a maximum of five Natural under *openUTM* applications.

Displaying Information about the Global Swap Pool

To obtain information on the current parameters settings of the global swap pool, as well as the date and time of its start,

Issue the console command:

```
/INTRtsn,DPR
```


44 Terminating the Global Swap Pool under openUTM

- Termination Using Console Commands 312
- Abnormal Termination with Dump 312
- Termination by Program 312

This document describes the ways in which a Natural global swap pool can be terminated under *openUTM*.



Caution: Before the swap pool is terminated, the Natural under *openUTM* application that uses it must be terminated.

Termination Using Console Commands

➤ To terminate the global swap pool normally

- 1 Issue the console command:

```
/INTR tsn,STOP
```

- 2 or issue the console command:

```
/INTR tsn,END
```

Abnormal Termination with Dump

➤ To terminate the global swap pool abnormally, producing a dump

- Issue the console command:

```
/INTR tsn,DUMP
```

The swap pool is terminated abnormally and a dump is produced.

Termination by Program

➤ To terminate the global swap pool normally, using the program CMPEND

- Issue the following command:

```
/SYSFILE SYSDTA=(SYSCMD)  
/EXEC (CMPEND,NAT230,MOD) name
```


XI System Spool Access

45 System Spool Access

- Purpose 318
- Prerequisite 318
- Using the Write-to-Spool Feature 318

This document describes the Write-to-Spool feature for Natural.

This feature is available under z/OS and z/VSE only.

See also:

- *Installing and Activating the Write-to-Spool Feature* in the section *Installing Entire System Server Interface on z/OS* in the *Installation for z/OS* documentation.
- *Installing and Activating the Write-to-Spool Feature* in the section *Installing Entire System Server Interface on z/VSE* in the *Installation for z/VSE* documentation.

Purpose

The Write-to-Spool feature enables Natural users to write reports to the system spool directly. It can be used in any Natural environment (Com-plete, TSO, CICS, IMS TM, batch, etc.) and uses the Entire System Server view `WRITE-SPPOOL`.

Under z/OS, the `SYSOUT` is part of the Entire System Server job stream within the JES spool, and it may be processed by any software which expects output in JES Spool, for example, Entire Output Management. The JES spool may be a JES2 or a JES3 spool.

Under z/VSE, the `SYSOUT` is a separate entry in POWER queue.

Prerequisite

To use the Write-to-Spool feature, the Entire System Server needs to be installed.

Using the Write-to-Spool Feature

The Write-to-Spool feature is handled by a so called “access method”, which is called `ESS` for Entire System Server. You may define your printer in the **Natural parameter module** or dynamically in your session parameters.

Defining Your Printer

» To define your printer

- 1 Define the printer in the **Natural parameter module**.

Use the NTPRINT macro to specify the printer number (n) and the access method (AM):

```
NTPRINT (n),AM=ESS
```

Example:

```
NTPRINT (1,3),AM=ESS
```

In this example, the printers 1 and 3 are defined for use with access method ESS (Entire System Server).

Or:

Define the printer during session startup by specifying the profile parameter PRINT, for example:

```
PRINT=((1-6),AM=ESS)
```

In this example, the printers 1 to 6 are defined for use with access method ESS (Entire System Server).

- 2 Link the access-method modules to the Natural nucleus.

See the platform-specific *Installation* documentation.

Or:

Load it dynamically by specifying the following profile parameters RCA and RCALIAS:

```
RCA=(NATAM11),RCALIAS=(NATAM11,NATPWSAM)
```

where NATPWSAM is the delivered write-to-spool module containing the default parameters.

If you have linked a module with adapted parameters, use the name of this module instead.

- 3 Define the JES destination with the OUTPUT option of the DEFINE PRINTER statement. You can use one of the following examples depending on whether you want to send the output to a spool file, a local JES printer, or through a remote JES node to a remote user or device.

Example:

```
DEFINE PRINTER (n) OUTPUT 'LOCAL' /* For printing on local JES/POWER printers
```

Or:

```
DEFINE PRINTER (n) OUTPUT 'DAEF' /* For printing to JES spool called DAEF
```

Or:

```
DEFINE PRINTER (n) OUTPUT 'DEST=node-name,REMOTE-USERID=user-id' /* For printing ↵  
to remote JES nodes
```

where:

- *n* is the number in the NTPRINT entry in the **Natural parameter module** described in **Step 1**.
- *node-name* is the name of the remote JES node.
- *user-id* the ID of the user or device who receives the output.

Reports can now be written to the system spool using one of the following statements:

```
DISPLAY (n)
```

or

```
PRINT (n)
```

where *n* is the number in the NTPRINT entry in the **Natural parameter module** in **Step 1**.

Users can set the output format and number of copies using the FORMS and COPIES clauses of the DEFINE PRINTER statement.

Example:

```
DEFINE PRINTER (2) OUTPUT 'DEST'  
FORMS 'FORM'
```

The defaults for items such as Entire System Server node, forms and output class can be found in the module NATWSPDF.

Examples for z/OS

Example 1

Assume using the factory settings and executing the Natural program:

```
DEFINE PRINTER (2) OUTPUT 'WK1'
WRITE (2) 'THIS IS A SMART RECORD'
CLOSE PRINTER (2)
```

During the execution of this program, you can see the following fields with their values in the **Display Active Tasks** panel:

```
DDNAME      DSID Owner C Dest Rec-Cnt Forms Wtr PageDef FormDef
SYS00001    104 WKK  A WK1      2     STD
```

Browsing this data set, you can see:

```
Page 1
THIS IS A SMART RECORD
```

Example 2

Assume using the default member:

Parameter	Explanation (Possible Values)
WSPDFLT NODE=55526,	Entire System Server target node number (5 characters at maximum)
PROGRAM=HUGO,	JES writer (8 characters at maximum)
CLASS=Y,	SYSOUT class (1 character)
HOLD=YES,	Hold (YES or NO)
CNTL=A,	Carriage control (A or M)
FORM=WOF0,	Form (4 characters at maximum)
RMT=JESWOLF,	JES remote (8 characters at maximum)
FORMDEF=FOWOLF,	Form definition (6 characters at maximum)
PAGEDEF=PAWOLF	Page definition (6 characters at maximum)

Execute the following Natural program:

```
DEFINE PRINTER (2) OUTPUT 'WK1'
WRITE (2) 'THIS IS A SMART RECORD'
CLOSE PRINTER (2)
```

During the execution of this program, you can see the following fields with their values in the **Display Active Tasks** panel:

```
DDNAME  DSID Owner C Dest Rec-Cnt Forms  Wtr  PageDef  FormDef
SYS00002 105  WKK  Y  WK1    2    WOFO  HUGO  PAWOLF  FOWOL
```

Browsing this data set, you can see:

```
Page 1
THIS IS A SMART RECORD
```

Example 3

Assume using the default member:

Parameter	Explanation (Possible Values)
WSPDFLT NODE=55526,	Entire System Server (NPR) target node (node number)
PROGRAM=*OUTPUT,	JES writer (8 characters at maximum)

The other parameters in the default member are not changed.

Run the following example program:

```
DEFINE PRINTER (2) OUTPUT 'KURT'
PRINT (2) ' here comes KURT'
CLOSE PRINTER (2)
```

After that, Entire System Server fetches the value from the field OUTPUT in the DEFINE PRINTER statement and inherits it as the JES writer attribute for the specific spool data set.

Looking in TSO/SDSF under the job name of Entire System Server, you can see the following:

```
PREFIX=NPR*  DEST=(ALL)  OWNER=*  SYSNAME=
NP  DDNAME  Time      Forms    FCB  UCS  Wtr    Flash
   SYS00005 10:20:48          ****  ****  KURT   ****
```

If in JES an associated JES writer program is defined, it gets control and handles this output as defined in the program.

Examples for z/VSE

In z/VSE, the output is written into the POWER spool under a new job number. The name of the printed spool data set will be the name of the original batch job or the user ID of the TP monitor.

Example 1

Assume we are using the factory settings for NATWSPDF and have the natural program:

```

DEFINE PRINTER (2) OUTPUT 'ELSA'
WRITE (2) 'THIS IS A SMART RECORD'
CLOSE PRINTER (2)

```

User WKK is running this program in a batch-mode Natural, with JOBNAME=GERHARD.

The power queue contains the following information:

JOBNAME	JOBNO	Q	NUM	C	D	PR	STAT	FROM	TO	CP	PAGES	RECORDS	ID
GERHARD	0020443	L		A	H	003	HOLD	WKK	ELSA	1	1	2	

Browsing this output displays the following information:

```

BROWSE-DJ:GERHARD(20443)-Queue:LS ----- Row 0 - Columns 001 076
  COMMAND====>                                SCROLL====> CSR
***** top of list *****
1Page      1                                     date, time
0THIS IS A SMART RECORD
***** bottom of list *****

```

Example 2

Assume we are using the factory settings for NATWSPDF and have the natural program:

```

DEFINE PRINTER (2) OUTPUT 'KARL'
                        FORMS 'F001'
                        COPIES 4
WRITE (2) 'PRINTER TEST'
WRITE (2) 'OK?'
CLOSE PRINTER (2)

```

The power queue contains the following information:

JOBNAME	JOBNO	Q	NUM	C	D	PR	STAT	FROM	TO	FORM	CP	PAGES	RECORDS	ID	PR
GERHARD	0020465	L		A	H	003	HOLD	WKK	KARL	F001	4	1	3		

XII

Natural 3GL CALLNAT Interface

This part contains information about the Natural 3GL CALLNAT Interface which enables 3GL programs to invoke and execute Natural subprograms.

[Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions](#)

[Natural 3GL CALLNAT Interface - Usage, Examples](#)

46 Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions

▪ Purpose of 3GL CALLNAT Interface	328
▪ Prerequisites	328
▪ Restrictions	330

This document describes the purpose of the 3GL CALLNAT interface and its prerequisites and restrictions.

Purpose of 3GL CALLNAT Interface

With the 3GL CALLNAT interface, Natural enables 3GL programs to invoke and execute Natural subprograms.

The 3GL can be any programming language which supports the standard linkage call interface. In most cases this will be a COBOL program, but the functionality can also be used by, for example, PL/1, FORTRAN, C or Assembler programs.

Availability

The interface is available in batch mode under the following operating systems:

- z/OS,
- z/VSE,
- BS2000,

and for the following TP-monitor environments:

- CICS,
- Com-plete,
- IMS TM,
- TIAM,
- TSO,
- *open*UTM.

Prerequisites

This section describes the prerequisites to execute a Natural subprogram from a 3GL program, using an internal CALLNAT statement. To achieve the desired functionality, a Natural environment must be set up before you execute the CALLNAT interface from your 3GL program.

- [Space Requirements](#)
- [Linking](#)

- [Environment Dependencies](#)

Space Requirements

The mechanism of parameter addressing in a Natural program requires that the parameters passed reside in an area allocated by Natural, that is, in any of its sizes. The 3GL program, however, allocates the storage for its variables somewhere in the address space of the task. To make addressing still successful, a “call-by-value” mechanism is used for those variables which do not already reside in a Natural area. This means that, prior to invoking the Natural subprogram, the parameters to be passed are transferred into a Natural area, namely the DATSIZE buffer.

In addition to the storage used for the contents of the variables, additional storage will be needed depending on the number of parameters. The total amount of space required is approximately the same as the space that would be needed in the DATSIZE buffer if the subprogram-invoking program were coded in Natural.

Linking

To invoke the Natural subprogram, the 3GL program must call the CALLNAT interface. Depending on the power and functionality of the call interface of the 3GL programming language, the CALLNAT interface can be either placed in an accessible load library for dynamic loading or linked to the 3GL program.

It is recommended, whenever possible, to load the CALLNAT interface dynamically from a Natural steplib, as this method makes sure that always the most recent version of that program is used.

The samples XNATGCP2 and XNATGCP2 are provided to elucidate the technique of dynamically loading and calling the CALLNAT interface from COBOL or PL/I, respectively.



Note: Check with the responsible system programmer for the best solution in your environment.

Environment Dependencies

The foreign 3GL module can be either linked to Natural as a CSTATIC module and then invoked via a branch and link instruction, or loaded dynamically and invoked via a TP-dependent link method.

In the latter case, the 3GL module is written in a TP-specific way and the CALLNAT interface must be adapted accordingly. For this purpose, multiple TP-specific interface modules are provided:

Interface Module	Purpose
NATXCAL	<p>To be used in the following cases:</p> <ul style="list-style-type: none"> ■ if the 3GL module is either loaded dynamically or linked to Natural and then invoked by a branch and link instruction (batch, Com-plete, IMS TM, TIAM, TSO, <i>openUTM</i>, %P=S and %P=LS in CICS). ■ if the 3GL module is called via the INTERFACE4 option of the CALL statement. It provides the INTERFACE4 Natural Callnat Interface as well as the INTERFACE4 Callback Functions. For further information on the INTERFACE4 functionality, see the CALL statement documentation. <p>Note: For CALL INTERFACE4 purposes, NATXCAL cannot be loaded dynamically but must be linked to the 3GL program.</p>
NCIXCALL	<p>To be used in a CICS environment if the 3GL module has been invoked using EXEC CICS LINK; NCIXCALL is delivered in source code to be compiled with your CICS macros. See also <i>Installing the Natural CICS Interface on z/OS</i> or <i>Installing the Natural CICS Interface on z/VSE</i> in the <i>Installation</i> documentation.</p>
NCIXCPRM	<p>To be used in a CICS environment to build the parameter address list used as COMMAREA for the subsequent EXEC CICS LINK command.</p>

Restrictions

Terminating a Natural Subprogram

The invoked Natural subprogram should be terminated with a return to the calling program.

Inadmissible Natural Statements

The following statements must not be used.

- FETCH
- RUN
- STOP
- TERMINATE

When used in the invoked Natural subprogram they will bring about an appropriate Natural runtime error (NAT0967).

Parameter Values Passed by the 3GL Program

The parameter values passed by the 3GL program must not reside in a write-protected storage area.

Dynamic Arrays

Arrays with dynamic ranges are not possible.

BS2000 Operating System Specific Restriction

Since BS2000 CRTE does not allow recursions, Natural subprograms, called by the 3GL module, cannot execute any subsequent 3GL calls. A subsequent 3GL call would cause either an enforced abnormal termination of the current Natural session or Natural error NAT0920.

This applies to BS2000 batch, TIAM, and *openUTM*.

TP-Monitor-Specific Restrictions

■ Under CICS

For CICS environments, the 3GL program that uses the Natural 3GL CALLNAT interface must be written for conversational mode. The 3GL program runs on the second CICS program level and pseudo-conversational program technique can therefore not be used.

■ Under IMS TM and *openUTM*

IMS TM and *openUTM* environments running Natural can use the 3GL CALLNAT interface only if both the 3GL program and the Natural subprogram do not issue any terminal I/O; when `DISPLAY`, `INPUT` and `WRITE` are used in the invoked Natural subprogram they will bring about an appropriate Natural runtime error (NAT0967).

47

Natural 3GL CALLNAT Interface - Usage, Examples

▪ Usage	334
▪ Sample Environments	338

This section describes the usage of the 3GL CALLNAT interface and describes a number of sample 3GL CALLNAT environments.

Usage

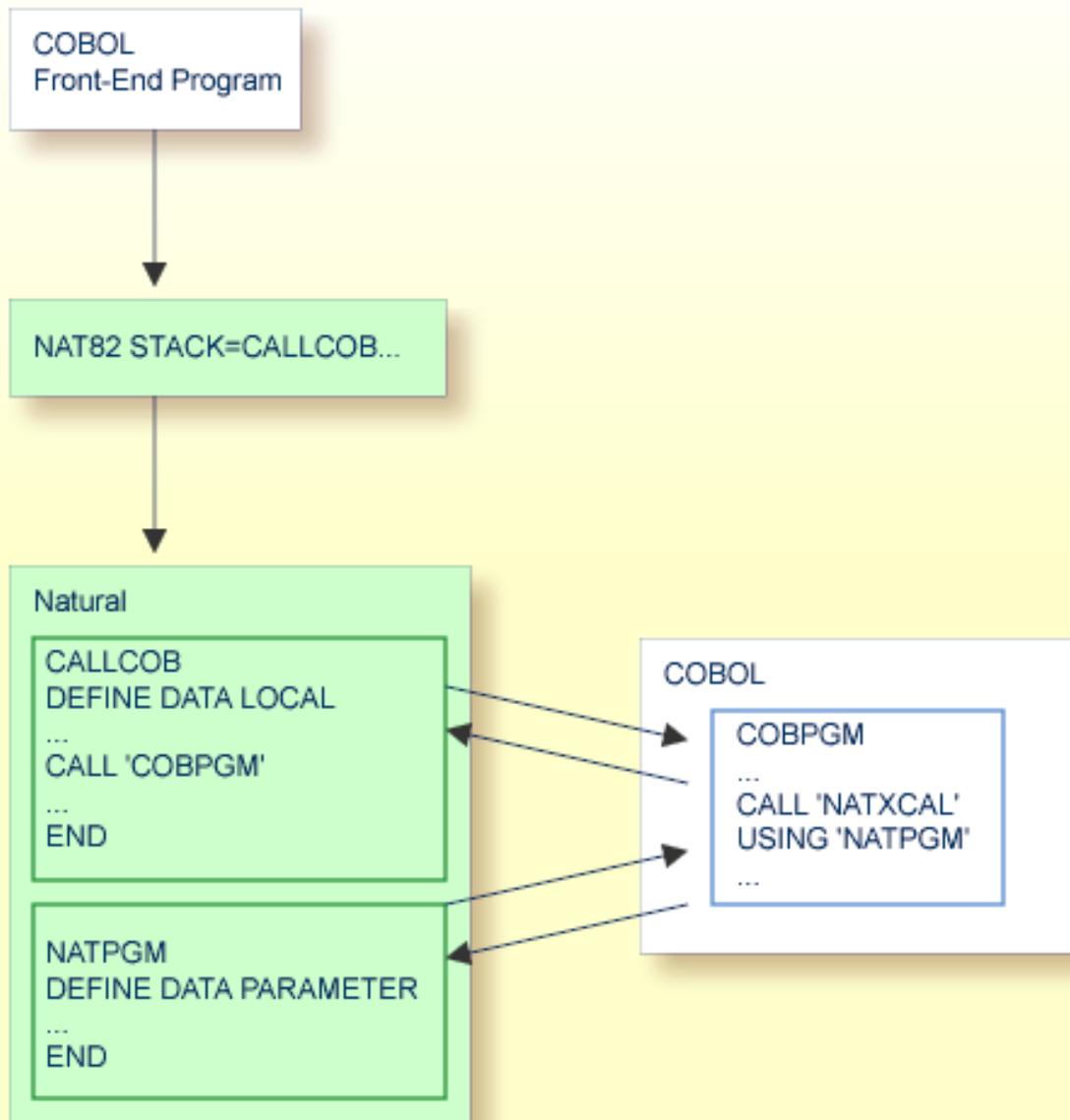
The following topics are covered:

- [Overview](#)
- [Call Structure](#)
- [Parameter Handling](#)

Overview

When you invoke a Natural subprogram from a 3GL program, a Natural session must be active, i.e. the 3GL program itself must be called by Natural.

Therefore you must take special precautions if you do not want the Natural layer to show up. The following figure is intended to give you an overview of how an application using the Natural 3GL CALLNAT interface may be designed in such a case:



The necessary environment is established by first invoking a Natural start-up program. By using the Natural `CALL` statement, this start-up program can then invoke a 3GL program from where you can invoke the `CALLNAT` interface.

Call Structure

The Natural main program is very simple; it only calls, for example, a COBOL program:

```
.....  
CALL 'COBPGM'  
END  
.....
```

The CALL statement of the 3GL programming language (for example, COBOL) must have access to the Natural 3GL CALLNAT interface, which then invokes the Natural subprogram:

```
.....  
CALL 'interface' USING natpgm p1 ... pn  
.....
```

The parameter *interface* is environment-dependent (for example, NATXCAL) and linked to the calling program. The parameter *natpgm* must be an alphanumeric variable of 8 bytes that contains the name of the Natural subprogram to be invoked. The parameters *p1 ... pn* are passed to the Natural subprogram.

Example (for all environments except CICS):

The COBOL program COBPGM could contain coding similar to the following one:

```
.....  
MOVE 'FINDNPGM' TO natpgm  
CALL 'interface' USING natpgm number name  
IF natpgm NE 'FINDNPGM'  
THEN GOTO error_handling_1  
.....
```

The invoked Natural subprogram FINDNPGM calculates the number of persons in the file EMPLOYEES with *name* equal to a value passed from the COBOL program:

```
DEFINE DATA  
PARAMETER  
1 pnumber (P10)  
1 pname (A20)  
LOCAL  
1 emp VIEW OF employees  
END-DEFINE  
*  
RESET presp  
FIND NUMBER emp WITH name=pname  
MOVE *NUMBER TO pnumber  
ESCAPE ROUTINE
```

If an error occurs while the subprogram is executed, information about this error will be returned in the variable *natpgm* in the form *NAT*nnnn*, where *nnnn* is the corresponding Natural error number.

Example (for CICS only):

Under CICS, the call of a Natural subroutine from, for example, COBOL should be as follows:

```

...
WORKING STORAGE SECTION
...
01 PARM-LIST PIC X(132).
01 NATPGM PIC X(8).
01 NUMBER PIC 9(10) comp-3.
01 NAME PIC X(20).
...
PROCEDURE DIVISION
...
MOVE 'FINDNPGM' TO NATPGM
CALL 'NCIXCPRM' USING PARM-LIST NATPGM NUMBER NAME ...
EXEC CICS LINK PROGRAM('NCIXCALL')
COMMAREA(PARM-LIST) LENGTH(132) END-EXEC.
...

```

The called subroutine `NCIXCPRM` builds the parameter address list used as `COMMAREA` in the subsequent `EXEC CICS LINK` command.

Parameter Handling

There is no format and length checking. It is the caller's responsibility to pass a correct parameter list. The number, format and length of the parameters are defined by the invoked Natural subprogram.

When you are passing parameters, group arrays should not be passed, since they are resolved as individual arrays:

Example of Invalid Syntax:

```

.....
01 GROUP (1:2)
02 F1
02 F2
.....
.....
CALL ..... F1 F2
.....

```

Example of Valid Syntax:

```
.....  
01 F1 (1:2)  
01 F2 (1:2)  
.....  
.....  
CALL ..... F1 F2  
.....
```

Arrays with dynamic ranges cannot be used as parameters.

Sample Environments

The objective for the sample 3GL CALLNAT environments below is to demonstrate how a COBOL routine can call a Natural subprogram under specific TP-monitor systems or in batch mode, and to give system-specific instructions to create such environments.

The following topics are covered:

- [Sample Environment for CICS](#)
- [More Samples](#)
- [Sample for Any Other Supported Environment](#)

Sample Environment for CICS

Perform the following steps to create a sample Natural 3GL CALLNAT environment under CICS:

Step 1: Create the Environment Initialization

- Set up the front-end program that initializes the 3GL CALLNAT environment.
- Use the COBOL front-end `XNCIFRCX` in the Natural/CICS source library. It starts Natural, stacks `LOGON YOURLIB` and executes the program `TSTCOB`, which initializes the Natural 3GL CALLNAT environment.
- Locate the string `NCvr` (where `vr` represents the relevant product version) in the source code and replace it with the valid transaction ID for Natural.
- Compile and link-edit the COBOL program and define program to CICS via `CEDA DEFINE PROGRAM`.

Step 2: Install the Sample COBOL Call

Provided in the Natural/CICS source library `NCI.SRCE` is the sample member `XNCI3GC1`, which contains a default call to the Natural subprogram `MYPROG`.

- For test purposes, create the following program in the library `SYSTEM` and stow it as:

```
WRITE 'BEFORE PGM EXECUTION'
CALL 'COBNAT'
WRITE 'AFTER PGM EXECUTION'
END
```

- Look at the XNCI3GC1 source and review the CALL and LINK. Compile and link as COBNAT with the following CICS INCLUDE directives or use Step 2 of the Sample Job NCTI070:

```
INCLUDE CICS LIB(DFHECI)
INCLUDE XNCI3GC1           <= output from translator and compiler
INCLUDE NCILIB(NCIXCPM)
ENTRY XNCI3GC1
NAME COBNAT(R)
```

Step 3: Create a Sample Natural Subprogram

By default, the source member XNCI3GC1 is set up to call the Natural subprogram MYPROG in the library YOURLIB. The program TSTCOB, as mentioned above, starts up the process by calling COBNAT that contains the actual call to the Natural subprogram MYPROG.

- Create the subprogram MYPROG to demonstrate the executing Natural subprogram.

```
DEFINE DATA PARAMETER
  01 PARM1 (A18)
  01 PARM2 (A18)
  01 PARM3 (A18)
END-DEFINE
*
  MOVE 'PARAM01' TO PARM1
  MOVE 'PARAM02' TO PARM2
  MOVE 'PARAM03' TO PARM3
END
```

Step 4: Verify the CICS Resources

- Use the job NCII005 for a guide to defining the CICS resources (PPT and PCT).
- Define the required CICS resources (PPT and PCT).

Step 5: Test the Environment

Test the environment by using the NCYC default transaction. Use CEDF to monitor the program control and observe the data areas in use.



Important: Since Natural is at the top of the CICS program hierarchy, any COBOL subprogram issuing terminal I/Os must run in conversational mode. Pseudo-conversational programs would need to be modified, and any new development using the Natural 3GL CALLNAT interface should be done in conversational mode.

More Samples

Sample Program	Description
XNCI3GC2	COBOL sample with same functionality as XNCI3GC1, but accepting parameters from the calling Natural program.
XNCI3GP1	PL/I sample with same functionality as COBOL sample XNCI3GC1.
XNCI3GP2	PL/I sample with same functionality as XNCI3GC1, but accepting parameters from the calling Natural program.

More Non-CICS Samples

Sample Program	Description
XNAT3GC2	COBOL sample with same functionality as CICS sample XNCI3GC2.
XNAT3GP2	PL/I sample with same functionality as CICS sample XNCI3GP2.

Sample for Any Other Supported Environment

Perform the following steps to create a sample Natural 3GL CALLNAT:

Step 1: Assemble and Link ASMNAT

The sample Assembler routine XNAT3GA1 contains a basic example to access the CALLNAT interface. The register calling conventions are in the source of this program.

Link NATXCAL with XNAT3GA1 with entry point ASMNAT.

Step 2: Start the Natural Session

Start a Natural session stacking a program that calls the ASMNAT program which in turn calls the Natural subroutine ASMNAT.

XIII

Operating the Software AG Editor

This part contains information on how to operate the Software AG Editor.

The Software AG Editor is a feature that represents basic functionality within Natural, exclusively used by several Natural subproducts and other Software AG products.

[Editor Work File](#)

[Editor Buffer Pool](#)

See also:

- *SYSEDT Utility - Editor Buffer Pool Administration* in the *Utilities* documentation
- *Installing the Software AG Editor* in the *Installation for z/OS* documentation
- *Installing the Software AG Editor* in the *Installation for z/VSE* documentation
- *Installing the Software AG Editor* in the *Installation for BS2000* documentation
- *Software AG Editor* in the *Editors* documentation

48 Editor Work File

- Editor Work File Structure 344
- Editor Work File under z/OS, z/VSE and BS2000 345
- Using the Software AG Editor Work File Formatting Utility 346
- Formatting during Initialization 346
- Maintaining the Editor Work File under z/OS and z/VSE 346
- Maintaining the Editor Work File under BS2000 347
- Editor Work File under Complete/SMARTS 348

This document describes structure, use and maintenance of the editor work file under the various operating systems.

See also:

- *SYSED*T Utility - Editor Buffer Pool Administration in the *Utilities* documentation
- *Installing the Software AG Editor on z/OS* in the *Installation for z/OS* documentation
- *Installing the Software AG Editor on z/VSE* in the *Installation for z/VSE* documentation
- *EDBP - Software AG Editor Buffer Pool Definitions* in the *Parameter Reference* documentation
- *Software AG Editor* in the *Editors* documentation

Editor Work File Structure

The editor work file is a relative record data set with fixed length records. It is divided into three parts:

- Control Record
- Work Records
- Recovery Records



Note: If you use an editor auxiliary buffer pool defined by the profile parameter `EDPSIZE`, no editor work file is required.

Control Record

The control record contains buffer pool control information including the buffer pool parameters.

During the first initialization of the work file or during a buffer pool cold start (triggered by editor buffer pool subparameter `COLD`), the values defined in the editor buffer pool parameter `EDBP` and/or in the corresponding macro `NTEDBP` are saved in the work file control record. Moreover, the current operating system Id (system variable `*HOSTNAME`) and the global buffer pool name or the current job name are saved for subsequent verification.

You can modify the control record by using the Generation Parameters function of the *SYSED*T Utility.

For buffer pool warm restarts, the buffer pool parameters are read from the control record.

Work Records

The work records contain logical file records which have been moved out of the buffer pool due to a lack of free buffer pool blocks.

Logical work file records are lost during a restart of the buffer pool or if a timeout occurs for the logical file.

Recovery Records

The recovery records hold checkpoint information of editor sessions. If the system terminates abnormally, this information can be used by the editor recovery facility to recover logical files. Recovery records are lost during a cold restart of the buffer pool.

The recovery facility is used by Natural ISPF only. If you do not intend to use this product, you can run without the recovery part by defining the editor buffer pool subparameter `PWORK=100`.

Editor Work File under z/OS, z/VSE and BS2000

One editor work file corresponds to one **Editor Buffer Pool**. If you intend to use a global editor buffer pool, the editor work file must be shared by all users using the same global editor buffer pool. The accessed editor work file can be used only by sessions within the same operating system (system variable `*HOSTNAME`) and with the same global buffer pool or the same job name for local buffer pools. This connection can be dropped by a buffer pool cold start only. Alternatively, the Software AG editor work file formatting utility can be used to reset the work file connection.

The editor work file must be large enough to contain the editor sessions of all users. A minimum number of 100 records per editor user is recommended. The record length of the work file must be fixed, can be defined from 504 to 16384 bytes, and must be a multiple of 8.



Note: The record length of data sets or PDS members, which will be edited with Natural ISPF, cannot be larger as the record length of this editor work file.

The size of a work file record is specified either when allocating the editor work file (under z/OS and z/VSE; default size is 4088) or by definition in the buffer pool parameter macro (under BS2000; default size is 4096).

The total number of editor work file records depends on the allocated data set space for the editor work file.

There are two alternative ways of formatting the editor work file:

- offline by using the Software AG editor work file formatting utility,
- online during buffer pool initialization.

Using the Software AG Editor Work File Formatting Utility

This method is to be preferred, because no online user has to wait until formatting is finished. Optionally, the **Natural parameter module** may be assembled and linked to the Software AG editor work file formatting utility to specify editor buffer pool parameters by means of the macro NTEDBP. Otherwise, the default parameter values apply.

During reformatting, however, the work file must not be in use, which means that the system(s) using the corresponding buffer pool have been terminated before reformatting.

Formatting during Initialization

When the editor buffer pool is in uninitialized or terminated state, then during the first session which uses the Software AG editor, a "buffer pool cold start" is performed on one of the following conditions:

1. if the work file has not been formatted yet,
2. if the control record indicates "cold start" (which can also be specified by using the Editor Buffer Pool Administration utility SYSEDT),
3. if the buffer pool subparameter COLD=ON was specified.

Otherwise, a buffer pool warm start is performed if a valid control record is found during buffer pool initialization. In this case, all buffer pool parameters are taken from the work file control record and no records are formatted.

Maintaining the Editor Work File under z/OS and z/VSE

If you want to change the size of the editor work file (for example, because it is too small), the COPY function of the Software AG editor work file formatting utility can be used to avoid a buffer pool cold start; that is, the loss of the recovery records.

To copy an existing editor work file, perform the following steps:

1. Modify any buffer pool parameters by using the *SYSEDT Utility*, for example, PWORK if you want to change the percentage of work records in the file.
2. Terminate the editor buffer pool by using the *System Administration Facilities* of the *SYSEDT Utility* and ensure that no Natural session is using the editor after the buffer pool termination.
3. Close (if necessary) and deallocate the editor work file.
4. Rename the editor work file by using the VSAM utility IDCAMS (ALTER command).

5. Define a new editor work file with the original name and possibly a different size, but with the same record length.
6. Perform the following steps:
 - In the EXEC JCL card, add PARM=COPY.
 - For the renamed editor work file CMCOPY to be copied into the new work file CMEDIT, add the following card:

Under z/OS:	//CMCOPY DD...
Under z/VSE:	//DLBL CMCOPY...

- Run the Software AG editor work file formatting utility with the new file.
7. Check the job log for potential errors.
 8. Reallocate and (if necessary) reopen the editor work file.
 9. Use the Editor Buffer Pool Administration utility SYSEDT to check that the buffer pool and the work file have been restarted successfully.



Important: All Natural sessions must be restarted if you want them to use the editor after the buffer pool restart.

Maintaining the Editor Work File under BS2000

If you want to change the size of the editor work file, format a new editor work file and copy the recovery records from the old work file into the new one as follows:

1. Shutdown all systems that use the editor.
2. Terminate the editor buffer pool.
3. Rename the current editor work file.
4. Create a new editor work file with the original name.
5. Execute the editor work file formatting program with the COPY instead of the FORMAT function after having added:

Example:

```
/CAT NATEDT.WORKFILE,NATEDT.COPYFILE
/FILE NATEDT.WORKFILE,LINK=CMEDIT,SPACE=nnn
/LOGON
/FILE NATEDT.COPYFILE,LINK=CMCOPY
/FILE NATEDT.WORKFILE,LINK=CMEDIT
/SYSFILE SYSLST=LST.NATEDFM2
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (NATEDFM2,NATvrs.MOD)
COPY
/LOGOFF N
```

where *vrs* represents the relevant product version.

See also *Installing the Software AG Editor on z/OS*, *Installing the Software AG Editor on z/VSE*, *Installing the Software AG Editor on BS2000*)

Editor Work File under Complete/SMARTS

SMARTS work files are located in the SMARTS Portable File System. The path must be specified with the SMARTS environment variable `$NAT_WORK_ROOT`. The name of the editor work file is specified with the EDBP subparameter `DDNAME`.

Formatting of an editor work file is only possible during buffer pool initialization (online). There is currently no tool under SMARTS to format an editor work file offline.

49 Editor Buffer Pool

- Purpose of the Editor Buffer Pool 350
- Obtaining Free Blocks 351
- Initializing the Editor Buffer Pool 351
- Restarting the Editor Buffer Pool 352
- Editor Buffer Pool Parameters 352
- Buffer Pool Initialization for Multi-User Environments 352

This document describes purpose, use and operation of the Editor Buffer Pool which is an intermediate main storage area used by the Software AG Editor.

Purpose of the Editor Buffer Pool

The editor buffer pool can be seen as an extension of the editor buffer (*SSIZE*). It is an intermediate main storage area used by the Software AG Editor to maintain its logical files.

A logical file consists of one or more logical records and contains the data of a Natural source object or a file (for example, a job, a PDS member or an LMS element) maintained by the editor. As a user can work with more than one object at the same time, several logical files can exist concurrently for each user.

The number of logical files (as well as the percentage of recovery records in the **Editor Work File**) is defined in the buffer pool parameter macro.

The editor buffer pool can be defined as a local or a global (*z/OS* and *BS2000* only) or an auxiliary (*EDPSIZE*) buffer pool. In multi-user environments (*CICS*, *IMS TM*, *openUTM*), the editor buffer pool is shared by all editor users of either the same region (local pool) or more than one region (global pool).

The editor buffer pool contains various control tables and a number of data blocks:

Area	Size
Main control block	500 bytes
Logical file table	20 bytes per logical file
Work file table	4 bytes per record
Recovery file table	16 bytes per record
Buffer pool block table	28 bytes per block
Buffer pool blocks	see text below

As the size of a buffer pool block is equal to the size of a work file record, one buffer pool block can contain one logical file record.

The buffer pool is initialized by the first editor user. During warm start buffer pool initialization, all recovery records are checked to build the recovery file table.

Several functions are provided to access the buffer pool (for example, functions to allocate, read, write or delete a record).

Obtaining Free Blocks

If the buffer pool becomes full, buffer pool blocks have to be moved to an external data set, the editor work file, to obtain free blocks.

In such a situation, the editor checks all logical files for their timeout value and deletes any logical file which has not been accessed within the specified time. This means that all its buffer pool blocks and work file records are freed, and the logical file is lost.

If there is still no buffer pool block available, the editor moves the oldest block to the work file, according to the specified timeout parameter values (see the *Generation Parameters* function of the *SYSEDT Utility* in the *Natural Utilities* documentation).

Initializing the Editor Buffer Pool

An uninitialized editor buffer pool is initialized when the Software AG editor is called for the first time. Then the various control blocks are created. There are two different modes of buffer pool and work file initialization: “cold start” and “warm start”.

Buffer Pool Cold Start

A buffer pool cold start can be triggered by the editor buffer pool subparameter `COLD` or by the Editor Buffer Pool Administration utility `SYSEDT` or automatically (if the editor work file is unformatted).

During a buffer pool cold start, the values of the editor buffer pool parameter `EDBP` or the corresponding macro `NTEDBP` are stored into the work file control record and all work file recovery records are cleared.

Buffer Pool Warm Start

During a buffer pool warm start, the buffer pool parameters are read from the work file control record and all work file recovery records are read to build the recovery file table in the buffer pool.

The accessed editor work file can be used only by sessions within the same operating system (system variable `*HOSTNAME`) and with the same global buffer pool or the same job name for local buffer pools. This connection can be dropped by a buffer pool cold start only. Alternatively, the Software AG editor work file formatting utility can be used to reset the work file connection.

Restarting the Editor Buffer Pool

The Editor Buffer Pool Administration utility `SYSEDT` can be used to terminate the editor buffer pool, that is, to set it to the uninitialized state. This avoids the restart of the TP system or of the global buffer pool.

If `SYSEDT` is not available due to buffer-pool problems, the program `BPTERM` can be used to terminate the buffer pool.

 **Important:** All Natural sessions must be restored if you want them to use the editor after buffer-pool restart.

Editor Buffer Pool Parameters

The editor buffer pool parameter `EDBP` or the corresponding macro `NTEDBP` in the [Natural parameter module](#) is required to define parameters for the operation of the editor buffer pool.

When the editor work file is formatted, these parameters are stored into the work file control record while all other records are cleared. Thus, reformatting a work file that has been previously used, means that all editor checkpoint and recovery information is lost.

Some of these parameters can be modified dynamically during execution of the buffer pool by using the Editor Buffer Pool Administration utility `SYSEDT`.

Buffer Pool Initialization for Multi-User Environments

During the buffer pool initialization, all recovery records are read from the editor work file. Therefore, the first users have to wait for a long time or even receive a timeout message until the editor buffer pool initialization is finished.

For this reason, a special Natural program has been supplied to trigger the buffer pool initialization before the first user becomes active. This program can be activated either during the startup of the TP monitor, or by a batch job if a global buffer pool is used.

The session must then be started with the session parameter:

```
STACK=(LOGON SYSEDT, user,password;BPINIT;FIN)
```

Under CICS: If the session runs asynchronously, `SENDER=CONSOLE` must be specified to obtain any error messages issued during initialization. The source program `FRONTPLT` is supplied as a sample program to show you how to start an asynchronous Natural session during CICS startup via `PLTPI`.

XIV

Natural Net Data Interface NATNETTO

50

Natural Net Data Interface NATNETTO

▪ Natural Net Data Driver Functional Description	358
▪ General Message Layout	359
▪ Layout of Header	359
▪ Format Buffer Layout	363
▪ Value Buffer Layout	366
▪ Attribute Buffer	366

This document provides information on the Natural Net Data Interface and the `net data` protocol definition.

See also:

- *Installing Natural Net Data Interface on z/OS* in the *Installation for z/OS* documentation.
- *Installing Natural Net Data Interface on BS2000* in the *Installation for BS2000* documentation.

Natural Net Data Driver Functional Description

The Natural Net Data Driver `NATNETTO` is a component that was introduced to support the EntireX CICS 3270 Bridge and similar client/server solutions in message oriented server environments, that is, TP monitors.

`NATNETTO` implements a protocol driver, which allows program-to-program communication with Natural (legacy) applications from client applications, using a `net-data` protocol. One typical scenario is a desktop client (for example, built with Natural for Windows or VBA) accessing a Natural application that runs under a TP monitor such as CICS, IMS TM or *openUTM*.

“Net data” means, that the protocol neither contains format data such as text constants nor any device-dependent control sequences. All data is communicated in printable format. This implies that eventually necessary marshaling and unmarshaling of non-alpha fields has to be done by the clients.

Basically, the protocol consists of two parts:

- A header or control block and a value buffer which contains the raw net data. This part is mandatory. The header contains control, environment and session information and maintains pointers to the other parts of the data buffer. The value buffer contains the actual net data which is to be exchanged between client and server.
- In addition, optional variable parts are available: format buffer and/or attribute buffer. The optional format buffer has an entry with descriptive data for each field in the value buffer. The attribute buffer consists of one byte with a preset value of 0 for each field in the value buffer. The client has to switch this value to 1 for each modified field, if the appropriate option is set, thus emulating the setting of MDT bits.

Header, value buffer and attribute buffer are parts of outbound and inbound messages; only the format buffer may occur in the outbound message only. The header maintains a transaction number which has to be mirrored by the client for flow-control purposes. Since legacy applications are mostly designed to be driven from block mode terminals, the protocol supports 3270 like functionality such as PF keys and cursor position.

General Message Layout

The following parts of the general message layout are mandatory:

- Header (the first two rows in the table below)
- Value buffer

The following parts are optional:

- Format buffer
- Attribute buffer

FSCB	Value Buffer Offset	Format Buffer Offset	...
	Attribute Buffer Offset	Aid Char.	Cursor Pos. ...
Value Buffer			
Format Buffer			
Attribute Buffer			

For detailed information on the layout parts, refer to Table 1.

Layout of Header

Table 1: Control Block - Fixed Part

Field	Format	Scope	Meaning
Eyecatcher	A4	FSCB	Eye catcher
Product code	A3	-	Product identification
Protocol version	N2	01 - 99	Version for specific product
Value buffer offset	N10	calculated	Value buffer offset from start of message
Format buffer offset	N10	calculated	Format buffer offset from start of message
Total message length	N10	calculated	Cumulated length of all buffers
Message number	N6	incremented by 1 every call	Echoed by communication partner
Block number	N5	01 - 99 (normally 01)	For block splitting within one message
Number of parameters	N5	calculated	Number of parameters in VB
Session token	A32		Security token
Message format	A1	see Table 2	Mode of field separation within value buffer
Delimiter character	A1	-	

Field	Format	Scope	Meaning
Architecture	A2	see Table 3	Architecture of sending partner
Call type	A2	see Table 4	Type of current call
Response code	N4	0001 - 9999	Response code from client
Block status	A1	L or N	Block is last one of msg or a next one follows
Server name / TAC	A8	-	TP transaction code or name of server
Aid character	A2	see Table 5	Aid character depressed or generated on client
Cursor line	N3	1 - max phys. line on client	Cursor line or 000 *)
Cursor column	N3	1 - max phys. col. on client	Cursor column or cursor field number *)
Attribute buffer offset	N10	calculated	AB offset from start of message
Timestamp	A16	generic	Store clock value: map stow time hex printable
DBID	N5	1 - 32767	DBID of FNAT on server
File number	N5	1 - 32767	File number of FNAT on server
Date form	A1	I, G, E, U	Date format according to Natural
Decimal character	A1	-	Natural delimiter character on server
Input delimiter char.	A1	-	Natural input delimiter character (server)
Control character	A1	-	Natural control character (server)
Language code	N2	01 - 99	Natural language code (server)
Application ID	A8	-	Natural application ID
Program name / map	A8	-	Program in execution / map or format name
Error number	N5	00001 - 99999	Natural error number
Line number	N4	0001 - 9999	Line number of current I/O statement
Error state	A1	-	Status byte
Error program	A8	-	Object causing an error
Error level	N2	01 - 15	Subroutine level of object in error
Message type	A1	see Table 6	Type of message
Option flag 1	A1	see Table 7	Control flag
Option flag 2	A1	see Table 7	Control flag
Option flag 3	A1	see Table 7	Control flag
Option flag 4	A1	see Table 7	Control flag
Option flag 5	A1	see Table 7	Control flag
Option flag 6	A1	see Table 7	Control flag
Option flag 7	A1	see Table 7	Control flag
Option flag 8	A1	see Table 7	Control flag

*) If the cursor field number notation is set in NATCONFIG, the cursor line will always be 000 and the cursor column will contain the absolute number of the field, where the cursor shall be placed (outbound) or was located at send time (inbound).



Note: Not all header fields are currently used!

Table 2: Modes of Field Separation

Format A1

Value	Meaning
D	Delimited mode
F	Fixed format mode
L	Length field precedes field (N3)

Table 3: Architecture of Sending Partner

Mask in Format A2

Value	Meaning
- 1	Mask for low order byte first (Vax)
- 2	Unused
- 4	Mask for EBCDIC architecture
- 8	Mask for ASCII 8 architecture
1 -	Mask for float representation VAX
2 -	Mask for float representation IEEE

Table 4: Call Type

Format A2

Type of Communication	Value	Meaning
Natural net data/3GL	MD	Map data (net data using format)
	ND	Net data
	CM	Command mode (server)
	FD	Map-format download
	IP	Normal input statement
	CS	Close session termination message

Table 5: Aid Character Table

Format A2

Aid Char.	PF Key
EN	Enter
CL	Clear
P1	PA1
P2	PA2
P3	PA3
01	PF1
02	PF2
03	PF3
...	...
47	PF47
48	PF48
CS	Close Session



Note: CS - Close Session - allows clients to enforce an immediate close of the server session. Therefore, it is in fact not a real PF key, but a command code for the server.

Table 6: Message Type

Value	Meaning
D	Dialog message
A	Async. message
P	Printout message

Table 7: Option Flags for Natural Net-Data Communication

All flags are of format A1.

Flag	Values	Meaning
Option 1	F	Message includes format buffer (fb-option).
Option 2	S	Net data is generated from screen buffer.
	P	Net data is generated from page buffer.
Option 3	A	Message includes attribute buffer (ab-option).
Option 4	P	Data in VB is in presentation format (printable).
	I	Data in VB is in internal format of sender.

Flag	Values	Meaning
	A	Data is in internal format converted to alpha.
Option 5	M	Outbound message contains overlay part.
Option 6	1	Extended format buffer option 1.
	2	Extended format buffer option 2.
Option 7		For future use.
Option 8		For future use.

Format Buffer Layout

Base Part

Each format buffer entry is a variable length string consisting of four elements:

- Identifier
- Protection indicator
- Format indicator
- Printable field length

Table 8: Format Buffer Entry

Element	Value	Meaning
Identifier	F	Field
	S	Subfield
Protection	M	Modifiable field
	O	Output only field, protected field
Format	A	Alpha
	N	Numeric
Field length	L - LLL, L	Length specification according to Natural standard

Examples:

FMA20	Field, modifiable, format alpha 20
SMN12,4	Subfield, modifiable, format numeric 12.4
0	Output only field, protected field

 **Note:** The precision part of a numeric length is always separated by a comma (,), regardless of the current values of delimiter and decimal character profile parameters! For alpha type fields the precision part is omitted.

Subfields are used to determine fields which had been separated out of a base field using the Natural dynamic attribute facility. If a field is dynamically divided into various subfields, this is marked as follows:

The first subfield is marked with identifier F as usual, all other subfields are identified by S.

Extension 1

The following figure shows a part of a DSECT, which describes layout of the Natural internal screen attribute buffer. If the format buffer extension option 1 is set, for each field those attribute bytes (PATTR1 - PATTR4) will be brought into printable format and added to the appropriate fields format buffer entry. The extension is separated by a "/" (slash) from the base format entry.

PATTR1	DS	X		ATTRIBUTE BYTE 1
P1TMP	EQU	X'80'	1000 0000	TEMPORARY PROTECTED (ONLY PAGE)
P1EXTLNG	EQU	X'80'	1000 0000	EXTENDED LENGTH (ONLY SCREEN)
P1RPA	EQU	X'40'	0100 0000	FIELD CAN BE REPEATED
P1PROT	EQU	X'20'	0010 0000	FIELD IS PROTECTED
P1NUM	EQU	X'10'	0001 0000	FIELD IS NUMERIC
P1SKIP	EQU	P1PROT+P1NUM (X'30')		FIELD WILL BE SKIPPED AUTOMATICALLY
P1HIGH	EQU	X'08'	0000 1000	FIELD IS HIGHLIGHTED
P1BLINK	EQU	X'04'	0000 0100	FIELD IS BLINKING
P1NOND	EQU	P1HIGH+P1BLINK (X'0C')		FIELD IS NON-DISPLAY
P1NHC	EQU	X'02'	0000 0010	FIELD MAY NOT BE PRINTED
P1CURS	EQU	X'01'	0000 0001	SET CURSOR HERE (ONLY UNPROT)
	SPACE			
PATTR2	DS	X		ATTRIBUTE BYTE 2
P2ITAL	EQU	X'80'	1000 0000	ITALIC/CURSIVE
P2MAND	EQU	X'40'	0100 0000	INPUT MANDATORY

P2MFILL	EQU	X'20'	0010 0000	MANDATORY FILL
P2LC	EQU	X'10'	0001 0000	DO NOT TRANSLATE (LOWER CASE)
P2CS2	EQU	X'08'	0000 1000	SECOND CHARACTER SET
P2UL	EQU	X'04'	0000 0100	UNDERLINED
P2RVID	EQU	X'02'	0000 0010	REVERSED VIDEO
P2RL	EQU	X'01'	0000 0001	RIGHT-LEFT
	SPACE			
PATTR3	DS	X	COLOR ATTRIBUTE	ATTRIBUTE BYTE 3
P3TP	EQU	X'80'	1000 0000	TERMINAL PROGRAM AVAILABLE
P3PFK	EQU	X'40'	0100 0000	*COM FIELD
P3NUM	EQU	X'20'	0010 0000	NUMERIC FIELDS
P3HELPR	EQU	X'10'	0001 0000	HELP ROUTINE AVAILABLE
P3FRAME	EQU	X'08'	0000 1000	FRAME ATTRIBUTE
P3NEUTR	EQU	X'07'	0000 0111	NEUTRAL
P3YELL	EQU	X'06'	0000 0110	YELLOW
P3TURQ	EQU	X'05'	0000 0101	TURQUOISE
P3GREEN	EQU	X'04'	0000 0100	GREEN
P3PINK	EQU	X'03'	0000 0011	PINK
P3RED	EQU	X'02'	0000 0010	RED
P3BLUE	EQU	X'01'	0000 0001	BLUE
*		FBI (DB)		(FIELD PROCESSING INFORMATION)
	SPACE			
PATTR4	DS	X		INTERNAL PROCESSING ATTRIBUTES
P4TEXT	EQU	X'80'	1000 0000	FIELD IS TEXT CONSTANT
P4SAME	EQU	X'40'	0100 0000	SAME ATTRIBUTE AS BEFORE
P4NATTR	EQU	X'20'	0010 0000	FIELD NEW ATTRIBUTE
*				PAGE BUFFER, DYNAMIC ATTRIBUTE
P4OVL	EQU	X'10'	0001 0000	FIELD BELONGS TO OVERLAY BUFFER
P4MDT	EQU	X'08'	0000 1000	FIELD HAS BEEN MODIFIED
P4MDTH	EQU	X'04'	0000 0100	UPDATE FROM HELP (PAGE BUFFER)
P4NFLD	EQU	X'04'	0000 0100	FIELD NEW ON SCREEN
*				IF SET FOR OVL, NEW LINE
P4CONT	EQU	X'02'	0000 0010	FIELD IS CONT OF BEFORE
P4LAST	EQU	X'01'	0000 0001	LAST ATTRIBUTE IN BUFFER
**P4HELP	EQU	P4TEXT+P4MDT		HELP REQUEST FOR THIS FIELD

Example:

An extended format buffer entry 18820300 means, the field is numeric and shall be presented highlighted italic in reversed video mode. The color of the field is pink!

Value Buffer Layout

Three modes of value buffer structure are possible:

■ **Fixed Format**

All parameters are simply concatenated without any delimitation. This means, that the single parameters have to be separated either according to the format description in the format buffer or by covering them with a C-structure, a data area or a DSECT.

■ **Delimited Format**

The parameters are separated by an configurable delimiter character.

■ **Length Preceded Format**

Each parameter is preceded by a length field of format N3. The length notation is explicit.

Attribute Buffer

The attribute buffer is optional. It consists of a one-byte entry for each parameter field, which represents the MDT flag. The MDT has to be set by the client for each modified field. The value of this flag is "0" or "1". A value of 1 means the MDT is set.

Example:

This example shows the screen image of a 3270 format in Figure 1 and the generated net-data stream for the same format in Figure 2. The name of the Natural map is NETM002.

```

                                TESTMAP NWI

AL20.0 ABCDEFGHIJKLMNOPQRST
NL20.0 1234567890
NL10.4 0000001234.5678
AL20C AAAAABBBBBCCCCDDDDZ
N20.0 999999999999999999

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
      Help                               -      +                               %%

```

Figure 1: NETM002 on a 3270 Device

```

FSCBNAT010000000206000000038000000004710000120000100006
  F 04MD0000LNATvrsXSEN0000010000000465B3E0C25A1A1DE4000000000000I.,%01NETT
0 NETM002 000000170  D  FSAP 1 ABCDEFGHIJKLMNOPQRST1234567890
  0000001234.5678AAAAABBBBBCCCCDDDDZ99999999999999999999
                                FMA20/08100024FOA20/
38102024FMA15/08102024FOA20/38101624FOA20/38102024FOA79/70000035.000000

```

where *vrs* represents the relevant product version.

Figure 2: Net-Data Stream Generated from NETM002 Execution

Configuration Settings: Fixed format, format buffer + extended format buffer, attribute buffer option, cursor position represented as field number.

XV

Selectable Units for New Natural Features

51

Selectable Units for New Natural Features

Natural selectable units provide the option to use selective new or changed Natural features as required instead of completely upgrading Natural.

Selectable units are implemented as NATSUPGM module which is loaded on request during Natural session start. However, if you want to use selectable units under CICS TS, you need to perform the optional installation step described in *Selectable Units Module NATSUPGM* in *Installing Natural CICS Interface on z/OS* in the *Installation* documentation.

➤ To select and activate or deactivate selectable units

- Set the SELUNIT profile parameter as described in the *Parameter Reference* documentation.

➤ To list all available selectable units and their operational status

- Issue the SHOWSU system command described in the *System Commands* documentation.

The **SHOWSU Selectable Units** screen appears with a list of all selectable units available in your environment and indicates their current status (available and/or active) as specified with the SELUNIT profile parameter.

➤ To list active selectable units only

- 1 If you only want to list all selectable units that are currently active in your environment, issue the SYSPROD system command (see the *System Commands* documentation).
- 2 On the **Installed Products** screen, enter the line command SU in the **Product Name** column next to Natural.

If a selectable unit is active in your environment, a window opens indicating the number of the unit and the Natural feature supported by this unit. Otherwise, the window shows the message `No unit activated.`

If you enter the command `SU` for a **Product Name** that does not support Natural selectable units, a corresponding message appears.

XVI

Natural as a Server

This part describes the use of Natural as a Server under the operating systems z/OS and z/VSE in batch mode, and under the TP monitor CICS.

Natural as a Server under z/OS Explains how Natural can act as a server in a client/server environment under z/OS in batch mode.

Natural as a Server under z/VSE Explains how Natural can act as a server in a client/server environment under z/VSE in batch mode.

Natural as a Server under CICS Explains how Natural can act as a server in a client/server environment under the TP monitor CICS; describes the functionality and the installation of the Natural CICS Interface in a server environment and informs about restrictions that apply in such an environment.

52 Natural as a Server under z/OS

▪ Functionality	376
▪ Natural Nucleus Installation in a Server Environment	377
▪ Print and Work File Handling with External Data Sets in a Server Environment	377

This document applies under z/OS only.

Functionality

Besides being a programming language, Natural can also act as a server in a client/server environment. It can provide services, such as the execution of Natural subprograms. Part of the server functionality is the enhanced batch driver. There are a lot of underlying protocols for the client/server communication, such as the execution of stored procedures for DB2 and the execution of remote procedure calls, see the *Natural RPC (Remote Procedure Call)* documentation.

Natural Server Stub

Natural as a server runs in a separate region or within the server subsystem region, for example, for DB2 stored procedures. To run Natural as a server, a service-specific server stub is required. This server stub is supplied as part of the server product. It controls all service requests and is the only interface to the Natural server front-end.

There are different server stubs for DB2, for Natural RPC and for others.

Natural Batch Driver

The Natural batch driver (that is, for example, NATOS under z/OS) has been enhanced to act as the environment-specific interface component which maintains the Natural server sessions and supplies environment-specific services to Natural. It can be linked to the server stub module or loaded by the server stub as a separate module.

The batch driver is able to create and to control multiple sessions by using storage threads including functionality for thread storage compression, decompression and rollout to external storage devices.

When the batch driver is called by the server stub for the first time (during server initialization), the storage threads are created in main storage. The number and size of the storage threads is determined by the server stub. Then a static Natural session is initialized. This includes profile parameter evaluation and the allocation of static storage buffers. The resulting pre-initialized storage thread is saved in main storage separately. For every new Natural session, this initial 'session clone' is copied into the thread.

When decided by the server stub, a session can be rolled out to be resumed at a later point of time. The **Natural Roll Server** is used by the driver to save the compressed thread storage of a session. As an alternative, main storage can be used to save the compressed thread storage. In this case, the number of sessions in rolled-out state is limited by the region size.

Natural Nucleus Installation in a Server Environment

The Natural nucleus and its batch driver are designed to support both, server and non-server environments. For the server-specific definitions and requirements, please refer to the specific documentation (for example, to the *Natural RPC (Remote Procedure Call)* documentation or to the *Natural for DB2* documentation).

If the number of sessions is not limited to a small number and if the server type supports session rollout, the **Natural Roll Server** must be installed and be started before the server initializes. To do this, ensure that the `SUBSID` parameter in the **Natural parameter module** is set to the correct value. For the server, the Adabas link interface (`ADALNK`) must be generated so that `ADALNK` is also reentrant, in addition to the server.

You can use a local or a global **Natural buffer pool**. If you define a local buffer pool, it will be shared by all sessions within the server region.

If a logical print or work file number is to be used for processing within any server session, it must be associated with an access method at session start time. This can be done in the **Natural parameter module** with the macros `NETWORK` and `NTPRINT`, as in the following example, if you want to allow the full range of all print and work file numbers possible:

```
NTPRINT (1-31),AM=STD,OPEN=ACC,DEST=*
NETWORK (1-32),AM=STD,OPEN=ACC,DEST=*
```

The subparameter `DEST=*` defines generic DD name generation during the first `DEFINE WORK FILE` or `DEFINE PRINTER` statement, `OUTPUT` clause (see below). Subparameter `OPEN=ACC` avoids pre-opening of the files at program start time. The open is issued upon the first access of the file.

Print and Work File Handling with External Data Sets in a Server Environment

When running many concurrent sessions in one region, there may be resource conflicts with external print and work files. The logical names (DD names) for print and work files are defined by the subparameter `DEST` of macro `NTPRINT`, respectively `NETWORK` or its dynamic equivalents, `PRINT` or `WORK` (defaults `CMPRTnn` and `CMWKFnn`). For normal Natural batch processing, these files are defined in JCL by a logical (DD) and a physical data set name.

However, DD names are reserved by the operating system for exclusive use by one task, respectively session, that is, if `CMWKF01` is opened by one session for processing, no other session could use this file until it is closed again. Other sessions would get an error if they would try to open it.

In a server environment, all print and work file requests are handled by a dedicated I/O subtask. This ensures data set integrity and avoids resource contention. It enables the shared usage of print and work files across Natural session boundaries, that is, multiple sessions can access the same

file concurrently. This is true only for print and work files whose DD-name starts with CM. All other files are considered as exclusive and cannot be shared.

For exclusive usage of print and work files, Natural offers the following two features to support print and work files in a server environment (both require a special implementation within the Natural application programs for the server environment):

- `DEFINE WORK FILE` or `DEFINE PRINTER` statements, `OUTPUT` clause and
- dynamic data set allocation (application programming interface `USR2021N`, see *SYSEXT - Natural Application Programming Interfaces*).

The `DEFINE WORK FILE` and the `DEFINE PRINTER` statement `OUTPUT` clause can be used

- to define the logical DD name for a work or print file, or
- to define the physical data set name, or
- to define an output spool class.

If a DD name is specified, the access method checks whether the data set is allocated. If not, an error is issued. The data set can be allocated by any Natural program using the `USR2021N` subprogram supplied in library `SYSEXT`.

If a physical data set name or a spool file class is specified, the access method itself allocates the data set dynamically during the execution of the `DEFINE . . .` statement. To ensure that a unique DD name is used, `DEST=*` should be predefined in the **Natural parameter module**. This avoids any DD name conflicts.

If the application is using the application programming interface `USR2021N`, it may specify an asterisk value for the DD name variable to get back a unique DD name from the access method. This DD name can be used for a subsequent `DEFINE . . .` statement.

By default, the access properties of the server job are used for print and work files. Some server types, for example, Natural Development Server and Natural RPC, support impersonation, that is, the access properties of the individual client account is used for exclusive print and work files. For more information, refer to the corresponding section in your server documentation.

53

Natural as a Server under z/VSE

▪ Functionality	380
▪ Natural Nucleus Installation in a Server Environment	381
▪ Print and Work File Handling with External Data Sets in a Server Environment	381

This document applies under z/VSE only.

Functionality

Besides being a programming language, Natural can also act as a server in a client/server environment. It can provide services, such as the execution of Natural subprograms. Part of the server functionality is the enhanced batch driver. There are a lot of underlying protocols for the client/server communication, such as the execution of stored procedures for DB2 and the execution of remote procedure calls, see the *Natural RPC (Remote Procedure Call)* documentation.

Natural Server Stub

Natural as a server runs in a separate region or within the server subsystem region, for example for DB2 stored procedures. To run Natural as a server, a service-specific server stub is required. This server stub is supplied as part of the server product. It controls all service requests and is the only interface to the Natural server front-end.

There are different server stubs for DB2, for RPC and for others.

Natural Batch Driver

The Natural batch driver (that is, for example, NATVSE under z/VSE) has been enhanced to act as the environment-specific interface component which maintains the Natural server sessions and supplies environment-specific services to Natural. It can be linked to the server stub module or loaded by the server stub as a separate module.

The batch driver is able to create and to control multiple sessions by using storage threads including functionality for thread storage compression, decompression and rollout to external storage devices.

When the batch driver is called by the server stub for the first time (during server initialization), the storage threads are created in main storage. The number and size of the storage threads is determined by the server stub. Then a static Natural session is initialized. This includes profile parameter evaluation and the allocation of static storage buffers. The resulting pre-initialized storage thread is saved in main storage separately. For every new Natural session, this initial 'session clone' is copied into the thread.

When decided by the server stub, a session can be rolled out to be resumed at a later point of time. A roll cache is used by the driver to save the compressed thread storage of a session.

Natural Nucleus Installation in a Server Environment

The Natural nucleus and its batch driver are designed to support both, server and non-server environments. For the server-specific definitions and requirements, please refer to the specific documentation (for example to the *Natural RPC (Remote Procedure Call)* documentation or to the *Natural for DB2* documentation).

You can use a local or a global **Natural buffer pool**. If you define a local buffer pool, it will be shared by all sessions within the server region.

If a logical print or work file number is to be used for processing within any server session, it must be associated with an access method at session start time. This can be done in the **Natural parameter module** with the macros `NTWORK` and `NTPRINT`, as in the following example, if you want to allow the full range of all print and work file numbers possible:

```
NTPRINT (1-31),AM=STD,OPEN=ACC,DEST=*
NTWORK (1-32),AM=STD,OPEN=ACC,DEST=*
```

The subparameter `DEST=*` defines generic DD name generation during the first `DEFINE WORK FILE` or `DEFINE PRINTER` statement, `OUTPUT` clause (see below). Subparameter `OPEN=ACC` avoids pre-opening of the files at program start time. The open is issued upon the first access of the file.

Print and Work File Handling with External Data Sets in a Server Environment

When running many concurrent sessions in one region, there may be resource conflicts with external print and work files. The logical names (DD names) for print and work files are defined by the subparameter `DEST` of macro `NTPRINT` or `NTWORK`, or its dynamic equivalents `PRINT` or `WORK` (defaults `CMPTnn` and `CMWKFnn`). For normal Natural batch processing, these files are defined in JCL by a logical file and a physical data set name.

However, DD names are reserved by the operating system for exclusive use by one task, respectively session, that is, if `CMWKF01` is opened by one session for processing, no other session could use this file until it is closed again. Other sessions would get an error if they would try to open it.

In a server environment, all print and work file requests are handled by a dedicated I/O subtask. This ensures data set integrity and avoids resource contention. It enables the shared usage of print and work files across Natural session boundaries, that is, multiple sessions can access the same file concurrently.

For exclusive usage of print and work files, Natural offers the following feature to support print and work files in a server environment (both require a special implementation within the Natural application programs for the server environment):

- `DEFINE WORK FILE` or `DEFINE PRINTER` statements, `OUTPUT` clause

The `OUTPUT` clause of these statements can be used

- to define the logical file name for a work or print file, or
- to define the physical data set name.

If a physical data set name or a spool file class is specified, the access method itself allocates the data set dynamically during the execution of the `DEFINE . . .` statement. To ensure an unique file name is used, `DEST=*` should be predefined in the [Natural parameter module](#). This avoids any file name conflicts.

54 Natural as a Server under CICS

▪ Functionality	384
▪ Natural CICS Interface Installation in a Server Environment	384
▪ Restrictions	385

This document applies under CICS only.

See also:

- *Natural under CICS* in the *TP Monitor Interfaces* documentation
- *Natural RPC (Remote Procedure Call)* documentation

Functionality

Natural as a Server

Besides being a programming language, Natural can also act as a server in a client/server environment. It can provide services, such as the execution of Natural subprograms. There are a lot of underlying protocols for the client/server communication, such as the execution of stored procedures for DB2 (see *Natural for DB2*) and the execution of remote procedure calls (see *Natural RPC (Remote Procedure Call)*).

Natural Server Stub

Natural as a server runs in a separate region or within the server subsystem region, for example for DB2 stored procedures. To run Natural as a server, a service-specific server stub is required. This server stub is supplied as part of the server product. It controls all service requests and is the only interface to the Natural server front-end.

There are different server stubs for DB2, for RPC and for others.

Natural CICS Interface Installation in a Server Environment

There is nothing specific to define when installing the Natural CICS Interface in order to serve as a Natural server environment. There are no requirements on thread type or type of rolling (CICS roll facilities or roll server).

Actually, Natural server sessions may share a Natural under CICS environment with “normal”, for example, terminal bound Natural sessions. The difference is that, in case of a Natural server session, the Natural CICS Interface does not deal with a principal facility, such as a terminal or printer, but with a server stub. In terms of CICS, a Natural server session is a series of asynchronous CICS tasks, and the session context (session restart data) is maintained by the server stub using a unique 8-byte session ID.

Restrictions

The following restrictions apply when Natural is used as a server under CICS:

1. Natural server sessions under CICS can only run in pseudo-conversational mode. A Natural server session cannot run in conversational mode, as the Natural CICS Interface always has to pass control back to the server stub; therefore `PSEUDO=ON` is forced for Natural server sessions under CICS. Because of the same reason `RELO=ON` is forced for Natural server sessions using `TYPE=GETM` threads.
2. 3GL programs called by Natural should be aware of the fact that Natural server sessions are running asynchronously in CICS, that is, no CICS terminal (TCTTE) is available.
3. The profile parameter `ADAMODE` should be set to 1 or 2, otherwise Adabas may build a different UQE ID for each dialog step of the Natural server session.
4. The profile parameter `PROGRAM` or equivalent back-end program settings by Natural are not honored, as the logic flow at session termination from the Natural CICS Interface to the server stub must not be interrupted and/or falsified by a potential back-end program.
5. Care should be taken when using the parameter `TERMVAR (&TID)` in the macro `NCMPRM` or `NTCICSP` (depending on the Natural CICS Interface version installed) in the file name setting for Natural print and work files: As a Natural server session runs asynchronously, there is no (unique) terminal ID or other unique four-character session identifier to insert. In CICS/TS 1.3 and above, the CICS Interface internally uses the `QNAME` option when dealing with CICS temporary storage for such Natural print and work files, that is internally a 16-byte temporary storage queue name is used (the 8-byte unique server session ID is appended to the file's `DEST` specification). This means on the other hand that such CICS temporary storage queues can only be accessed by the originating session.

XVII

Natural Execution - Miscellaneous Topics

This part provides general information on Natural execution.

Natural 31-Bit Mode Support

Support and Use of Natural and Non-Natural Objects

Input/Output Devices

Double-Byte Character Sets

Asynchronous Processing

For explanations of the terms used in this document, see the *Glossary*.

55

Natural 31-Bit Mode Support

In general, Natural runs with the following settings:

```
AMODE=31
```

```
RMODE=ANY
```

Exceptions to this are described with the corresponding environment documentation.

56 Support and Use of Natural and Non-Natural Objects

- Support for Natural Objects from Previous Natural Versions 392
- Back-End Program Calling Conventions 392
- LE Subprograms 394
- External Sort Programs 397

Support for Natural Objects from Previous Natural Versions

Natural objects created in an earlier version of Natural (from version 2.2 onwards) can be executed in the current Natural version without any adjustments to the objects or any conversion or migration procedure being required. This also applies to objects that have been cataloged with the Natural Optimizer Compiler.

Back-End Program Calling Conventions

This section describes the conventions that apply to invoking a back-end program.



Notes:

1. This section does not apply to BS2000; refer to *Calling Non-Natural Programs* and *Calling openUTM Chained Partial Programs* in the *Natural TP Monitor Interfaces* documentation, section *Natural under openUTM*.
2. Except under z/OS in batch mode, a specified back-end program is *not* invoked if the Natural session is executing on a Natural Development Server.

This section covers the following topics:

- [Back-End Program Calling Conventions \(Batch Mode\)](#)
- [Special Considerations under CICS](#)
- [Special Considerations under IMS TM](#)
- [Sample Back-End Programs](#)

Back-End Program Calling Conventions (Batch Mode)

If the profile parameter `PROGRAM` is specified (or set dynamically during a Natural session by calling the subprogram `CMPGMSET` in the library `SYSEXTP`), a back-end program is invoked, regardless of whether the session terminated normally or abnormally. The back-end program is called using standard OS linkage conventions and must return the control to its caller.

If a back-end program is available, Natural does not issue any session termination messages. Non-zero user return codes, specified via *operand1* of the Natural `TERMINATE` statement, are indicated by the Natural error message NAT9987.

A parameter area containing the following information is passed to the back-end program:

- a fullword that holds the Natural system or user return code,
- a Natural termination message of 72 characters,

- a fullword that holds the length of the Natural termination data (or zero),
- the termination data passed by *operand2* of the `TERMINATE` statement (if any).

The back-end program parameter area is at least 80 bytes long. The macro `NAMBCKP`, which contains a `DSECT` layout of the back-end program parameter area, is supplied in the Natural source library and can be used by Assembler back-end programs.

Special Considerations under CICS

Under CICS, the back-end program parameter data is passed in the `COMMAREA` and in the `TWA`. In the `TWA`, only 80 bytes are passed, containing return code and message, while the length field contains an address that points to the full back-end program parameter area. The same `TWA` is also provided if Natural has been invoked via `EXEC CICS LINK`; see also *Natural under CICS, Front-End Invoked via LINK* in the *Natural TP Monitor Interfaces* documentation.

If the parameter `BACKRPL=ALL` is set in the `NCPRM` or `NTCICSP` macro (depending on the Natural CICS Interface version installed), only the termination data is passed in the `COMMAREA`.

Special Considerations under IMS TM

Under IMS TM, the calling conventions for a back-end program are different in a dialog-oriented environment. There, the back-end program is called by a program-to-program switch and the name of the back-end program is used as an IMS TM transaction code. In this case, the Natural environment is terminated before the program-to-program switch takes place; see *Natural under IMS TM, Support of Natural Profile Parameter PROGRAM* in the *Natural TP Monitor Interfaces* documentation.

Sample Back-End Programs

The following table contains a number of sample programs:

Sample Back-end Program for Batch and TSO Environments in COBOL:		
LINKAGE	SECTION	
01	BACKEND-PARM-AREA.	
02	TERMINATION-RETURN-CODE	PIC S9(8) COMP.
02	TERMINATION-MESSAGE	PIC X(72).
02	TERMINATION-DATA-LENGTH	PIC S9(8) COMP.
02	TERMINATION-DATA	PIC X(100)
...		
PROCEDURE DIVISION USING BACKEND-PARM-AREA		

Sample Back-end Program for Batch and TSO Environments in Assembler:
--

```
BACKPROG CSECT
          SAVE   (14,12)
          LR     11,15
          USING  BACKPROG,11
          L      2,0(1)
          USING  BCKPARAM,2
          ...
          RETURN (14,12)
BCKPARAM NAMBCKP
          END
```

Sample Back-end Program for CICS in Assembler:

```
L         2,DFHEICAP
USING     BCKPARAM,2
...
BCKPARAM NAMBCKP
          END
```

Sample Back-end Program XNATBACK for Batch Mode (z/OS and z/VSE):

A sample program for batch mode is supplied as XNATBACK in the Natural source library. This program issues the Natural termination message on both SYSPRINT (z/OS) or SYSLST (z/VSE) and the operator console; potential termination data is printed on SYSPRINT or SYSLST in dump format.

LE Subprograms

This section applies to z/OS batch mode, z/VSE batch mode, IMS TM and TSO. It provides information on how Natural supports IBM Language Environment (LE) subprograms.

This section covers the following topics:

- [Support of IBM LE Subprograms](#)
- [Enabling Natural Support of LE Subprograms](#)
- [Passing LE Runtime Options](#)

- LE Abend Handling

Support of IBM LE Subprograms

To support IBM Language Environment (LE) subprograms, Natural must be prepared for the `CALL` statement to be able to call LE subprograms. LE subprograms can be static (profile parameters `CSTATIC` and `RCA`) or dynamic subprograms of Natural.

Dynamic subprograms of Natural (LE and non-LE) are loaded via LE services (`CEEFETCH` or `CEELOAD` macro). All dynamic subprograms loaded during a Natural session are deleted upon LE environment termination, i.e. during termination of the Natural session. That is, the profile parameter `DELETE` does not have any effect.

Enabling Natural Support of LE Subprograms

The following is required to be able to call IBM Language Environment (LE) subprograms from Natural:

1. When installing Natural, the corresponding driver must be generated as described in the appropriate installation steps in *Installing Natural CICS Interface on z/OS* and *Installing Natural CICS Interface on z/VSE* in the *Installation* documentation.

For LE enablement of Natural under CICS, see also the appropriate installation steps and the section *Natural CICS Interface and IBM Language Environment (LE)* in the *TP Monitor Interfaces* documentation.

For LE enablement of Natural under Com-plete, the `LE370` keyword subparameter of the `NTCOMP` macro must be set to `ON` (see the *Parameter Reference* documentation). See also the chapter *IBM Language Environment Considerations* in your *Com-plete* documentation.

2. The IBM LE runtime modules must automatically be included from the IBM LE library during the linkage editor step. There must not be any unresolved externals starting with "CEE". Do not set the linkage editor option `NCAL` for z/OS or `NOAUTO` for z/VSE.
3. Under z/OS batch, IMS TM and TSO, Natural can also call LE main programs, but only as dynamic subprograms. If an LE main program is to be called dynamically, this has to be indicated by specifying `SET CONTROL 'P=L'` before the `CALL` statement. Otherwise, the LE environment created by Natural will be terminated by the LE main program.

Passing LE Runtime Options

Under z/OS Batch and TSO:

You have three options:

1. You can pass LE run-time options by using the `PARM=` parameter in your JCL. The following applies:
 - The run-time options that are passed to the main routine must be followed by a slash (/) to separate them from the Natural parameters.
 - If you want to use a slash within your Natural parameters, then your Natural parameters must begin with a slash.

Example:

```
PARM=' /ID=/ , . . . '
```

2. You can pass LE run-time options by using the `CEEEOPTS` input data set in your JCL. With the use of `CEEEOPTS` the LE run-time options are also available to all subtasks. The use of `CEEEOPTS` is especially required with a Natural RPC server in batch mode.

Example:

```
//CEEEOPTS DD *  
POSIX(ON)  
/*
```

3. You can define LE run-time options by modifying and re-assembling the supplied source module `NATLEOPT`. For example, if you have any subprograms still running in 24 bit mode, set `SYSPARM(RMODE24)` as parameter for the assembler rather than changing `NATLEOPT`.

If you have other specific requirements for your LE subprograms, you can add the desired LE options for the `CEEEOPTS` macro in source module `NATLEOPT`.

Under z/VSE Batch:

You can pass LE run-time options by using the `PARM=` parameter in your JCL. The following applies:

- The run-time options that are passed to the main routine must be followed by a slash (/) to separate them from the Natural parameters.
- If you want to use a slash within your Natural parameters, then your Natural parameters must begin with a slash.

Example:

```
PARM=' / ID= / , . . . '
```

Under IMS TM:

You can pass LE run-time options by providing the region-specific run-time options load module `CEEROPT` in your `STEPLIB` concatenation. In addition, the LE library routine retention initialization routine `CEELRRIN` must be present on the `PREINIT` list of your region `JCL`.

The following is a sample definition of a `CEEROPT` load module that allows the execution of `AMODE(24)` subprograms:

```
CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
          CEEXOPT ALL31=((OFF),OVR) , X
          STACK=((128K,128K,BELOW,KEEP,512K,128K),OVR)
          END CEEROPT
```

LE Abend Handling

Natural supports the LE-specific user error handling, that is, if an LE subprogram has defined a user error handler, this handler gets control when an abend, a program check or any other LE error condition occurs in the subprogram. If no LE user error handler has been defined, Natural reacts according to the setting of the `DU` profile parameter.

In this case, a special error message (`NAT0950` if `DU=OFF` or `NAT9967` if `DU=ON`) is issued which indicates the LE error number. In addition, the corresponding LE error message is issued on `CEEMSG` and an LE snap dump is written to `CEEDUMP` according to LE run-time option `TERMTHDACT`.



Note: In case of `DU=FORCE`, the abend handling of Natural is disabled and the LE error handling takes place even if no LE subprogram is active at the time of the abend. In this case, it is strongly recommended to specify the LE run-time option `TERMTHDACT(UAImm)` to get all required diagnostic information.

External Sort Programs

This document provides information on using external sort programs with Natural.

The following topics are covered:

- [Support of External Sort Programs](#)
- [Special Considerations for z/OS](#)
- [Special Considerations for z/VSE](#)

- [Special Considerations for BS2000](#)

Support of External Sort Programs

The Natural `SORT` statement may optionally invoke an external sort program that carries out the actual sorting. An external sort program is used if the keyword subparameter `EXT` of the macro `NTSORT` is set to `ON` in the Natural parameter module.

Natural supports all external sort programs that comply with the sort interface documented in the manuals for the relevant operating system.

The requirements (for example, space and data sets) are identical to those for the execution of a 3GL (for example, COBOL, PL/I) application program that invokes the operating system sort program and can vary according to the external sort program in use.

The communication with the external sort program is via the E15 and E35 user-exit routines. As a consequence, Natural does not require the data sets `SORTIN` and `SORTOUT`.

Special Considerations for z/OS

All external sort programs supporting the extended parameter list can be used.

Special Considerations for z/VSE

The external sort program is loaded into the partition program area. For this reason, you must add round about 200 KB additional storage to the size requirements of the Natural batch nucleus specified in the `SIZE` parameter of the `EXEC` statement.

Example:

```
// EXEC natural,SIZE(natural,200K)
```

where `natural` is the name of your Natural phase.

Special Considerations for BS2000

The external sort program is called using the level 1 interface. That is, Natural passes all `SORT` control statements to the external sort program and data set `SYSDTA` is not used for input.

The external sort program is searched for in the following libraries:

- User `TASKLIB` concatenated with the `BLSLIB` chain, if a User `TASKLIB` was specified,
- System `TASKLIB` (`$TSOS.TASKLIB`) concatenated with the `BLSLIB` chain.

57

Input/Output Devices

▪ Terminal Support	400
▪ Light Pen Support	400
▪ Printer Support	401

This document provides some additional information on input/output devices supported by Natural.

Terminal Support

Natural supports a wide variety of terminal types for the use with mainframe computers. In TP monitor environments in which the terminal type information is not supplied automatically to Natural, you can use the Natural profile parameter `TTYTYPE` so that Natural can activate the appropriate converter routine to operate a specific type of terminal.

Links to related topics:

- [NTDVCE - Terminal-Device Specification Table](#)
- [Terminal Communication - Profile Parameters Grouped by Function](#) (*Parameter Reference* documentation)
- [NATCONFIG Module](#) (various I/O translation topics)
- [Terminal Types Supported under BS2000](#)
- [Natural Terminal Commands](#)

Light Pen Support

The support of light pens has been enhanced by the terminal command `%RM`. This command causes all light-pen-sensitive fields on the screen to be made write-protected; that is, the user can select them with a light pen, but cannot overwrite their contents.

For a field to be light-pen sensitive, it must be displayed intensified (session parameter `AD=I`) or blinking (`AD=B`), and the first character of the field must be a light-pen designator character (see below). Selecting a field with a light pen causes the designator character to be changed; therefore, you can make the processing of fields selected with a light pen dependent on the values of the designator characters.

The following designator characters are available:

Character	Meaning
?	You can select multiple fields before pressing ENTER.
>	It was selected and if it is selected again, it becomes a question mark ?; the characters ? and > will toggle.
&	You can select only one field and it will be as an ENTER for both the field and the MDT (modified data tag).

Character	Meaning
' ' (blank)	You can select only one field and you will only see the MDT.

As designator characters, you have to distinguish selection fields (?, >) and attention fields (&, blank or null). Selection fields do not start an immediate data transmission, so you are able to select more than one field. Attention fields result in an immediate action.

The `SELECT CURSOR` key emulates a light-pen selection. If you move the cursor to the field you want to select and press `SELECT CURSOR`, this field will be selected.

Sample Natural Program for Light Pen Usage

```

RESET #FIELD-1 (A8)
  #FIELD-2 (A8) #FIELD-3 (A8) #CV-1 (C) #CV-2 (C) #CV-3 (C)
SET KEY ALL
/* SET CONTROL 'RM' IS A TOGGLE. AFTER IT IS EXECUTED ONCE MAKE IT A
/* COMMENT, SO THAT YOU DO NOT TOGGLE IT 'OFF'.
**SET CONTROL 'RM'
REPEAT
  IF *PF-KEY NOT = 'ENTR' AND *PF-KEY NOT = 'PEN' ESCAPE BOTTOM
  MOVE (AD=I CD=YE) TO #CV-1
  MOVE (AD=I CD=RE) TO #CV-2
  MOVE (AD=I CD=BL) TO #CV-3
  MOVE ' FIELD-1' TO #FIELD-1
  MOVE '&FIELD-2' TO #FIELD-2
  MOVE '?FIELD-3' TO #FIELD-3
  INPUT (SG=OFF IP=OFF)
    01/01 #FIELD-1 (CV=#CV-1 AD=M)
    03/01 #FIELD-2 (CV=#CV-2 AD=M)
    05/01 #FIELD-3 (CV=#CV-3 AD=M)
  WRITE 'PF-KEY =' *PF-KEY
  IF #CV-1 MODIFIED WRITE '#CV-1 MODIFIED' #FIELD-1
  IF #CV-2 MODIFIED WRITE '#CV-2 MODIFIED' #FIELD-2
  IF #CV-3 MODIFIED WRITE '#CV-3 MODIFIED' #FIELD-3
LOOP
END

```

Printer Support

The following topics are covered:

- [Printer-Advance Control Characters](#)

- Natural Laser-Printer Support

Printer-Advance Control Characters

Printer-advance control characters can be generated within a Natural program by using the `DEFINE PRINTER` statement as follows:

```
....
DEFINE PRINTER (n) OUTPUT 'name'
DEFINE PRINTER (n+1) OUTPUT 'CCONTROL'
....
```

Both `DEFINE PRINTER` statements work together so that all Natural output for the printer (n) follows the normal Natural report-output rules and all Natural output for the printer ($n+1$) is also written to the printer (n). Natural does not generate a printer-advance control character for this report. Therefore, the first character in the output variable is the control character.

With this method, it is possible to merge control characters for laser-printer systems and channel-advance characters for line printers in a normal Natural output report.

Sample Natural Program for Printer-Advance Control Character

```
...
DEFINE PRINTER (1) OUTPUT 'CMPRT01'
DEFINE PRINTER (2) OUTPUT 'CCONTROL'
WRITE (1) 'TEST'
WRITE (2) NOTITLE '+TEST'
MOVE H'5A' TO A(A1)
WRITE (2) A '....'
...
```

The corresponding hexadecimal data in the spool file starting from column 0 are:

```
I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I
F1 E3 C5 E2 E3
1 T E S T
4E E3 C5 E2 E3
+ T E S T
5A ....
```

`CCONTROL` is the name of a special printer control table associated to the printer $n-1$; it must not be modified.

Natural Laser-Printer Support

Natural supports IBM 3800 laser-printer systems.

The `DEFINE PRINTER` statement is used to control and allocate a report for the 3800 printer system. With this statement, you can specify that the Natural print output for report 1 is routed to a 3800 printer system.

```
DEFINE PRINTER (1) OUTPUT 'LAS3800'
  I I => 1-31 for CMPRT01 to CMPRT31
  ....
```

Depending on the setting of the `INTENS` parameter, Natural repeats each line up to four times and recognizes the Natural attributes `AD=D`, `AD=I`, `AD=C` and `AD=V` (see session parameter `AD`).

The first line contains the ASA control code in the first column and the 3800-font control character (hexadecimal `F0`) for the first font in the second column. The columns 2 to `nnn` contain the print data which are not flagged with the attribute `AD=I`, `AD=C` or `AD=V`.

The second line contains the ASA control code + (for printing without line advance) in the first column and the 3800-font control character (hexadecimal `F1`) for the second font in the second column. The columns 2 to `nnn` contain the print data which are flagged with `AD=I`.

The third line contains the ASA control code + (for printing without line advance) in the first column and the 3800-font control character (hexadecimal `F2`) for the third font in the second column. The columns 2 to `nnn` contain the print data which are flagged with `AD=C`.

The fourth line contains the ASA control code + (for printing without line advance) in the first column and the 3800-font control character (hexadecimal `F3`) for the fourth font in the second column. The columns 2 to `nnn` contain the print data which are flagged with `AD=V`.

If `INTENS` is specified with a value less than 4, all non-supported fonts are printed with hexadecimal `F0`.

Sample Natural Program for Laser Printer Usage

```
....
DEFINE PRINTER (1) OUTPUT 'LAS3800'
WRITE (1) 'FIRST' 'SECOND' (AD=I) 'THIRD' (AD=C) 'FOURTH' (AD=V)
....
```

The corresponding hexadecimal data in the spool file starting from column 0 are:

```

I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I..I
40 F0 C6 C9 D9 E2 E3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 (hex)
    O F I R S T
4E F1 40 40 40 40 40 40 40 40 E2 C5 C3 E4 D5 C4 C4 40 40 40 40 40 40 40 (hex)
+ 1          S E C O N D
4E F2 40 40 40 40 40 40 40 40 40 40 40 40 40 40 E3 C8 C9 D9 D4 40 40 (hex)
+ 2          T H I R D
4E F3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 C5 (hex)
+ 3          F

```

Sample JCL for Laser Printer Usage

```

....
//xxxx JOB xxxxx,....
.
//xxxxx EXEC PGM= XXXXXX;.....
.
// PARM='INTENS=4,XXXX,.....'
.
.
//OUT1 OUTPUT PAGEDEF=XXXX,FORMDEF=XXXX,TRC=ON
.           I           I
.           I           I => 3800 form definition
.           I
.           I => 3800 page definition .
//CMPRT01 DD SYSOUT=Y
//          DCB=(RECFM=FBA,LRECL=133),OUTPUT=*,OUT1
//          CHARS=(WWW,XXXX,YYYY, ZZZZ)
.           I
.           I => IBM font names
....

```

58

Double-Byte Character Sets

▪ Natural Profile Parameter SOSI	406
▪ Output Format Specification	406
▪ Parameter Definitions for DBCS Support	406
▪ Editor Profile Options	407
▪ Input Data Check	407
▪ Output Data Adjustment	408
▪ Natural Stack Data	408
▪ Application Programming Interfaces for DBCS Handling	408
▪ Alternate Text Module NATTXT2U	409

This document is only relevant for Asian countries which use double-byte character sets. It describes all features implemented in Natural to support DBCS terminals and printers.

Natural Profile Parameter SOSI

In alphanumeric fields with SBCS and DBCS characters mixed, the DBCS character strings are separated from the SBCS strings by shift codes called SO (shift-out) and SI (shift-in). The Natural profile parameter SOSI is used to pass the values of the shift-in and shift-out codes used in the current environment to Natural.

It is strongly recommended to use the IBM characters X'0E' and X'0F' internally. With this technique, all applications and data can be handled in a compatible manner, which means that a network supporting different mainframe types can still use the same Natural applications and process the same data.

For detailed information on this parameter, see SOSI.

Output Format Specification

The Natural session parameter PM=D is used to define DBCS-only fields. A DBCS-only field must contain only valid DBCS characters; shift-out/shift-in characters (SO/SI) are not allowed within such a field. To display a field with the session parameter PM=D specified, the screen attribute X'43F8' is added for IBM terminals; for Fujitsu terminals, the field content is enclosed in the required shift-out/shift-in characters (SO/SI).

Parameter Definitions for DBCS Support

The following parameters must be specified in the setup for Natural for the support of double-byte character sets:

Parameter	Explanation
TS=ON	If Latin lower-case characters are not available, this parameter translates all Natural system output using the translation table defined by the macro NTTABL in the NATCONFIG module.
SOSI=(0E,0E,0F,0F,1)	Defines the DBCS shift-out and shift-in values for IBM hardware.
SOSI=(28,28,29,29,0)	Defines the DBCS shift-out and shift-in values for Fujitsu hardware.
LC=ON	Does not translate all input data to uppercase, which again would destroy possible DBCS input data.

In addition to `TS=ON`, further parameters to provide for translation of messages into upper case are provided by several Natural components. For detailed information, see *Other Parameters to Provide Upper Case Translation* in the `TS` profile parameter documentation.

Editor Profile Options

If you want to enter DBCS or half-width Katakana characters in one of the Natural editors, the following editor general default options should be set in the editor profile to avoid that character constants or field names containing DBCS or half-width Katakana characters are unintentionally converted to upper case:

Option	Value	Explanation
Editing in Lower Case	Y	Lower-case characters in the source code are not automatically converted to upper case. This option is required if you are using DBCS or half-width Katakana characters.
Dynamic Conversion of Lower Case	N	Any source code remains as you enter it. This option is required if you are using half-width Katakana characters.

For detailed information on the editor general default options, see *General Defaults*. For detailed information on the editor profile, see *Editor Profile* in the *Editors* documentation. To avoid the need to change these options for every user, you can modify the default profile for your installation by means of the user exit routine `USR0070P`, which also supports DBCS; see [USR0070P - User Exit for Editor Profiles](#) in the section *Configuring Natural*.

Input Data Check

If the session parameter `PM=D` is set for a field, it is verified that the input data

- contains an even number of bytes,
- contains only valid DBCS characters,
- does not contain shift-out/shift-in characters (SO/SI).

Because the detection of non-DBCS characters requires ICU, this check will not be performed if ICU is not available (that is, if the profile parameter `CFICU=OFF` has been set).

Output Data Adjustment

If a window is to be displayed for user interaction, the window might overlay DBCS characters that are already displayed, or the window might itself contain DBCS characters which are truncated because of the window size. An overlay may also occur if the `NO ERASE` option is used with an `INPUT` statement. In order to prevent screen corruption in case of such an overlay, the following actions are performed to adjust the output data, if necessary:

- if the session parameter `PM=D` is set for a field, an orphan byte (that is, a single byte left at the beginning or end of the data to be displayed as a result of a partial overlay of a DBCS character) is replaced by an attribute; this operation assures that only valid DBCS characters are displayed;
- if the profile parameter `SOSI` has been set, the field contents of an alphanumeric field for which `PM=D` is not specified is examined for shift-out/shift-in characters (SO/SI); if a shift-out character (SO) is found for which the correlating shift-in character (SI) is missing, either the last character of the output data is replaced by a shift-in character (SI) or the last two characters are replaced by a shift-in character (SI) followed by a blank; if a shift-in character (SI) is found for which the correlating shift-out character (SO) is missing, either the first character of the output data is replaced by a shift-out character (SO) or the leading two characters are replaced by a blank followed by a shift-out character (SO); this operation assures that DBCS characters are enclosed properly by shift-out/shift-in characters (SO/SI).

Natural Stack Data

To avoid unintentional interpretation of DBCS characters as delimiter or control characters, the `FORMATTED` option of the `STACK` statement should be used if the data to be placed on the Natural stack contains DBCS characters.

See the *Statements* documentation for further information on the `STACK` statement.

See the *Programming Guide* for further information on the Natural Stack.

Application Programming Interfaces for DBCS Handling

The following user application programming interfaces (API) are available to support DBCS handling:

- [USR4211N - Get DBCS Characters](#)

- [USR4213N - String Handling for DBCS Support](#)

These APIs are contained as subprograms in the Natural library `SYSEXT`. Detailed information on how to use an API is included in the corresponding text object (`USRXXXXT`). See also *SYSEXT Utility - Natural Application Programming Interfaces* in the *Utilities* documentation.

USR4211N - Get DBCS Characters

The application programming interface `USR4211N` can be used to obtain information on the availability of DBCS support and the defined SOSI characters.

USR4213N - String Handling for DBCS Support

The application programming interface `USR4213N` can be used to perform the following functions:

- Convert a normal Latin character string into the corresponding DBCS character string.
- Convert a DBCS character string that contains Latin data only into a single-byte character string.
- Add the current shift codes at the beginning and at the end of a character string.
- Remove leading and trailing shift codes from a character string.

The last two functions can be used to either produce native DBCS strings or generate mixed-mode data out of native DBCS strings.

Alternate Text Module NATTXT2U

The alternate text module `NATTXT2U` contains certain keywords for English language in all upper case which are contained in mixed case in text module `NATTXT2`. `NATTXT2U` should be linked to the Natural nucleus instead of `NATTXT2` in environments where lower case code points H'81' to H'A9' are used to display national characters.

59 Asynchronous Processing

- Identifying Asynchronous Natural Sessions 412
- Handling Output of an Asynchronous Natural Session 412
- Handling Unexpected or Unwanted Input 413
- Other Profile Parameter Considerations 413

This document describes asynchronous Natural processing, a method which is available under all TP monitors supported by Natural.

An asynchronous Natural session is a session which is not associated with any terminal and therefore cannot interact with a terminal user. It can be used to execute a time-consuming task “in the background” without the user having to wait for the task to finish.

Related Topics:

- *Asynchronous Natural Processing under CICS*
- *Asynchronous Natural Processing under Complete/SMARTS*
- *Asynchronous Transaction Processing under openUTM*

Identifying Asynchronous Natural Sessions

To identify a session as being asynchronous, the Natural system variable `*DEVICE` is assigned the value `ASYNCH`.



Note: The value of `*DEVICE` may be modified by the Natural profile parameter `TTYTYPE` and by any `SET CONTROL 'T=xxxx'` statement; see also profile parameter `TTYTYPE` in the *Parameter Reference* documentation and terminal command `%T=` in the *Terminal Command* documentation.

Handling Output of an Asynchronous Natural Session

As an asynchronous session is a session that is not associated with any terminal, this means that any output produced by the session cannot simply be displayed on the screen; instead, you have to explicitly specify an output destination. You specify this destination with the Natural profile parameter `SENDER` when invoking Natural. The `SENDER` destination applies to hardcopy output and primary reports; any additional reports are sent to the destinations specified with the `DEFINE PRINTER` statement, just as in a synchronous online session.

As an asynchronous session can also cause a Natural error, the destination to which any Natural error message is to be sent must also be specified; this is done with the Natural profile parameter `OUTDEST`. This parameter also provides an option to have error messages sent to the operator console. After an error message has been sent, Natural terminates the asynchronous session.

The profile parameters `SENDER` and `OUTDEST` should be set accordingly to be prepared for unexpected output by the asynchronous Natural session; otherwise, the asynchronous Natural session may abend in such a scenario.

Handling Unexpected or Unwanted Input

An asynchronous Natural session only has the Natural stack to enter the name of Natural programs and Natural system commands to be executed. If a Natural program or a Natural system command fails with an unhandled Natural error or if the entire Natural stack is exhausted and NEXT mode would be entered, the asynchronous Natural session is terminated with termination message NAT9943.

Depending on the TP monitor in use and depending on the TTYPE setting, either the CLEAR key or the EOF indicator is passed back to Natural on an INPUT request by default. This measure helps to prevent error loop situations if a program unintentionally executes an INPUT statement. To pass the ENTER key indicator back, you can issue a SET CONTROL 'N' statement prior to the INPUT statement.

 **Tip:** You can make your application compatible with asynchronous sessions by evaluating the system variable *SCREEN-IO accordingly.

Other Profile Parameter Considerations

The following Natural profile parameters should be considered in the case of an asynchronous Natural session:

Profile Parameter	Comment
AUTO	Asynchronous sessions may have non-alphabetical user IDs. In this case, AUTO=ON will fail.
CM	An unwanted input situation may happen if the Natural session accidentally falls onto the NEXT level. Setting CM=OFF will terminate the session immediately in such a situation.
ENDMSG	The error message NAT9995 (normal termination message) can be suppressed by specifying ENDMSG=OFF.
IMSG	Natural initialization error messages and warnings can be suppressed by specifying IMSG=OFF.
MENU	Asynchronous sessions only have the Natural stack for command inputs; therefore, it is recommended to specify MENU=OFF and to navigate through Natural by using direct commands.
PLOG	Dynamic parameter logging is executed by sending all parameters line by line to the SENDER destination.
PROGRAM	If a standard back-end program/transaction is defined in your installation, it should be checked if this program can run asynchronously or if it is desired to deal with terminal-bound sessions only. Specifying PROGRAM=0 bypasses the back-end logic.

