

Natural for Mainframes

Operations

Version 4.2.6 for Mainframes (Update)

February 2010

This document applies to Natural Version 4.2.6 for Mainframes (Update).

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

Copyright © 1979-2010 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

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

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://documentation.softwareag.com/legal/> 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 and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Table of Contents

1 Operations	1
2 Configuring Natural	3
3 Linking Natural Objects to the Natural Nucleus	5
Benefits	6
ULDOBJ Utility	6
Using ULDOBJ to Generate an Object Module	7
Additional Considerations for Linking Subroutines	9
Operating System Dependency of Object Module Generation	9
Example of Linking a Natural Object to the Natural Nucleus	10
4 Natural User Exits	13
NATUEX1 - User Exit for Authorization Control	14
NATSREX2 and NATSREX3 - User Exits for Sort Processing	15
NATUSKnn - User Exit for Computation of Sort Keys	16
NATPM - User Exit for Inverted Output	17
NREXPG -User Exit for NATRJE	18
USR0070P - User Exit for Editor Profiles	19
USR2002P - User Exit for Help Window Text Strings	19
USR2003P - User Exit for Main Menu	19
5 Natural User Access Method for Print and Work Files	21
NATAMUSR Module Description	22
NATAMUSR Module Installation	22
Invoking the Third Party Product	22
6 Natural Scratch-Pad File	23
Purpose of a Natural Scratch-Pad File	24
How to Define a Scratch-Pad File	24
What is Stored on the Scratch-Pad File and How to Size it	25
Scratch-Pad File Maintenance	26
7 Natural Text Modules	27
Function and Usage of Text Modules	28
NATTEXT Module	28
NATTXT2 Module	29
NATTXT3 Module	32
8 Natural Configuration Tables	35
NATCONFIG Module	36
General Overview of Macros Used by NATCONFIG	37
NTDVCE - Terminal-Device Specification Table	38
NTMSG - Message Log Table Definitions	38
NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus	39
NTCPAGE - Code Page Definitions	39
Code Page Support	40
Output Devices Supported	41
Specification of NTDVCE	42
Translation Tables	42

Upper-/Lower-Case Translation	45
CMULT Entry	46
Output Translation	46
Input Translation	47
Code Translation of DBCS Data	47
NTTZ - Time Zone Definitions	48
9 Natural Storage Management	51
Thread and Non-thread Environments	52
Buffer Types	52
Fixed Buffers	53
Variable Buffers	53
Customization of Buffer Characteristics	54
10 Profile Parameter Usage	57
11 Natural Parameter Hierarchy	59
Natural Parameter Hierarchy Overview	60
General Rules for Parameter Usage	60
Natural Standard Parameter Module	61
Alternative Parameter Module	61
Predefined Dynamic Parameter Sets	62
Predefined User Parameter Profiles	62
Dynamic Parameter Entry	62
Natural Security Definitions	63
Session Parameter Settings	63
Program/Statement Level Settings	63
Development Environment Settings	64
Examples of Various Parameter Strings	64
12 Assignment of Parameter Values	67
Sources for Parameter Value Assignment	68
Static Assignment of Parameter Values	69
Dynamic Assignment of Parameter Values	70
Session Parameters for Runtime Assignment of Parameter Values	72
13 Profile Parameters Grouped by Function	73
System Files	74
Buffer Sizes	74
External Subprograms	75
Output Reports and Work Files	75
Date/Time Settings	76
Limits	77
Character Assignments	77
Terminal Communication	78
Buffer Pools	78
Translation Tables	79
Code Page and Unicode Support	79
Usage of Profile Parameters	80
Compiler Options	80

Debugging	80
Batch Mode	81
TP Monitors	81
Database Access	82
Natural with Adabas	82
Natural with Other Software AG Products	83
Miscellaneous Profile Parameters	85
Session Initialization and Termination	86
Parameters Reserved for Internal Use	87
14 Using a Natural Parameter Module	89
Using the Default Natural Parameter Module NATPARM	90
Creating a New Natural Parameter Module	90
NTPRM Macro - Create a Natural Parameter Module	90
Restricting the Use of a Parameter Module	91
Using Macros in a Natural Parameter Module	92
15 z/OS Environment	95
16 Natural under z/OS	97
Natural Subsystem	98
Shared Nucleus	98
TP Monitor Interfaces	98
Interfaces to Database Management Systems	99
Natural in Batch Mode under z/OS	99
Natural as a Server under z/OS	99
17 Authorized Services Manager under z/OS	101
ASM Overview	102
ASM System Requirements	103
ASM Operation	104
18 Natural Shared Nucleus under z/OS and z/VSE	109
Environment-Independent Nucleus	110
Creating a Shared Nucleus	113
Installing a Shared Nucleus	114
Linking Subproducts to the Nucleus	114
Single-Environment Shared Nucleus	115
Environment-Dependent Nucleus	116
Statically Linked Non-Natural Programs	116
Dynamically Called Non-Natural Programs	117
19 Natural Roll Server Functionality	119
Natural Roll-Server Overview	120
Roll Server in a Single z/OS System	120
Roll Server in a z/OS Parallel Sysplex Environment	122
Roll File and LRB	124
20 Natural Roll Server Operation	127
Roll Server System Requirements	128
Formatting the Roll File	129
Starting the Roll Server	133

Roll Server Messages, Condition Codes and Abend Codes	136
Return Codes and Reason Codes of the Roll Server Request	137
Operating the Roll Server	137
Roll Server Performance Tuning	138
Roll Server User Exits	139
21 z/VSE Environment	143
22 Natural under z/VSE	145
Natural Subsystem	146
Natural Shared Nucleus	146
TP Monitor Interfaces	146
Interfaces to Database Management Systems	146
Natural in Batch Mode under z/VSE	147
23 Natural Shared Nucleus under z/OS and z/VSE	149
24 VM/CMS Environment	151
25 Natural under VM/CMS	153
Issuing CP and CMS Commands from Natural	154
Reading the CMS Program Stack	154
Hardcopy Function	155
Applying Fixes to Natural	155
Natural in Batch Mode under CMS	155
Using TCP/IP Communication	155
Calling Natural Subprograms from Rexx	156
26 Print File and Work File Support	157
Defining Print Files and Work Files	158
Access Method STD	158
Access Method CMS	158
27 BS2000/OSD Environment	161
Related Topics	162
Other Natural Functions for BS2000/OSD-Specific Purposes	163
28 Natural Shared Nucleus under BS2000/OSD	165
Rules for Using a Natural Shared Nucleus	166
29 Refresh of Natural Load Pool	169
Prerequisites/Restrictions	170
Procedure	170
Keyword Parameters for the Program PREFRESH	171
30 Optimization of Message Handling	175
Screen Output Handling	176
Restoring the Screen Content	176
31 Siemens Terminal Types Supported by Natural	177
Type 9748	178
975n Series	178
Type 9763M	179
32 Function Key Support with 9750 Devices	181
Key Assignment	182
Modes for Key Assignment	182

33 Common Memory Pools	185
Global Common Memory Pools	186
Local Common Memory Pools	190
34 Calling Dynamically Reloadable 3GL Programs in a Natural Application	197
Storage Allocation Rule	198
Thread-Creation Rule	198
Address-Mode Dependencies	198
35 Print File/Work File Server NATPWSV2	201
Setup	202
Operation	203
36 RPC Server Front-End	205
Setup	206
37 Natural in Batch Mode	211
38 Natural in Batch Mode under z/OS	213
General Information about the Natural z/OS Batch Mode Interface	214
Natural z/OS Generation Parameters	214
Datasets Used by Natural in z/OS Batch Mode	217
39 Natural in Batch Mode under z/VSE	223
NATVSE - Natural z/VSE Batch Mode Interface	224
NTVSE Macro - Generation Parameters for Natural under z/VSE	224
Natural Datasets Used under a z/VSE Batch Mode Session	230
NATVSE Print and Work File Support for z/VSE Library Members	235
NATVSE Dynamic Work File Allocation (DYNALLOC) Support	237
Debugging Facilities for Natural under z/VSE	240
NATVSE Attention Interrupts	244
40 Natural in Batch Mode under CMS	245
Running Natural in Batch Mode under CMS	246
41 Natural in Batch Mode under BS2000/OSD	247
Files and System Files Used by Natural in BS2000/OSD Batch Mode	248
Keyword Parameters	250
BS2000/OSD Job Variables	259
42 Natural in Batch Mode (All Environments)	261
Adabas Datasets	262
Sort Datasets	262
Subtasking Session Support for Batch Mode Environments	262
43 Natural Buffer Pools	267
44 Natural Buffer Pool - General	269
Natural Buffer Pool Principle of Operation	270
Buffer-Pool Monitoring and Maintenance	275
Natural Global Buffer Pool	278
45 Natural Global Buffer Pool under z/OS	281
Using a Natural Global Buffer Pool	282
Operating the Natural Global Buffer Pool	282
Sample NATGBPvr Execution Jobs	284
Localization	286

46	Natural Global Buffer Pool under z/VSE	287
	Using a Natural Global Buffer Pool	288
	Operating the Natural Global Buffer Pool	289
	Sample NATGBPvr Execution Jobs	290
	Localization	292
47	Common Natural GBP Operating Functions under z/OS and z/VSE	293
	Global Buffer Pool Manager Parameter Module	294
	Global Buffer Pool Operating Functions	294
	Global Buffer Pool Function Parameters	296
	Examples of NATBUFFER Specifications	301
48	Natural Global Buffer Pool under BS2000/OSD	303
	Using a Natural Global Buffer Pool under BS2000/OSD	304
	Establishing the Global Buffer Pool under BS2000/OSD	304
	Administering the Global Buffer Pool under BS2000/OSD	305
49	Natural Swap Pool	307
50	Purpose of a Natural Swap Pool	309
	Purpose of a Natural Swap Pool	310
	Benefits of Using a Natural Swap Pool	310
	Swap Pool Structure	311
51	Natural Swap Pool Operation	313
	Users are On their Way to Natural - No Session Start	314
	Users are Returning from Natural	314
52	Natural Swap Pool Initialization	317
	Swap Pool Initialization Control	318
	Swap Pool Initialization Parameters	319
53	Dynamic Swap-Pool Reorganization	321
	Requirements for Dynamic Swap-Pool Reorganization	322
	Statistics Tables	322
	Swap-Pool-Reorganization Plus Table	322
	Swap-Pool-Reorganization Minus Table	323
	Parameters for Swap-Pool Reorganization	323
	Checking for the Necessity of Swap-Pool Reorganization	324
	Flow of Dynamic Swap-Pool Reorganization	324
	Start of Dynamic Swap-Pool Reorganization	325
54	Defining the Natural Swap Pool	327
	Environment-Specific Requirements	328
	Keyword Parameters of Macro NTSWPRM	328
55	Natural User Area Size Considerations	335
	Using the MAXSIZE Parameter	336
	Defining the Size of the Individual Natural Buffers	336
	Possible Error Messages	336
	Displaying the Aggregate Size of All Buffers	337
	Calculating the Maximum Size	337
56	Swap Pool Data Space	339
	Using ESA Data Space in Addition	340

ESA Data Space Slot Size Adjustment	340
57 Global Restartable Swap Pool under UTM	341
Purpose of a Natural Global Swap Pool under UTM	342
Installing a Natural Global Swap Pool under UTM	342
Starting a Natural Global Swap Pool under UTM	343
Displaying Information about the Global Swap Pool	343
58 Terminating the Global Swap Pool under UTM	345
Termination Using Console Commands	346
Abnormal Termination with Dump	346
Termination by Program	347
59 Natural 3GL CALLNAT Interface	349
60 Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions	351
Purpose of 3GL CALLNAT Interface	352
Prerequisites	353
Restrictions	354
61 Natural 3GL CALLNAT Interface - Usage, Examples	357
Usage	358
Sample Environments	361
62 Operating the Software AG Editor	365
63 Editor Work File	367
Editor Work File Structure	368
Editor Work File under z/OS, z/VSE and BS2000/OSD	369
Using the Batch Format Utility	370
Formatting during Initialization	370
Maintaining the Editor Work File under z/OS and z/VSE	370
Maintaining the Editor Work File under BS2000/OSD	371
Editor Work File under VM/CMS	372
Editor Work File under Complete/SMARTS	372
64 Editor Buffer Pool	373
Purpose of the Editor Buffer Pool	374
Obtaining Free Blocks	375
Initializing the Editor Buffer Pool	375
Restarting the Editor Buffer Pool	376
Editor Buffer Pool Parameters	376
Buffer Pool Initialization for Multi-User Environments	376
65 Natural Net Data Interface NATNETTO	379
Natural Net Data Driver Functional Description	380
General Message Layout	381
Layout of Header	381
Format Buffer Layout	385
Value Buffer Layout	390
Attribute Buffer	391
66 Natural as a Server	393
67 Natural as a Server under z/OS	395
Functionality	396

Natural Nucleus Installation in a Server Environment	397
Print and Work File Handling with External Datasets in a Server Environment	397
68 Natural as a Server under z/VSE	399
Functionality	400
Natural Nucleus Installation in a Server Environment	401
Print and Work File Handling with External Datasets in a Server Environment	401
69 Natural as a Server under CICS	403
Functionality	404
Natural CICS Interface Installation in a Server Environment	404
Restrictions	405
70 Natural Execution - Miscellaneous Topics	407
71 Asynchronous Processing	409
Identifying Asynchronous Natural Sessions	410
Handling Output of an Asynchronous Natural Session	410
Handling Unexpected or Unwanted Input	411
Other Profile Parameter Considerations	411
72 Double-Byte Character Sets	413
Natural Profile Parameter SOSI	414
Output Format Specification	414
Parameter Definitions for DBCS Support	414
Editor Profile Options	415
Input Data Check	415
Output Data Adjustment	416
Natural Stack Data	416
Application Programming Interfaces for DBCS Handling	416
73 Input/Output Devices	419
Terminal Support	420
Light Pen Support	420
Printer Support	422
74 Back-End Program Calling Conventions	425
Back-End Program Calling Conventions (Batch Mode)	426
Special Considerations under CICS	427
Special Considerations under IMS TM	427
Sample Back-End Programs	427
75 Natural 31-Bit Mode Support	429
76 LE Subprograms	431
Support of IBM LE Subprograms	432
Enabling Natural Support of LE Subprograms	432
Passing LE Runtime Options	432
LE Abend Handling	434
77 External SORT	435
Support of External SORT	436
Special Considerations for z/OS	436

Special Considerations for z/VSE	436
Special Considerations for BS2000/OSD	437
Index	439

1 Operations

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 scratch-pad file, 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, provides an overview of the profile parameters available (grouped by function). Describes how to use 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). Describes the function and the use of the Shared Natural nucleus. Explains the functions of the Natural Roll Server. Provides information on the Roll Server system requirements, operation, performance tuning and restartability.
	z/VSE Environment	Contains special considerations that apply when you are running Natural under z/VSE online or in batch mode. Explains the function and the use of the Shared Natural nucleus.
	VM/CMS Environment	Explains topics such as issuing CP and CMS commands from Natural, reading the CMS program stack, hardcopy function and applying fixes to Natural. In addition, links are available to topics that apply when you are using Natural under CMS in batch mode. Provides information on how to define print files and work files in the Natural parameter module.
	BS2000/OSD Environment	Contains special considerations that apply when running Natural under the operating system BS2000/OSD.

	Natural in Batch Mode	Contains considerations that apply when running Natural in batch (Adabas datasets, sort datasets, subtasking session support for batch environments), and specifically when running Natural in batch mode under z/OS, z/VSE, VM/CMS and BS2000/OSD.
	Natural Buffer Pools	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/OSD.
	Natural Swap Pool	Provides information on the Natural swap pool which is available when you are using the TP monitor CICS or UTM.
	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.
	Natural as a Server	Describes the use of Natural as a Server and the Natural Server Monitor.
	Natural Execution - Miscellaneous Topics	Provides general information on Natural execution (asynchronous processing, double-byte character sets, input/output devices, back-end program calling conventions, Natural 31-bit mode support, LE subprograms, and external SORT).

Related Documents:

- Installation
- Messages and Codes
- Natural TP Monitor Interfaces
- Natural Remote Procedure Call
- Natural Utilities
- Software AG Editor
- Natural Security
- Natural for VSAM
- Natural for DB2
- Natural for DL/I
- Natural for SQL/DS

2

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 Scratch-Pad File](#)
- [Natural Text Modules](#)
- [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*

3 Linking Natural Objects to the Natural Nucleus

- Benefits 6
- ULDOBJ Utility 6
- Using ULDOBJ to Generate an Object Module 7
- Additional Considerations for Linking Subroutines 9
- Operating System Dependency of Object Module Generation 9
- Example of Linking a Natural Object to the Natural Nucleus 10

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.

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

The following topics are covered:

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 a [Natural shared nucleus](#) is used, the generated object module has to be linked to the environment-*independent* 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.

```

10:12:19          ***** NATURAL OBJECT MAINTENANCE *****          2005-01-05
User: XYZ          - NATURAL ULDOBJ UTILITY -                          Library: SYSMISC
                                                           Opsys .. z/OS

Specify parameters below ....

Object ..... _____ (Enter '.' to exit)
Library ..... SYSMISC_
OP System ... _____
    
```

- 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/OSD systems CMS VM/CMS systems

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

- the object type (Program, Subprogram, Subroutine, Helproutine, Map, Adapter);
- 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 and system maintenance (SM) level of the cataloged object;
- 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/OSD	<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 data-bbox="305 380 682 478"> ADDR >TEST object module ... END </pre> <p>When the LMS utility is executed, assign SYSDTA to the work file written by the ULDOBJ utility (SYSDTA=work-file-1). The file name generated is Nvr.MOD, where vr stands for the current Natural version and release number.</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 dataset.

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.V2.USER.LOAD,DISP=SHR/**
//SYSLIN DD*
...
...          INCLUDE MODULES FOR NUCLEUS
...
INCLUDE NATLIB(NATPARM)    NATPARM MODULE
INCLUDE SYSLMOD(LOGPROG)  LOGON NATURAL PGM
INCLUDE SYSLMOD(EDITPROG) EDITOR NATURAL PGM
...
...          INCLUDE ENTRY AND NAME CARDS
...
/**
```

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.

4 Natural User Exits

▪ NATUEX1 - User Exit for Authorization Control	14
▪ NATSREX2 and NATSREX3 - User Exits for Sort Processing	15
▪ NATUSKnn - User Exit for Computation of Sort Keys	16
▪ NATPM - User Exit for Inverted Output	17
▪ NREXPG -User Exit for NATRJE	18
▪ USR0070P - User Exit for Editor Profiles	19
▪ USR2002P - User Exit for Help Window Text Strings	19
▪ USR2003P - User Exit for Main Menu	19

A Natural user exit is a programming object that is invoked by Natural, a subcomponent or a subproduct. Usually, a sample user exit is delivered in source form. The instructions contained in the user exit have to be written or adjusted by the user. The purpose of a user exit is to manipulate data or make decisions. Most user exits take advantage of the Natural programming language; a small subset has to be written in Assembler language.

This document describes the following Natural user exits:

Other Natural user exits and application programming interfaces are described in the relevant places in the Natural subcomponent or subproduct documentation (*Natural RPC, TP Monitor Interfaces, Utilities, add-on products, etc.*).

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 a shared nucleus or to an environment-independent nucleus. It is also possible to link it to an alternative 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/OSD*.

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 <i>NATSREX2</i> and <i>NATSREX3</i>
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 `NATUSKnn` will be invoked - `nn` being the current language code (that is, the current value of the system variable `*LANGUAGE`).

You can write a `NATUSKnn` 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 corresponding "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 `NATCONFG`; this means that `NTUTAB1` may have to be adjusted accordingly.

`NATUSKnn` is called using standard calling conventions:

Register	Contents
15	Entry address of <code>NATUSKnn</code>
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 <code>NTUTAB1</code> .

`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.

When a shared nucleus is used, `NATUSKnn` can be linked to the environment-independent part of the nucleus.

It is also possible to link it to an alternative parameter module, or as a separate module if you supply the name(s) of the invoked `NATUSKnn` module(s) with the profile parameter `RCA`.

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 member 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 member 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 member USR2003T.

5

Natural User Access Method for Print and Work Files

- NATAMUSR Module Description 22
- NATAMUSR Module Installation 22
- Invoking the Third Party Product 22

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.

The following topics are covered:

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 Natural nucleus,
- linked to the Natural driver (when driver and front-end are split),
- 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.

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 `NTWORK`).

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.

6 Natural Scratch-Pad File

- Purpose of a Natural Scratch-Pad File 24
- How to Define a Scratch-Pad File 24
- What is Stored on the Scratch-Pad File and How to Size it 25
- Scratch-Pad File Maintenance 26

This document provides information on purpose, use and maintenance of a Natural scratch-pad file.

The following topics are covered:

Purpose of a Natural Scratch-Pad File

What is it, what does it do?

The scratch-pad file is just another Natural system file like `FNAT` and `FUSER`, and has the same physical file layout. It enables the storage of, for example, saved screen images and other types, data which are not stored explicitly like Natural sources, objects (`SAVE`, `CATALOG`, `STOW`) and error messages, on a file other than the system file `FNAT` or `FUSER`.

When do I need it?

In contrast to `FNAT` and `FUSER`, a scratch-pad file is *not mandatory* in a Natural session.

However, if you are working with read-only access to system files (profile parameter `ROSY=ON`), you *must* define a scratch-pad file, because otherwise the above mentioned data could not be stored and a corresponding error message (NAT0106) would be issued instead. The scratch-pad file is excluded from read-only access.

How to Define a Scratch-Pad File

Like all other system files of Software AG products, the scratch-pad file is a logical file. The logical file number of the scratch-pad file is 212.

Since there is no mnemonic for the scratch-pad file such as `FNAT` and `FUSER` or `FDIC`, it has to be defined:

- either statically by using the macro `NTLFILE` in the Natural parameter module `NATPARM` or
- dynamically by using the profile parameter `LFILE`.

Examples of NTLFILE and LFILE definition:

LFILE Parameter:

```
LFILE=(212,physical-dbid,physical-fnr,password,cipher-key)
```

NTLFILE Macro:

```
NTLFILE 212,physical-dbid,physical-fnr,password,cipher-key
```

What is Stored on the Scratch-Pad File and How to Size it

The objects that are stored on the scratch-pad file are:

- **Recordings**
- **Screen Captures** (NATPAGE utility)

As the amount of usage of the Recording Utility and the NATPAGE utility cannot be calculated beforehand, a reasonable estimate about the related storage requirements is hardly possible. 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.

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 paging 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 %O 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.

Scratch-Pad 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 SYSMAIN.
- 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.

7 Natural Text Modules

- Function and Usage of Text Modules 28
- NATTEXT Module 28
- NATTXT2 Module 29
- NATTXT3 Module 32

This document describes the Natural text modules NATTEXT, NATTXT2 and NATTXT3. It covers the following topics:

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 Natural *Installation* documentation.

NATTEXT Module

The NATTEXT module contains NTKEY and NTALT macros 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 Module

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

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

- [Natural Termination Messages and Return Codes](#)

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*.

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 it 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 and if supplied in source code, the environment dependant driver have 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 to abend or severe environment failure..

User-written termination messages can be added to `NATXT2` for all return codes (1 - 255) which can be issued with a `TERMINATE` statement and which normally lead to the Natural termination message `NAT9987`.

For user-written termination messages, the corresponding return code must be specified as the second parameter.

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.

Example of a User Termination Message:

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

NATTXT3 Module

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,programma    4  SPANISH
      SPACE
```

*-----
MSGGEN

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

8 Natural Configuration Tables

▪ NATCONFIG Module	36
▪ General Overview of Macros Used by NATCONFIG	37
▪ NTDVCE - Terminal-Device Specification Table	38
▪ NTMSG - Message Log Table Definitions	38
▪ NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus	39
▪ NTCPAGE - Code Page Definitions	39
▪ Code Page Support	40
▪ Output Devices Supported	41
▪ Specification of NTDVCE	42
▪ Translation Tables	42
▪ Upper-/Lower-Case Translation	45
▪ CMULT Entry	46
▪ Output Translation	46
▪ Input Translation	47
▪ Code Translation of DBCS Data	47
▪ NTTZ - Time Zone Definitions	48

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

The following topics are covered:

See also:

- *Input/Output Devices Supported*

NATCONFIG Module

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 NATPARAM 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 system maintenance (SM) 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. 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.
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 and upper case. 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.
NLANG *	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, upper-case alphabetical, numeric and special characters (applies to dynamic profile parameters, MASK and SCAN options).
NTTZ	Time zone definitions. The NTTZ macro enables specifications about zonetime 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, the telex driver module `NATTLX` is provided for Connect in order to provide faster telex, telefax and teletex communication from and to the `TOPCALL` system. `NATTLX` 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 `NATPARAM` 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
NAT $nnnn$	$nnnn$ is the Natural message number (mandatory).
$logid$	Indicates the log destination, that is, the operator console or job message log or both. Possible values: WTO, WTL or WTO+WTL

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 [ULDOBJ](#) 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 } ,ALIAS=value,PHC=value
```

NTCPAGE Macro Parameters

Parameter	Description
IANA	The standard name of the code page. Maximum length: 64 characters. This parameter is mandatory.
CCSID	Coded Character Set IDentification (IBM). A numeric value with up to 5 digits. Examples: 1141 German EBCDIC code page 62243 Hebrew/Latin (ISO 8859) code page
CCSN	Coded Character Set Name (Siemens BS2000/OSD). An alphanumeric string of up to 8 characters. Examples: EDF041 Latin code page for Western Europe EDF045 Latin/Cyrillic code page
ALIAS	Alias code page name. Maximum length: 32 characters. This parameter is optional, not unique.
PHC	Place Holder Character. Length: 2 byte hexadecimal. This parameter is optional



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

Example:

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

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.

Specification of NTDVCE

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

<p>Example of NTDVCE macro:</p> <pre>NTDVCE TYP=EBS2,NAME=BS2CHAR,ENTRY=VC3270,WXTRN=OFF,RTAL=5, FLAG1=CM3270,TCIO=(X'CO',X'FB',X'6A',X'4F',X'DO',X'FD', X'4A',X'BB',X'E0',X'BC',X'5A',X'BD',X'A1',X'FF',X'4F', X'5A')</pre> <p>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.</p>

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>
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 placeholder character, followed by a complete list of character mappings.</p>
NTSCTAB	<p>The SCAN/MASK character table which defines the properties of each printable character for the Natural mask definition function.</p> <p>This table can be used to define upper-case attributes, lower-case attributes, special characters, hexadecimal characters and numeric characters.</p> <p>It can be modified by the user and the result can be used directly in the Natural MASK clause.</p> <p>To modify this table, use the macro NTSCTAB in the Natural parameter module or the corresponding dynamic profile parameter SCTAB.</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
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 into question marks.</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. Then, although Natural is running with a code page, the translation using this table continues in order to avoid invalid, unprintable characters from the resulting output data.</p>
NTTAB1	<p>The alternative (secondary) output translation table for the secondary character set used when the Natural parameter PM is set to C.</p> <p>The important aspect is the translation of all possible terminal-control characters. If PM=C is specified, all Natural output data are translated with NTTAB1. A possible application of NTTAB1 is to avoid the translation of escape sequences for printer control.</p> <p>To modify this table, use the macro NTTAB1 in the Natural parameter module or the corresponding dynamic profile parameter TAB1.</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>
NTTAB2	<p>The secondary input translation table used when the Natural parameter PM is set to "C". If PM=C is specified, all Natural input data are translated with NTTAB2. Conversion between different languages or code pages can be performed with this table together with NTTAB1.</p> <p>To modify this table, use the macro NTTAB2 in the Natural parameter module or the corresponding dynamic profile parameter TAB2.</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>
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 CP (CP=ON, CP=AUTO or CP=<i>code-page</i>), the table is adjusted by ICU according to the code page used at session start.</p>

Macro	Table
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>
NTUTAB2	<p>The sample user-specific translation table which performs the translation specified with the statement <code>EXAMINE TRANSLATE INTO LOWER CASE</code>.</p> <p>To modify this table, you can use the macro <code>NTUTAB2</code> in the Natural parameter module or the corresponding profile parameter <code>UTAB2</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>
NTLANG	<p>The language-code table, which defines which language number is assigned to which language code in the system variable <code>*LANGUAGE</code>.</p>
NTTABL	<p>The <code>SYS*</code> output translation table, which is controlled by the Natural profile parameter <code>TS</code>. With <code>TS=ON</code>, this table is used to translate output produced by programs located in Natural <code>SYS*</code> 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 <code>NTTABL</code> in the Natural parameter module or the corresponding dynamic profile parameter <code>TABL</code>.</p> <p>If Natural is running with an MBCS code page (for example, <code>CP='IBM-939'</code>), 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	<p>Upper-/lower-case translation can also be performed with the EXAMINE TRANSLATE statement.</p> <p>By default, EXAMINE TRANSLATE translates to upper case by using the translation table NTUTAB1, and to lower case by using the translation table NTUTAB2.</p>

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 following topics are covered below:

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

NTTZ Macro

The NTTZ macro enables specifications about zonetime and automatic switching to and from summertime.

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 timezone. 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 shipped with NATCONFIG.

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 for automatic switching to and from summertime is currently for years in the range from 2002 to 2041.



Notes:

1. 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 concurrent use of `DD` or `YD` with `TD=zonename`.
2. Concurrent use of `TD=zonename` 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=zonename`.

NTTZ Macro Syntax

The syntax of the NTTZ macro is as follows:

```
NTTZ ZONE=time-zone-name, TDON=+/-hh:mm:ss,
      [TDOFF=+/-hh:mm:ss, SWTON=hh:mm:ss,
      SWTOFF=hh:mm:ss,
      DSTON=( [{FIRST | SECOND | THIRD | FOURTH | LAST},
               {MONDAY | ... | SUNDAY},
               {AFTER | BEFORE | IN}],
               {JANUARY | ... | DECEMBER},
               [, day-number]),
      DSTOFF=( [{FIRST | SECOND | THIRD | FOURTH | LAST},
                {MONDAY | ... | SUNDAY},
                {AFTER | BEFORE | IN}],
                {JANUARY | ... | DECEMBER}
                [, day-number]) ] ]
```

NTTZ Macro Parameters

NTTZ Macro Parameter	Description
<i><time +/- hh:mm:ss></i>	The basic format is <i><{+/-} hh:mm:ss></i> ranging from 00:00:00 through 23:59:59; abbreviations are also allowed, like: <i><hh:mm></i> or simply <i><hh></i> . The plus sign (+) is assumed by default, the minus sign (-) may be necessary with the parameters TDON or TDOFF (see below).
<i>time-zone-name</i>	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. 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	Denotes the difference of local daylight saving time (summertime) to UTC time (formerly GMT). This parameter corresponds to the parameter SWTON. If only the TDON parameter is defined, the user gets display of local time as his zone time, without automatic switching to and from summertime.
TDOFF	Denotes the difference of local zone time to UTC time (formerly GMT). This parameter corresponds to the parameter SWTOFF.
SWTON	Denotes the UTC point of time when daylight saving time (summertime) is switched on.
SWTOFF	Denotes the UTC point of time when daylight saving time is switched off.
DSTON	Denotes the day when daylight saving time (summertime) is switched on.
DSTOFF	Denotes the day when daylight saving time (summertime) is switched off.
<i>day-number</i>	A valid day number for the respective month; the default for <i>day-number</i> being 1.

Restrictions of NTTZ Macro

- LAST requires BEFORE or IN.
- If IN is specified, no *day number* must be specified.



Note: 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 specifications in local time.

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](#).

9 Natural Storage Management

▪ Thread and Non-thread Environments	52
▪ Buffer Types	52
▪ Fixed Buffers	53
▪ Variable Buffers	53
▪ Customization of Buffer Characteristics	54

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

The following topics are covered:

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 (`NATPARM`) 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.

<code>DATSIZE</code>	Data areas
<code>GLBTOOL</code>	Utility GDA
<code>GLBUSER</code>	User GDA
<code>GLBSYS</code>	System GDA
<code>AIVDAT</code>	AIV area
<code>CONTEXT</code>	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 [Buffer Sizes](#).

10 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** Provides an 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** Explains how values can be assigned to profile parameters statically, dynamically and at runtime.
- **Profile Parameters Grouped by Function** Provides an overview of the profile parameters available, grouped by function.
- **Using a Natural Parameter Module** Covers the following topics: assembling a Natural parameter module, using the NATPARM default Natural parameter module, creating the NTPRM macro, optional macros used in a Natural parameter module.

For details of the individual profile parameters, see *Profile Parameters* documentation.

11 Natural Parameter Hierarchy

▪ Natural Parameter Hierarchy Overview	60
▪ General Rules for Parameter Usage	60
▪ Natural Standard Parameter Module	61
▪ Alternative Parameter Module	61
▪ Predefined Dynamic Parameter Sets	62
▪ Predefined User Parameter Profiles	62
▪ Dynamic Parameter Entry	62
▪ Natural Security Definitions	63
▪ Session Parameter Settings	63
▪ Program/Statement Level Settings	63
▪ Development Environment Settings	64
▪ Examples of Various Parameter Strings	64

This document describes the hierarchical structure of the different levels on which Natural profile parameters can be set. Various examples are given to illustrate the scenario.

The following topics are covered:

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

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
Static	<ul style="list-style-type: none"> ■ Alternative Parameter Module ■ Natural Standard 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 Standard Parameter Module

Natural parameters are defined in the standard (default) parameter module which is linked to the Natural nucleus. This module constitutes the bottom level of the Natural parameter hierarchy.

Special Case:

If a shared Natural nucleus is used, this parameter module must be linked to the environment-dependent nucleus module. This parameter module then constitutes the *second hierarchical level* and overwrites *all* the parameters of the parameter module which is linked to the shared nucleus (if any). Exception: the `CSTATIC` subprograms of the shared nucleus, see [Statically Linked Non-Natural Programs](#).

Alternative Parameter Module

In addition to the Natural standard parameter module, the Natural administrator can define any number of additional (alternative) parameter modules. Such a module is stored in a TP or operating-system library and can be used as alternative parameter module by the parameter `PARM` when Natural is started.

These parameters cause the parameters of the standard parameter module to be completely overwritten.

Exception: `CSTATIC` entries, see [Statically Linked Non-Natural Programs](#).



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

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

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 `SYSPARM` 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 `SYSPARM` 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 `DYNPARM` (only dynamically, for instance in a profile).

You can use the macro `NTDYNP` in the parameter module `NATPARM` to make analog settings. This, however, will prohibit the use of the profile parameter `DYNPARM`.

You can use the file `CMPRMIN` to define dynamic parameters in batch mode under z/OS, BS2000/OSD and z/VSE, or in batch-like systems such as TSO, TIAM, CMS 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 Parameter Settings

The Natural system command `GLOBALS` or, in Reporting Mode, 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 Various Parameter Strings

The examples below are based on the following parameter settings:

Parameter	Param. Module, Shared Nucleus (special case)	Param. Module Front-End	Alternative Param. Module ALTPARM	User Profile MYPROF
DATSIZE	32 (default)	40	50	60
DSIZE	4	6	2 (default)	Not specified
ESIZE	20	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	Front-end module
DSIZE 6	Front-end module
ESIZE 28	Front-end module
Others: Default	Front-end module

Example 2: PARM=ALTPARM

Resulting Values	Origin
DATSIZE 50	ALTPARM
Others: Default	ALTPARM

Example 3: SYS=A

Resulting Values	Origin
DATSIZE 40	Front-end module
DSIZE 6	Front-end module
ESIZE 40	NTSYS front-end module

Example 4: PARM=ALTPARM, SYS=A

Resulting Values	Origin
DATSIZE 50	ALTPARM
DSIZE 2	ALTPARM
ESIZE 60	NTSYS, ALTPARM

Example 5: PARM=ALTPARM, SYS=B

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

Example 6: SYS=A, PROFILE=MYPROF

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Front-end module
ESIZE 80	MYPROF

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

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Front-end module
ESIZE 100	Dynamic parameter

Example 8: PROFILE=MYPROF, SYS=A

Resulting Values	Origin
DATSIZE 60	MYPROF
DSIZE 6	Front-end module
ESIZE 40	NTSYS front-end module

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

Resulting Values	Origin
DATSIZE 50	ALTPARM
Others Default	ALTPARM

12 Assignment of Parameter Values

- Sources for Parameter Value Assignment 68
- Static Assignment of Parameter Values 69
- Dynamic Assignment of Parameter Values 70
- Session Parameters for Runtime Assignment of Parameter Values 72

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

The following topics are covered:

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 by the macro `NTPRM` and other macros in the Natural parameter source module (`NATPARM`). These macros are then assembled and linked with the Natural nucleus. All parameters not specified are assigned their default values.

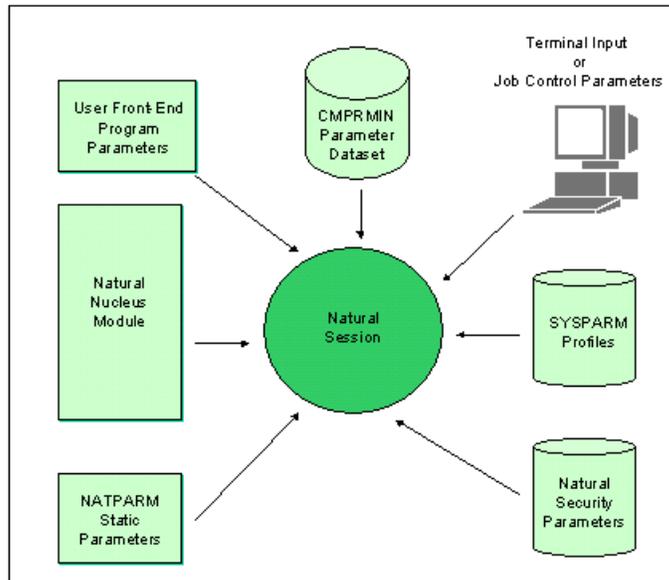
2. Dynamic assignments

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 dataset (`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, in Reporting mode) 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 `NATPARM` is used for the static assignment of profile parameters for all Natural environments.

In the 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 parameter module can be overwritten dynamically at the start of a Natural session.

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

Static:	<code>MACRO-NAME KEYWORD1=value1,KEYWORD2=value2,...</code>
Dynamic:	<code>PARAMETER-NAME=(KEYWORD1=value1,KEYWORD2=value2,...)</code>

Example:

- Macro in the parameter module: `NTSORT WRKSIZE=500,EXT=0N`
- Equivalent dynamic profile parameters: `SORT=(WRKSIZE=500,EXT=0N)`

If there is a corresponding macro for a profile parameter, this is indicated in the parameter description.

For more details on static assignment, see [Using a Natural Parameter Module](#).

Some Natural subproducts (for example, Natural for DL/I or Natural for DB2) use additional parameter modules. These modules are described in the documentation of these subproducts.

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 `NATPARM`.

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 dataset `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
```

In reporting mode, 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.

13 Profile Parameters Grouped by Function

▪ System Files	74
▪ Buffer Sizes	74
▪ External Subprograms	75
▪ Output Reports and Work Files	75
▪ Date/Time Settings	76
▪ Limits	77
▪ Character Assignments	77
▪ Terminal Communication	78
▪ Buffer Pools	78
▪ Translation Tables	79
▪ Code Page and Unicode Support	79
▪ Usage of Profile Parameters	80
▪ Compiler Options	80
▪ Debugging	80
▪ Batch Mode	81
▪ TP Monitors	81
▪ Database Access	82
▪ Natural with Adabas	82
▪ Natural with Other Software AG Products	83
▪ Miscellaneous Profile Parameters	85
▪ Session Initialization and Termination	86
▪ Parameters Reserved for Internal Use	87

To assist you as a Natural administrator in determining which parameters are applicable for your site, this document provides an overview of the profile parameters that are available to you. The parameters are grouped according to their functions:

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

System Files

Natural system files are used for the storage of various data and programs. See *Natural System Files* in the *Natural System Architecture* documentation.

The following profile parameters apply to all system files:

Parameter	Short Description
DBID	Default Database ID of Natural system files
FNR	Default File Number of Natural system files
SYSPSW	Default Password for Natural system files
SYSCIP	Default Cipher Key for Natural system files
ROSY	Read-only access to system files (FNAT, FUSER and FSEC only)

With the following parameters, you can override the default values for individual system files:

Parameter	Short Description
FNAT	Natural system file for system programs
FUSER	Natural system file for user programs
FDIC	Predict system file
FSEC	Natural Security system file
FSPool	Natural Advanced Facilities spool file

Buffer Sizes

Natural uses several buffer areas to store programs and data. You may need to adjust the size of one or more of these areas in order to achieve maximum buffer efficiency. If the specified space is not available, the size of the requested buffer is set to zero.

Parameter	Short Description
DATSIZE	Size of buffer for local data
DS	Size of storage buffer
DSIZE	Size of debug buffer area
ESIZE	Size of user buffer extension area
ISIZE	Size of initialization buffer
MONSIZE	Size of SYSTP monitor buffer
RDCSIZE	Size of buffer for the Natural data collector
RJESIZE	Initial Size of NATRJE buffer
RUNSIZE	Size of runtime buffer
WPSIZE	Sizes of Natural work pools

External Subprograms

The following parameters affect the dynamic loading and deletion of non-Natural programs:

Parameter	Short Description
CDYNAM	Dynamic loading of non-Natural programs
CSTATIC	Programs statically linked to Natural
DELETE	Deletion of dynamically loaded non-Natural programs
LIBNAM	Name of external program load library (BS2000/OSD, z/OS, TSO only)
RCA	Resolve addresses of static non-Natural programs
RCALIAS	External name definition for non-Natural programs

Output Reports and Work Files

The following parameters control various standard attributes used during the creation of Natural reports:

Parameter	Short Description
DL	Display Length for Output
EJ	Page Eject
FAMSTD	Overwriting of Print and Work File Access Method Assignments
HCAM	Hardcopy Access Method
HCDEST	Hardcopy Output Destination

Parameter	Short Description
INTENS	Printing of Intensified Fields
LS	Line Size for Natural Records
MAINPR	Override Default Output Report Number
MP	Maximum Number of Pages of a Report
PCNTRL	Print Control Characters
PM	Print Mode
PRINT	Printer Assignments
PS	Page Size for Natural Reports
SF	Spacing Factor
TQ	Translate Quotation Marks
TS	Translate Output from Programs in System Libraries
WORK	Work File Assignments
ZP	Zero Printing

Date/Time Settings

The following parameters affect the handling of date and time values by Natural as well as the internal date/time used by Natural:

Parameter	Short Description
DD	Day Differential
DFOUT	Date Format for Output
DFSTACK	Date Format for Stack
DFTITLE	Date Format in Default Page Title
DTFORM	Date Format
STACKD	Stack Delimiter Character
TD	Time Differential
YD	Year Differential
YSLW	Year Sliding Window

Limits

The following parameters can be used to prevent a single program from consuming an excessive amount of internal resources:

Parameter	Short Description
LE	Reaction when Limit for Processing Loop Exceeded
LT	Limit for Processing Loops
MADIO	Maximum DBMS Calls between Screen I/O Operations
MAXCL	Maximum Number of Program Calls
MAXYEAR	Maximum Year for Date/Time Values
MT	Maximum CPU Time
OVSIZE	Storage Thread Overflow Size
PD	Number of Pages captured by NATPAGE

Character Assignments

The following parameters can be used to change default character assignments:

Parameter	Short Description
CVMIN	Control Variable Modified at Input
FC	Filler Character for INPUT Statement
FCDP	Filler Character for Dynamically Protected Input Fields
CF	Character for Terminal Commands
DC	Decimal Character
HI	Help Character
IA	Input Assign Character
ID	Input Delimiter Character
SOSI	Shift-Out/Shift-In Codes for Double-Byte Character Set
THSEPCH	Thousands Separator Character

Terminal Communication

The following parameters affect the usage of Natural on video terminals:

Parameter	Short Description
ATTN	Attention Key Interrupt Support
CLEAR	Processing of CLEAR Key in NEXT Mode
DSC	Data Stream Compression (for 3270-Type Terminals)
ESCAPE	Ignore Terminal Commands %% and %.
IKEY	Processing of PA Keys and PF Keys
IM	Input Mode
KEY	Value Assignments to PA, PF, CLEAR Keys
LC	Lower- to Upper-Case Translation
ML	Position of Message Line
RM	Retransmit Modified Fields
SA	Sound Terminal Alarm
TMODEL	IBM 3270 Terminal Model
TTYTYPE	Terminal Type

Buffer Pools

The following parameters affect the Natural buffer pools:

Parameter	Short Description
BPCSIZE	Cache Size for Natural Buffer Pool
BPC64	Cache Size for Natural Buffer Pool
BPI	Buffer Pool Initialization
BPLIST	Name of Preload List for Natural Buffer Pool
BPMETH	Buffer Pool Space Search Algorithm
BPNAME	Name of Natural Global Buffer Pool
BPPROP	Global Buffer Pool Propagation
BPSFI	Object Search First in Buffer Pool
BPSIZE	Size of Natural Local Buffer Pool
BPTTEXT	Size of Text Segments in Natural Buffer Pool

Translation Tables

The following parameters can be used to overwrite various character translation tables used by Natural:

Parameter	Short Description
CCTAB	Printer Escape-Sequence Control Character
CP	Code Page
SCTAB	Scanner Character Type Table
TAB	Standard Output Translation Table
TABA1	EBCDIC to ASCII Translation Table
TABA2	ASCII to EBCDIC Translation Table
TABL	Translation Table for Output from "SYS" Libraries
TAB1	Alternative Output Translation Table
TAB2	Alternative Input Translation Table
UTAB1	Translation Table for Lower to Upper Case
UTAB2	Translation Table for Upper to Lower Case

Code Page and Unicode Support

Parameter	Short Description
CFICU	Unicode and Code Page Support

For an overview of other profile parameters involved in code page and Unicode support, refer to *Configuration and Administration of the Unicode/Code Page Environment, Profile Parameters* in the *Unicode and Code Page Support* document.

Usage of Profile Parameters

The following parameters affect the usage of Natural profile parameters:

Parameter	Short Description
DYNPARM	Control Use of Dynamic Parameters
PARM	Alternative Parameter Module
PLOG	Logging of Dynamic Parameters
PROFILE	Activate Dynamic Parameter Profile
SYS	Activate Set of Dynamic Profile Parameters
USER	Restrict the Use of Profile Parameters

Compiler Options

The following parameters can be used to control the Natural compiler:

Parameter	Short Description
CMPO	Compilation Options
FS	Default Format/Length Setting for User-Defined Variables
SM	Programming in Structured Mode
XREF	Activate Cross-Reference Feature

Debugging

The following parameters can be used for debugging purposes:

Parameter	Short Description
CANCEL	Session Cancellation with Dump
DBGERR	Automatic Start of Debugger at Runtime Error
DU	Dump Generation
DUE	Dump for Specific Errors
ETRACE	External Trace Function
ITRACE	Internal Trace Function
RELO	Storage Thread Relocation

Parameter	Short Description
TRACE	Define Components to be Traced
UPSI	z/VSE User Program Switches

Batch Mode

The following parameters apply if Natural is used in batch mode:

Parameter	Short Description
CC	Error Processing in Batch Mode
CPOBJIN	Code Page of Batch Input File.
CPPRINT	Code Page of Batch Output File.
CPSYNIN	Code Page of Batch Input File for Commands.
ECHO	Control Printing of Input Data
OBJIN	Use of CMOBJIN as Natural Input File
READER	System Logical Units for Input (z/VSE only)

TP Monitors

The following parameters apply if Natural is used with a TP monitor (Com-plete, CICS, CMS, IMS TM, UTM):

Parameter	Short Description
ASYNNAM	Output System ID for Asynchronous Processing (UTM)
OUTDEST	Output Destination for Asynchronous Processing (CICS, Com-plete, UTM)
PSEUDO	Pseudo-Conversational Mode (CICS)
SENDER	Screen Output Destination for Asynchronous Processing (CICS, Com-plete, IMS TM, UTM)
SKEY	Storage Protection Key
SUBSID	Subsystem ID

Database Access

The following parameters determine how Natural handles the access to databases:

Parameter	Short Description
DB	Database Types and Options
DBCLOSE	Database Close at Session End
DBOPEN	Database Open Without ETID
DBROLL	Database Calls Before Roll-Out
DBUPD	Database Updating
ENDBT	Issue BACKOUT TRANSACTION at Session End
ET	Execution of END/BACKOUT TRANSACTION Statements
ETDB	Database for Transaction Data
ETEOP	Issue END TRANSACTION at End of Program
ETIO	Issue END TRANSACTION upon Terminal I/O
ETSYNC	Issue Syncpoint upon End of Transaction/Backout Transaction
LFILE	Dynamic Specification of Logical File
OPRB	Database Open/Close Processing
RCFIND	Handling of Response Code 113 for FIND Statement
RCGET	Handling of Response Code 113 for GET Statement
TF	Translation of Databas ID/File Number
UDB	User Database ID

Natural with Adabas

The following parameters apply if Natural is used with Adabas:

Parameter	Short Description
ADANAME	Name of Adabas Link Routine
ADAMODE	Adabas Interface Mode
ADAPRM	Review/DB Support
ADASBV	Adabas Security By Value
ETID	Adabas User Identification
RI	Release ISNs
WH	Wait for Record in Hold Status

Natural with Other Software AG Products

Adabas Text Retrieval

Parameter	Short Description
TSIZE	Size of Buffer Area for Adabas Text Retrieval

Con-nect

Parameter	Short Description
CSIZE	Size of Con-nect/Com-pose Buffer Area

EntireX Broker

Parameter	Short Description
BFSIZE	Size of EntireX Broker Buffer

Entire DB

Parameter	Short Description
ZSIZE	Size of Entire DB Buffer Area

Entire System Server

Parameter	Short Description
ASIZE	Entire System Server Auxiliary Buffer

Entire Transaction Propagator

The following parameter applies if you are using the Entire Transaction Propagator:

Parameter	Short Description
ETPSIZE	Size of Entire Transaction Propagator Buffer

Natural Advanced Facilities

The following parameters apply if you are using Natural Advanced Facilities:

Parameter	Short Description
NAFSIZE	Size of Buffer for Natural Advanced Facilities
NAFUPF	Natural Advanced Facilities User Profile

Natural Connection

The following parameters apply if you are using Natural Connection:

Parameter	Short Description
PC	Control of Personal Computer Access Method (Natural Connection)
XSIZE	Size of Buffer for User Subsystem

Natural Database Interfaces

The following parameters apply if you are using the database interfaces listed below:

Parameter	Short Description
DB2SIZE	Size of Buffer Area for Natural DB2 or SQL/DS interface
DLISIZE	Size of Buffer Area for Natural DL/I interface
VSIZE	Size of Buffer Area for Natural VSAM interface

Natural Expert

Parameter	Short Description
EXCSIZE	Size of Buffer for Natural Expert C Interface
EXRSIZE	Size of Buffer for Natural Expert Rule Tables

Natural Optimizer Compiler

The following parameter applies if you are using the Natural Optimizer Compiler:

Parameter	Short Description
OPT	Control of Natural Optimizer Compiler

Natural Workstation Interface

The following parameter applies if you are using the Natural Workstation Interface:

Parameter	Short Description
WSISIZE	Buffer for Natural Workstation Interface

Software AG Editor

The following parameter applies if you are using the Software AG Editor:

Parameter	Short Description
EDBP	Software AG Editor Buffer Pool Definitions
EDPSIZE	Size of Software AG Editor Auxiliary Buffer Pool
SSIZE	Size of Buffer for the Software AG Editor

Miscellaneous Profile Parameters

Parameter	Short Description
CM	Command Mode
CPCVERR	Conversion Error
EMFM	Edit Mask Free Mode
ETA	Error Transaction Program
FREEGDA	Release GDA in Utility Mode

Parameter	Short Description
MAXROLL	Number of CMROLL Calls Before Roll-Out
MSGSF	Display System Error Messages in Full
NC	Use of Natural System Commands
OPF	Overwriting of Protected Fields by Helproutines
PLUGIN	Enable the Natural Plug-In Components
POS22	Version 2.2 Algorithm for POS System Function
RDCEXIT	Define Natural Data Collector User Exits
RECAT	Dynamic Recataloging
REINP	Issue Internal REINPUT Statement for Invalid Data
RFILE	File for Recordings
RPC	Remote Procedure-Call Settings
SI	Shift-In Code for Double-Byte Character Set
SL	Source-Line Length
S0	Shift-Out Code for Double-Byte Character Set
SORT	Control of Sort Program
SYNERR	Control of Syntax Errors
ULANG	User Language
ZD	Zero-Division Check

Session Initialization and Termination

The following parameters have an influence on the initialization or termination of a Natural session:

Parameter	Short Description
AUTO	Automatic Logon
ENDMSG	Display of Session-End Message
IMSG	Session Initialization Error Messages
ITERM	Session Termination in Case of Initialization Error
MENU	Menu Mode
NUCNAME	Name of Shared Nucleus
PROGRAM	Program to Receive Control after Natural Session
STACK	Place Data/Commands on the Stack
STEPLIB	Additional Steplib Library

Parameters Reserved for Internal Use



Caution: The values of the following parameters must not be changed!

These parameters are reserved for internal use by Natural.

Parameter	Short Description
ASPSIZE	(Internal Use)
CFWSIZE	(Internal Use)
LOG	(Internal Use)
NISN	(Internal Use)
RDACT	(Internal Use)
RDNODE	(Internal Use)
RDPORT	(Internal Use)
TPF	(Internal Use)
USERBUF	(Internal Use)

14

Using a Natural Parameter Module

- Using the Default Natural Parameter Module NATPARM 90
- Creating a New Natural Parameter Module 90
- NTPRM Macro - Create a Natural Parameter Module 90
- Restricting the Use of a Parameter Module 91
- Using Macros in a Natural Parameter Module 92

This document provides information on how to assemble a Natural parameter module.

The following topics are covered:

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

Using the Default Natural Parameter Module NATPARM

The default Natural parameter module NATPARM contains a set of predefined parameters that are sufficient for most environments. The module is delivered in source form to enable you to change it according to your requirements.

Creating a New Natural Parameter Module

Instead of using or modifying the default Natural parameter module, you can create one or several alternative Natural parameter modules for various purposes which can be loaded as appropriate using the Natural profile parameter PARM and whose use can be restricted to certain users (See *Restricting the Use of a Parameter Module*).

► To create a new (alternative) Natural parameter module

- 1 Assemble the macro NTPRM (see also *Assembler Macro Coding Conventions* below).
- 2 Add one or more of the optional parameter macros (see **below**).

If more than one parameter macro is specified, the NTPRM macro must be specified first; any other macros after the NTPRM macro can be specified in any order.



Note: It is not necessary to create separate parameter modules for batch and teleprocessing modes of operation. Those parameters which are not applicable to the environment in which Natural is executed are ignored.

NTPRM Macro - Create a Natural Parameter Module

The NTPRM macro must be assembled in order to create a Natural parameter module.

Generally, you can use the default values of the profile parameters in the NTPRM macro. If any of the default values do not suit your requirements, you can overwrite them with your own values.

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

NTPRM Syntax

The syntax for this macro is:

```
NTPRM parameter=value,...
```

Assembler Macro Coding Conventions

Assembler macro coding conventions must be adhered to when changing parameter values, for example,

- the first entry must begin in Column 2 or beyond and cannot extend beyond Column 71;
- continuation to another line is accomplished by placing a comma after the last entry, inserting a non-blank character in Column 72 and continuing the entry on the next line starting in Column 16;
- a parameter and its value must always be entered on the same line.

Restricting the Use of a Parameter Module

You can add the macro `NTUSER` to a parameter module to restrict its use to certain users.

▶ To restrict the use of a parameter module

- 1 Add the macro `NTUSER` to the parameter module.
- 2 In this macro, define the IDs of those users who are to be enabled to use that parameter module.

Only these users will be allowed to specify the name of that parameter module with the profile parameter `PARM`.

Using Macros in a Natural Parameter Module

A Natural parameter module contains the macro `NTPRM` in first place. In addition, you can specify the following macros in any order.

Macro	Function
NTALIAS	External names of non-Natural programs.
NTBPI	Buffer pool initialization.
NTCCTAB	Printer escape sequence definition.
NTCFICU	Enables Unicode support for various Unicode settings.
NTCMP0	Compilation options.
NTCSTAT	Programs statically linked to Natural.
NTDB	Database types and options.
NTDS	Define size of storage buffer
NTDYNP	Control use of dynamic parameters.
NTEDBP	Software AG editor buffer pool definitions
NTFILE	See <code>NTLFILE > Old NTFILE Macro Syntax</code>
NTLFILE	Specification of logical files.
NTOPRB	Database open/close processing.
NTOPT	Control of Natural Optimizer Compiler.
NTPRINT	Print file assignments.
NTPRM	Create a Natural Parameter Module
NTRPC	Handling of remote procedure calls.
NTSCTAB	Scanner characters.
NTSORT	Control of sort program.
NTSYS	Define and activate a set of dynamic profile parameters.
NTTAB	Standard output character translation.
NTTABA1	EBCDIC-ASCII translation.
NTTABA2	ASCII-EBCDIC translation.
NTTABL	"SYS" library output translation.
NTTAB1	Alternative output translation.
NTTAB2	Alternative input translation.
NTTF	Translation of database ID/file number.
NTTRACE	Define components to be traced.
NTUSER	Restrict use of profile parameter strings and modules.
NTUTAB1	Lower-case/upper-case translation.

Macro	Function
NTUTAB2	Upper-case/lower-case translation.
NTWEBIO	Enable or disable the rendering of certain features of the Natural Web I/O Interface display.
NTWORK	Work files assignments.
NTXML	Activate PARSE XML and REQUEST DOCUMENT statements.

15 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 Shared Nucleus under z/OS and z/VSE** Explains the function and the use of the Shared Natural nucleus.
- **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 restartability.

 **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.

16 Natural under z/OS

▪ Natural Subsystem	98
▪ Shared Nucleus	98
▪ TP Monitor Interfaces	98
▪ Interfaces to Database Management Systems	99
▪ Natural in Batch Mode under z/OS	99
▪ Natural as a Server under z/OS	99

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 `NAT4`.

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.

Shared Nucleus

The advantages of a Natural shared nucleus are explained in the section [Natural Shared Nucleus under z/OS and z/VSE](#).

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*.

17 Authorized Services Manager under z/OS

■ ASM Overview	102
■ ASM System Requirements	103
■ ASM Operation	104

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

The following topics are covered:

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).

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 [Installing the Natural GBP Operating Program](#).
- Natural under CICS is used in a z/OS Parallel Sysplex (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.

The Session Information Pool (SIP) holds the Natural session information records. In terminal-oriented non-conversational mode, the NCI and NII interfaces 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 data space is used as an intermediate buffer to avoid unnecessary access to the CF. Otherwise, the SIP is created in a data space.

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

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

APF Authorization

Link the modules NATASM vr (vr =version, release number) and NATBPMGR to an Authorized Program Facility (APF) library, specifying IEWL parameter AC(1). Refer to *Installation Procedure for Natural under z/OS*.

System Linkage Index (System LX)

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 only used if you run the SIP in a z/OS Parallel Sysplex environment. The space required can be calculated using the following formula:

$$30 \text{ KB} + (\text{SIP slot size in bytes} + 165) * (\text{number of SIP slots} + 8)$$

For 500 SIP slots of 512 bytes each, define:

```
STRUCTURE NAME(NATASM) SIZE(380) PREFLIST(CF1)
```

XCF Signalling Paths

To propagate buffer pool administration messages in a z/OS Parallel Sysplex environment, the XCF Signalling Services are used. 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* stands for the current Natural version and release number. 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 <i>v</i>	The specified value must match the value of the Natural profile parameter SUBSID (<i>v</i> =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 signalling 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 - 32767	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.

Parameter	Possible Values	Default	Comment
<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:

```
//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*s (*vr*=version, *re*lease, *system* maintenance level) 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	DPSERV 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.
TRSTART	Debugging function, only to be used at Software AG's advice. Activates the Trace Task. If the GTF is started and enabled for User Records 202, the trace records are written to the GTF.
TRSTOP	Deactivates the Trace Task.
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.

18

Natural Shared Nucleus under z/OS and z/VSE

▪ Environment-Independent Nucleus	110
▪ Creating a Shared Nucleus	113
▪ Installing a Shared Nucleus	114
▪ Linking Subproducts to the Nucleus	114
▪ Single-Environment Shared Nucleus	115
▪ Environment-Dependent Nucleus	116
▪ Statically Linked Non-Natural Programs	116
▪ Dynamically Called Non-Natural Programs	117

This document refers to the Natural shared nucleus under z/OS and z/VSE only.

Environment-Independent Nucleus

Natural can be split into two functional parts: an environment-independent nucleus and an environment-dependent nucleus.

The environment-independent part of the shared nucleus can reside in the shared area of the operating system; that is,

- in z/OS environments: the link pack area (LPA) or extended link pack area (ELPA),
- in z/VSE environments: the shared virtual area (SVA).

By executing from these special areas of the operating system, the independent nucleus can be commonly accessed (shared) by multiple address spaces (that is, regions or partitions), for example, CICS, Com-plete, TSO and batch mode, within the same operating system.

Components of the Shared Nucleus

The following basic modules must be linked together to build the independent (shared) Natural nucleus:

Module	Function
NATSTUB	Natural stub module.
NATURAL	Natural compiler and runtime.
NATCONFIG	Natural configuration module.
NATCMOD	Bundling module of C routines (server calls).
NATBPMGR	Natural buffer pool manager.
NAT2SORT	Natural Sort for all systems (if you wish to use a sort program, either Natural's internal one or an external one). It is also possible to place NAT2SORT in a load library from where it can be loaded dynamically at runtime; this requires that NAT2SORT is specified with the profile parameter RCA.
NATRPC <i>vr</i> or NTRPC	Natural RPC runtime. Note: If more than one version of this module is available, see the current <i>Natural Release Notes</i> for the available Natural RPC versions. If only a single version of this module is available, see the installation job to link a shared nucleus in the <i>Installation</i> documentation for the actual module name.
NATEDIT	Natural program editor and map editor.
NATTEXT	Natural syntax.
NATTXT2	Natural keywords.

Module	Function
NATTXT3	Substitution fragments for Natural error messages.
NATPM	Natural print mode.
INPL	INPL module.
NATEDT	Software AG Editor module.
NATLAST	Final include.

Terminal Drivers and Batch Mode Modules

The following modules are optional:

Module	Function
NATTTY	Natural Teletype Support
NAT3270	3270 Terminal Support
NAT3279	3279 Terminal Support
NATWEB	Web I/O Terminal Converter; see <i>Unicode Input/Output Handling in Natural Applications, Web I/O Interface</i> , in the <i>Unicode and Code Page Support</i> documentation.
NATBTCH	Natural Batch Module

Modules Required for Unicode and Code Page Support

For a list of the mainframe-specific modules to be linked for Unicode and Code Page Support, refer to *Configuration and Administration of the Unicode/Code Page Environment, ICU Library* in the *Unicode and Code Page Support* documentation.

Module Required for REQUEST DOCUMENT and PARSE XML Statement Support

Module	Function
NATXML	Nucleus Routine Module

For further information, see *Installation Steps for REQUEST DOCUMENT and PARSE XML* in the *Natural Installation* documentation.

Linking Additional Modules

Linking of additional modules may be required, for example, common user exits or user-defined programs used by all Natural regions. The entry name of the linked module must be `CMSTUB`.

Benefits of a Shared Nucleus

The benefits of a shared nucleus are:

- virtual storage relief;
- less paging activity, as there is only one copy of the nucleus in the system;
- less maintenance, as Zaps must be applied only once.

By removing the environment-independent parts of Natural and placing them in the shared nucleus, you achieve a significant reduction of the size of the environment-dependent nucleus, since only the environment-dependent part is loaded into the batch or TP-monitor address space, and the shared nucleus is accessed from the operating system's link pack area.

Since less storage is required by a Natural batch job, this results in less paging and better overall performance of the operating system. The more batch jobs that concurrently access the shared nucleus, the greater the savings.

As is the case with batch environments, Natural running in an online environment can also access the same common nucleus. In production environments which, for example, run Natural under multiple CICS regions, the savings in virtual storage can be substantial.

There are also benefits when you apply corrective fixes to the Natural nucleus, since you only need to apply these ZAPs once to the shared nucleus, which is then accessed by the multiple environments (for example, CICS, Com-plete, TSO and batch).

Additional benefits are possible if you use products such as Natural for VSAM, Natural for DB2, Natural for DL/I or Natural Advanced Facilities, since these products are all eligible to execute from the shared nucleus. When installing these products, you would simply place the `INCLUDE` statements specific to these products into the link-edit of the shared nucleus.

Administration Aspects

In any module installed in the LPA/ELPA or SVA, Zaps cannot be applied online, because the LPA/ELPA or SVA is write protected. Under z/OS, you can use the operator command `SETPROG` to load a new copy of the shared nucleus into the LPA/ELPA. However, to test corrective fixes in a specific environment, it is recommended that you use one of the following methods which can be adapted to suit your site-dependent needs:

Environment	Requirement
Batch Mode	Link-edit the shared nucleus to a load library which you add to the <code>STEPLIB</code> concatenation. The operating system will access this copy of the shared nucleus instead of the copy in the shared area.
CICS	<p>Link-edit the shared nucleus to a load library which you add to the <code>DFHRPL</code> concatenation in the CICS startup procedure. This allows CICS to load the shared nucleus from your <code>DFHRPL</code> library instead of from the shared area.</p> <p>You need to modify the <code>ALT</code> (alternate load table) entry for the shared nucleus to read <code>SHR=NO</code> so that CICS will access the <code>DFHRPL</code> libraries instead of the shared area.</p> <p>Users of CICS Version 3.3.0 and above make this change to the <code>PPT</code> entry for the shared nucleus instead, since the <code>ALT</code> has been eliminated in these releases of CICS:</p> <p>Specify <code>USELPACOPY (NO)</code> in z/OS and <code>USESVACOPY (NO)</code> in z/VSE, respectively, for this program definition.</p>
Com-plete/TPF	Link the shared nucleus to your Com-plete user program library and add the shared nucleus to the list of <code>RESIDENTPAGE</code> programs in your Com-plete <code>SYSPARMs</code> or load the shared nucleus dynamically as <code>RESIDENTPAGE</code> .
TSO	Link-edit the shared nucleus to the same load library that contains the TP-dependent nucleus for Natural under TSO. When the <code>CLIST</code> is executed, the operating system will access this copy of the shared nucleus instead of the copy in the shared area.
IMS TM	Link-edit the shared nucleus to a load library which you add to the <code>STEPLIB</code> concatenation in your procedure used for executing the IMS TM application region. When Natural is started, the operating system will access the shared nucleus from <code>STEPLIB</code> instead of from the shared area.

Creating a Shared Nucleus

The shared nucleus is created via the linkage editor in the SMA Job `NATI060` as an optional part of the base Natural installation.

When setting up the linkage editor `INCLUDE` statements for the shared nucleus, it is important to carefully follow the installation instructions outlined in the Natural *Installation* documentation.

A common error is to omit or add link-edit `INCLUDE` statements to the shared and/or non-shared nucleus, which can cause unpredictable results when you attempt to start a Natural session. If this happens, please review the installation instructions and if necessary, call Software AG support for assistance.

Installing a Shared Nucleus

The installation of the shared nucleus is described in the Natural *Installation* documentation in the installation sections for the various Natural TP monitor interfaces included in the *TP Monitor Interfaces* documentation. The following points should be noted in general:

- The shared nucleus is created by an additional link step. The target library for this link can be any library, in which the operating system loader searches for executable modules. For test purposes, it may be easier to first link the shared nucleus in one of the libraries in your `STEPLIB` (or `SEARCH` chain) and later into an LPA (or `SVA`) library for production. To avoid confusion, you should delete the module in the `STEPLIB` library when linking it into the LPA library.
- The name of the shared nucleus to be used is specified with the profile parameter `NUCNAME` in the Natural parameter module when installing the environment-dependent part. It is possible to specify `NUCNAME` as a dynamic parameter in the primary parameter input, but not in the `PROFILE` or `SYS` parameter strings.

Linking Subproducts to the Nucleus

Most Software AG subproducts can be linked either to the environment-independent Natural nucleus or to the environment-dependent part. Refer to the installation instructions of your subproducts.

The following Natural subproduct, however, must be linked to the environment-dependent part and cannot be linked to the shared nucleus:

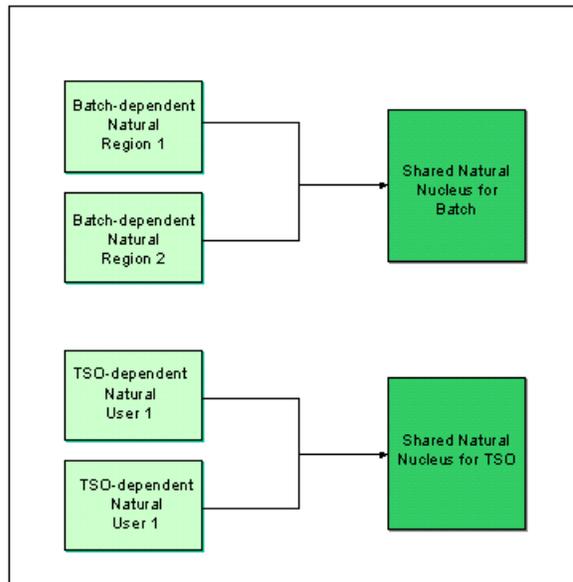
- The Adabas link routine (`ADALNK` or `ADAUSER`)

For a few other products, separate portions need to be linked to the shared nucleus as well as to the environment-dependent part. This is documented in detail with the respective subproducts.

Single-Environment Shared Nucleus

Some subproducts of Natural require that TP-specific modules be included in the Natural nucleus. In this case, you need to create one Single-Environment Shared Nucleus for each operating environment (for example, one for batch mode, one for TSO and one for CICS.) The advantage is still that all batch regions or all TSO users share their own Natural nucleus.

The following diagram shows an example for this situation:



As this concept of Single-Environment Shared Nuclei can always be installed, Software AG's System Maintenance Aid (SMA) generates this type of shared nucleus if the parameter `SHARED-NUC` is set to `Y`.

If all your single-environment shared nuclei are identical and do not contain TP-monitor-specific modules, you can then go from a single-environment shared nucleus to a multi-environment shared nucleus.

Environment-Dependent Nucleus

In addition to the environment-independent part of the shared Natural nucleus, every single Natural region runs one or more environment-dependent module(s), which differ(s) according to the actual environment; that is, Com-plete, CICS, IMS TM, TSO, or batch mode. The environment-dependent part receives control at the beginning of a session and checks whether the Natural nucleus is linked. If not, the shared nucleus is loaded or located and communication is established.

The following modules must be linked together to build the dependent part of Natural specific to each environment:

- the Natural environment-specific interface (that is, NCFNUC, NATCICS, NATIMS, NATTSO or NATOS/NATVSE) together with other interface-related modules;
- the environment-specific Natural parameter module NATPARM;
- Natural subproduct modules with entries defined in the internal CSTATIC list via macro NTINV, or specified as CSTATIC in the Natural parameter module;
- non-Natural programs defined as CSTATIC in the Natural parameter module.
- work-file and print-file modules for Com-plete, TSO or batch mode.

Statically Linked Non-Natural Programs

The Natural parameter module NATPARM contains the list of all non-Natural programs to be statically linked. This list consists of an internal part defined by the macro NTINV and an external part defined by the CSTATIC parameter. Each entry of the list consists of a program name and a V-constant which must be resolved by linking the corresponding module to the Natural parameter module.

The internal list is permanently present in the NATCONFIG module of the independent nucleus and is used if no parameter module is linked to the independent module. If there are non-Natural programs statically linked to the independent nucleus, a parameter module must be linked, too, where all these programs are defined.

Optionally, an alternative parameter module can be specified via the PARM parameter. An alternative parameter module has precedence over a linked parameter module. At session initialization time, up to three lists of statically linked programs are merged together. The base list for this merge is that of the actual parameter module, which means that only its entries are used. V-constants not resolved in this list are tried to be satisfied by the environment parameter module if an alternative parameter module is used. V-constants not resolved in the environment parameter module are tried to be satisfied by the environment-independent nucleus.

If a non-Natural program is to be statically linked to the independent nucleus, it must be specified in a parameter module linked to the independent nucleus as well as in the parameter module actually used for the session.

Additionally, "dynamic" linking of non-Natural programs defined for being statically linked is possible during initialization of a Natural session. Refer to the description of the `RCA` profile parameter for further details.

Dynamically Called Non-Natural Programs

Instead of statically linking a non-Natural program, you can also call it dynamically at execution time by using the Natural `CALL` statement. In this case, however, the program must not be defined as statically linked.

When the `CALL` statement is executed, Natural tries a dynamic load and call operation with the help of the environment (sub)system (for example, with `EXEC CICS LINK` under CICS).

19

Natural Roll Server Functionality

- Natural Roll-Server Overview 120
- Roll Server in a Single z/OS System 120
- Roll Server in a z/OS Parallel Sysplex Environment 122
- Roll File and LRB 124

This document covers the following topics:

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). 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. When the terminal I/O is completed, Natural requests the thread from the Roll Server, and continues the application. 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.

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 by the Natural command `DUMP ZAPS`. In addition, the list of applied Zaps is written to `JESMSG LG` during Roll Server startup.

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

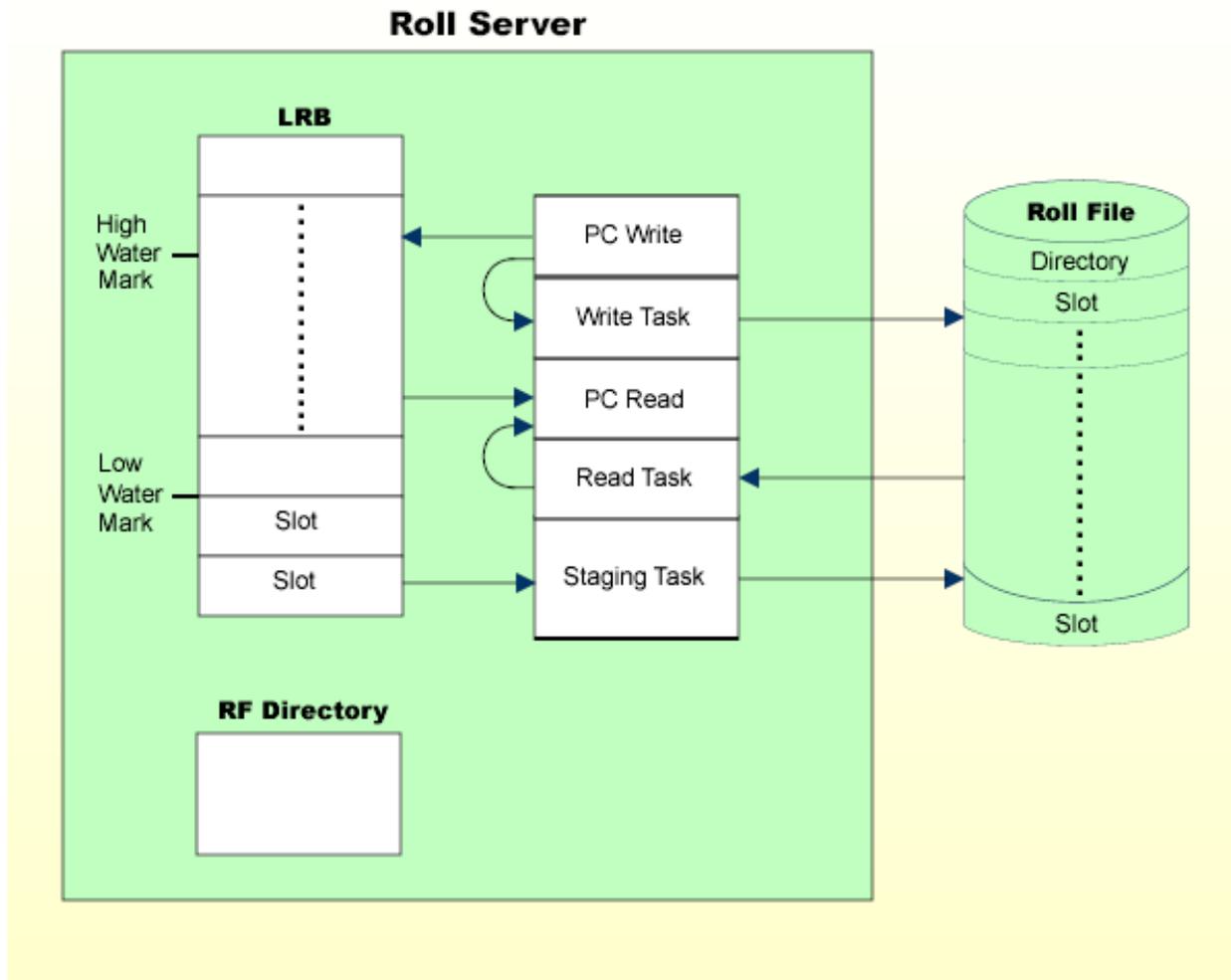
Roll Server in a Single z/OS System

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. The thread is also written to the roll file if it is larger than the LRB slot size. 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.

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



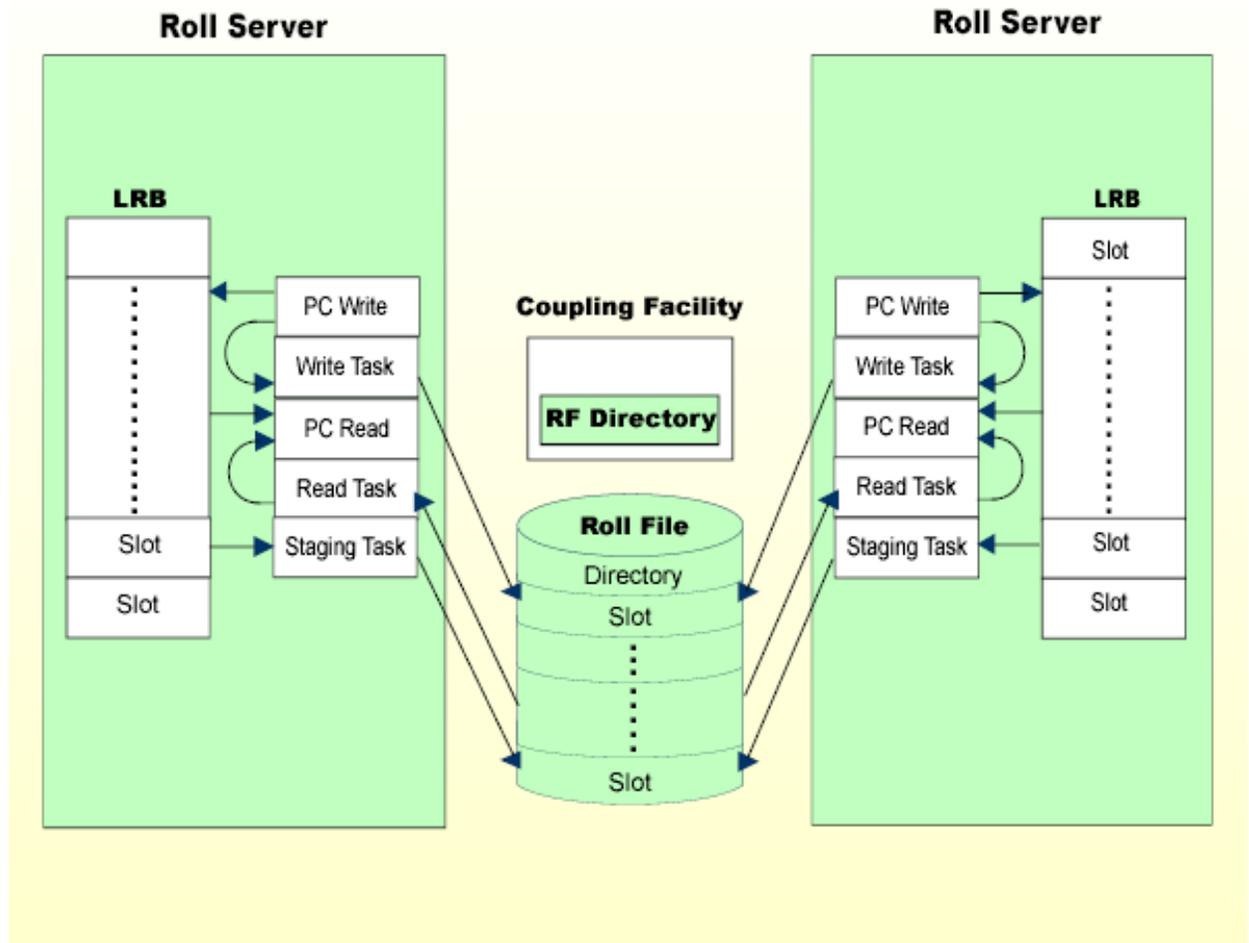
Roll Server in a z/OS Parallel Sysplex Environment

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 a 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, and written asynchronously from the LRB to the roll file. 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*.

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.

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. Slots must be larger than the largest compressed Natural thread expected.

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 or de-allocates its resources. Refer to *Roll Server Operation*, DEAL and TERM commands.

The local roll buffer is contained in a data space or z/OS memory object and subdivided into fixed-length slots. LRB slots may be smaller than roll-file slots. When a thread is larger than the LRB slot size, it is written directly to the roll file. The number of LRB slots and their size are Roll Server startup parameters; they are important factors in system performance.

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 roll files according to the following algorithm:

```

RN := (first four bytes of roll-server-user-id interpreted as positive integer)
modulo number of roll files + 1
ALLOCNUM := 0
FOR I = RN TO number of allocated roll files
  IF ROLLFILE(I) is not full THEN
    ALLOCNUM := I
    ESCAPE BOTTOM
  END-IF
END-FOR
IF ALLOCNUM = 0 THEN
  FOR I = 1 TO RN-1
    IF ROLLFILE(I) is not full
      ALLOCNUM := I
      ESCAPE BOTTOM
    END-IF
  END-FOR
END-IF
IF ALLOCNUM = 0 THEN
  Return 'roll file full' error
ELSE
  Allocate userid to roll file number ALLOCNUM
END-IF

```

where *roll-server-user-id* is a 16-byte, unique string provided by the Natural interface; for more information, see the corresponding TP monitor interface section in this documentation.

Example:

- There are five roll files and the *roll-server-user-id* is UF. Roll File 2 is full, but Roll File 3 has free slots available:

```
E4C64040 - 80000000 = 64C64040
64C64040 modulo 5 = 1
```

- Roll File 2 is the first file to be checked for a free slot. Since the check fails, Roll File 3 is tried next, and a free slot is found.
- User UF is therefore allocated to Roll File 3.

If this algorithm does not guarantee that your user IDs are evenly distributed among the roll files, the Roll Server's user exit NATRSU14 will help. This is especially relevant in server environments (see *Natural as a Server under z/OS*), because the first eight bytes of the roll server user ID are filled with the name of the server. For more information on this user exit, see *Natural Roll Server Operation, NATRSU14 User Exit*.

To see how evenly your user IDs are distributed, display the Roll-Server statistics using the Natural command SYSTP, selection "R".

20

Natural Roll Server Operation

▪ Roll Server System Requirements	128
▪ Formatting the Roll File	129
▪ Starting the Roll Server	133
▪ Roll Server Messages, Condition Codes and Abend Codes	136
▪ Return Codes and Reason Codes of the Roll Server Request	137
▪ Operating the Roll Server	137
▪ Roll Server Performance Tuning	138
▪ Roll Server User Exits	139

This document covers the following topics:

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

Roll Server System Requirements

This section describes the Roll Server system requirements.

The following topics are covered:

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

APF Authorization

Link the module NATRSM vr (vr =version, release number) to an Authorized Program Facility (APF) library, specifying IEWL parameter AC(1). Refer to *Installation Procedure for Natural under 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 IEASYS xx 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.

To avoid this, deactivate the Roll Server with the DEAL operator command and restart it afterwards.

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

The space required is calculated using the following formula:

$$24 \text{ KB} + (\text{RFN} + 1) * 1 \text{ KB} + (\text{RFS} + 8) * 160 \text{ bytes}$$

where RFN denotes the number of roll files and RFS denotes the total number of roll-file slots in all roll files.

Example:

- There are five roll files with 1000 slots each.

$$24 \text{ KB} + 6 \text{ KB} + 5008 * 160 \text{ bytes} = 24 \text{ KB} + 6 \text{ KB} + 783 \text{ KB} = 813 \text{ KB}$$

- The CF structure should thus be defined with 820 KB:

```
STRUCTURE NAME(NATROLLS) SIZE(820) PREFLIST(CF1)
```

XCF Signalling Paths

In a z/OS Parallel Sysplex environment, the Roll Servers communicate via the XCF Signalling 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 groupname, 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:

- Allocate it as a physical, sequential dataset with a fixed-record format.
- 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 `RFIPARMS`, 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.	
<i>slot-size</i>	The size of a roll file slot in bytes. This size is rounded to the next higher multiple of the block size used. 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.	
<i>number-of-slots</i>	The number of roll file slots to be allocated. This number is the maximum number of concurrently active users. This parameter is optional. If omitted, the entire roll file, as allocated, will be formatted.	

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

$$\text{number-of-cylinders} = \text{ceiling}(\text{number-of-slots} * \text{slot-size} / 30 * \text{block-size})$$

or in tracks

$$\text{number-of-tracks} = \text{ceiling}(\text{number-of-slots} * \text{slot-size} / 2 * \text{block-size})$$

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 dataset. When `RFIPRINT` is omitted, output is written to `JESMSGLG` using the `WTO` macro (`ROUTDCE=11`). Note that `RFIPRINT` must be specified for the `LIST` function.

NATRSRFI Conditon and Abend Codes:

The following condition codes are used:

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 <code>RFIPARMS</code> or <code>RFIPRINT</code> failed.
U0101	Open for roll file failed.

Example 1:

```
//FBRUNRFI JOB (FB,218),FB,CLASS=K,MSGCLASS=X,NOTIFY=FB
//FORMAT EXEC PGM=NATRSRFI
//STEPLIB DD DISP=SHR,DSN=NATURAL.NATvr.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
/*
```

Exerpt from resulting JESMSG LG:

```
+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.
```

where *vrs* = version, release, system maintenance level.

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*v*r. 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-rare, 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.

Parameter	Possible Values	Default	Comment
<i>number-of-roll-files</i>	0 - 5	1	In a z/OS non-Parallel Sysplex system, 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: <i>number-of-roll-files * number-of-LRB-slots</i>
<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>	1 - 9	7	Specifies the low water mark in steps of ten percent of the number of LRB slots. This parameter is ignored in a z/OS Parallel Sysplex environment.
<i>high-water-mark</i>	1 - 10	8	Analogous to low-water-mark parameter. Value "10" means that the staging task will never be activated. It is only recommended to specify "10" if the LRB is large enough to serve all simultaneously active Natural sessions. This parameter is ignored in a z/OS Parallel Sysplex environment.
<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.

Parameter	Possible Values	Default	Comment
<i>timeout-check-time</i>	0000 - 2359	none	<p>The time of day that the timeout check is to be run.</p> <p>Sessions will be deleted if they have been inactive longer than the non-activity time specified by the preceding parameter.</p> <p>If this parameter is omitted, no timeout check will be scheduled.</p> <p>This parameter can be changed using operator command <code>TIMEOUT</code>, see below.</p>
<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

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

The subsystem ID is `NA vr`, 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% (default), the high water mark is 80% (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 (ROUTCODE=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
```

The messages are explained in the section *Roll Server Messages* in the *Messages and Codes* document-ation.

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

Abend Code	Cause
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 or Natural IMS 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
DEAL	The Roll Server is stopped, but the address space is not terminated. The roll file directory and all modified LRB slots are written to the roll file. In a de-allocated status, the Roll Server can be restarted with new parameters and the old address space ID. The roll files can be reformatted in de-allocated status. If you do that, however, currently active Natural sessions are no longer restartable. 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.
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.
SNAP	Debugging function. The Roll Server's address space is dumped to SYSUDUMP.
START, <i>parmstring</i>	Reactivates the Roll Server with the specified parameters. You can only use this command in deactivated status.
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.

Command	Description	
TERM	<p>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.</p> <p>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.</p>	
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.
	? (or no specification)	Displays current timeout settings. The question mark (?) is optional and can be omitted.
TRSTART	<p>Debugging function, only to be used at Software AG's advice. Activates the Trace Task. If the General Trace Facility (GTF) is started and enabled for user records of Type 200, trace records are written to the GTF.</p>	
TRSTOP	Deactivates the Trace Task.	

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%).

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	

vrs stands for the current Natural version, release and system maintenance level.

- 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.

21 z/VSE Environment

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

- **Natural under z/VSE** Contains special consideration that apply when you are running Natural under z/VSE online or in batch mode.
- **Shared Natural Nucleus under z/VSE** Explains the function and the use of the Shared Natural nucleus.

22

Natural under z/VSE

- Natural Subsystem 146
- Natural Shared Nucleus 146
- TP Monitor Interfaces 146
- Interfaces to Database Management Systems 146
- Natural in Batch Mode under z/VSE 147

This document contains an overview of 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 `NATv`, where *v* is the product's version number.

Natural Shared Nucleus

The advantages of a Natural shared nucleus are explained in the section [Natural Shared Nucleus under z/OS and z/VSE](#).

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 DB/2
- 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*.

23

Natural Shared Nucleus under z/OS and z/VSE

The function and the use of the shared nucleus are almost identical under the operating systems z/OS and z/VSE.

To avoid redundant descriptions, reference is made to the corresponding document for the z/OS environment. There, the following topics are covered:

- *Environment-Independent Nucleus*
- *Creating a Shared Nucleus*
- *Installing a Shared Nucleus*
- *Linking Subproducts to the Nucleus*
- *Single-Environment Shared Nucleus*
- *Environment-Dependent Nucleus*
- *Statically Linked Non-Natural Programs*
- *Dynamically Called Non-Natural Program*

24 VM/CMS Environment

This part contains information about Natural in a VM/CMS environment.

- **Natural under VM/CMS** Explains topics such as issuing CP and CMS commands from Natural, reading the CMS program stack, hardcopy function and applying fixes to Natural. In addition, links are available to topics that apply when you are using Natural under CMS in batch mode.
- **Print File and Work File Support** Provides information on how to define print files and work files in the Natural parameter module.

25 Natural under VM/CMS

- Issuing CP and CMS Commands from Natural 154
- Reading the CMS Program Stack 154
- Hardcopy Function 155
- Applying Fixes to Natural 155
- Natural in Batch Mode under CMS 155
- Using TCP/IP Communication 155
- Calling Natural Subprograms from Rexx 156

This document contains special considerations that apply when running Natural under VM/CMS.

The following topics are covered:

Issuing CP and CMS Commands from Natural

You can use the Natural system command `CMS` to issue CP and CMS commands; for example, `CMS FLIST * DATA B` or `CMS CP SPOOL PRT *`.

If you enter `CMS` without parameters, a menu prompts you for a CP/CMS command. To exit from the menu enter a period (.) in the first position.

To issue CP or CMS commands from within a Natural program, code the following statement:

```
CALL 'CMS' command rc
```

where:

command is either an alphanumeric variable or a constant,

rc is a variable (format/length I4) which receives the return code from the CP or CMS command.

The second parameter (*rc*) is optional. Full command resolution is provided just as in normal CMS interactive command mode.

Reading the CMS Program Stack

To read a line from the CMS program stack into a Natural variable, code the following:

```
CALL 'CMSREAD' line
```

where *line* is an alphanumeric variable.

The line read from the program stack is either truncated or padded with blanks to fit the length of the variable.

If the program stack is empty, `CMSREAD` returns the character string `*EOD*`.

Hardcopy Function

The hardcopy function of the Natural CMS Interface is enabled by specifying parameter

```
HCAM=CMS
```

either in NATPARM ASSEMBLE, or dynamically when invoking Natural.

The Natural terminal command %H sends output to your virtual printer. Specifying %HL produces a file called NATURAL LISTING A.

Applying Fixes to Natural

Software AG provides fixes in the form of Zaps to remedy problems which are discovered after your Natural installation tape was shipped.

- Before applying the Zaps, ensure that you have made backup copies of the files.
- Use the NATZAP facility to apply these Zaps to your Natural text files.
- After applying the Zaps, the Natural module and DCSS must be built anew. Use the NATBLDM and NATBLDS commands respectively to do this.

For more information about NATZAP, type HELP NATZAP in the Natural installation user ID.

Natural in Batch Mode under CMS

See [Natural in Batch Mode \(All Environments\)](#) and [Natural in Batch under CMS](#).

Using TCP/IP Communication

The Natural statement REQUEST DOCUMENT is used to connect to an http server to retrieve HTML or XML files. The file TCPIP DATA contains configuration information for TCP/IP client programs. This file resides on the TCP/IP client minidisk and is typically accessed by the command

```
VMLINK TCPIP
```

If you are planning to use `REQUEST DOCUMENT`, include in a Rexx program to invoke Natural the line:

```
EXEC VMLINK TCPIP
```

Calling Natural Subprograms from Rexx

In a Rexx program, you can use the `CALLNAT` function to execute a Natural subprogram and pass parameters to it. The Rexx program must be executed while Natural is active, for example, by a `CALL` statement in a Natural program:

```
CALL 'CMS' 'EXEC MYREXX'
```

`MYREXX` can then execute a Natural subprogram with the `CALLNAT` function:

```
result = callnat('MYNAT', parm1, parm2)
```

`MYNAT` should reside in the same Natural library as the Natural program that called the Rexx program. Upon successful execution of `MYNAT`, the Rexx variable `result` will contain the name of the called subprogram, padded with blanks to a length of 8 (that is, in this example: `result='MYNAT '`). If an error was encountered, `result` will contain the Natural error number prefixed with "NAT", for example: `NAT0082`.

To exchange data between Rexx and Natural, you can use the statements `READ WORK FILE` and `WRITE WORK FILE`, preceded by

```
DEFINE WORK FILE n 'STEM rexxstem.'
```

The work file must be declared using the parameter macro `NTWORK` or the profile parameter `WORK` with `AM=CMS`. See also [Print File and Work File Support](#).

26

Print File and Work File Support

- Defining Print Files and Work Files 158
- Access Method STD 158
- Access Method CMS 158

This document describes special considerations on how to use print files and work files in Natural for VM/CMS.

The following topics are covered:

Defining Print Files and Work Files

Print files and work files are defined in the Natural parameter module with the macros `NTPRINT` and `NTWORK`. The corresponding dynamic parameters are `PRINT` and `WORK`.

In the following, the subparameters `AM` (access method) and `DEST` (destination) are described: They are available both in `NTPRINT` and `NTWORK`.

For both print and work files, Natural/CMS provides two access methods: “STD” and “CMS”.

Access Method STD

(AM=STD)

This access method uses the CMS simulation of the z/OS QSAM access method. Specify `AM=STD` if you want to read or write tape or spool (RDR, PRT, PUN) files, or if you want to read work files from z/OS-formatted disks.

A `FILEDEF` command must be issued before the corresponding print or work file is opened. The DD name to be used in the `FILEDEF` command is the name specified in the subparameter `DEST`.

Access Method CMS

(AM=CMS)

This access method uses the standard CMS file system to read and write CMS files on accessed mini disks and SFS directories.

The file names of the resulting CMS files are:

- `CMVRT nn` for print files,
- `CMWKF nn` for work files

where *nn* denotes the number of the file.

Their file type is the same as the name specified in the subparameter `DEST`. The filemode is always "A1".

Special Destination Names for AM=CMS

DEST=FD	Destination FD allows greater flexibility in assigning a CMS file to a Natural print or work file. When Natural opens a print or work file with destination FD, it searches for a FILEDEF for the DD name <code>CMPRTnn</code> or <code>CMWKFnn</code> , respectively (where <code>nn</code> denotes the print or work file number). It then uses the CMS file ID given in the FILEDEF command.
DEST=LISTING	This DEST setting applies to print files only. When specifying this destination, the print file is written to the CMS disk that has the most free space available. The CMS file ID is <code>CMPRTnn LISTING m</code> where <code>m</code> denotes the filemode of the mini disk that had the most free space. When the printer is closed, the print file is printed on the virtual printer and subsequently deleted.
DEST=UEXXXXXX	This DEST setting applies to print files only. If you specify a destination that starts with UEX, the print file is treated as if LISTING had been specified. In addition, a CMS command of this name is issued by Natural when the printer is closed. The CMS command (for example, a Rexx procedure) receives the CMS file ID of the print file as parameter.

Examples:

Example 1:

When the following FILEDEFS and NATPARM settings are in effect

```
FILEDEF CMWKF05 CLEAR
FILEDEF CMPRT01 DISK MY REPORT D
FILEDEF CMPRT04 DISK MY REPORT A
```

```
NETWORK (1),AM=CMS,DEST=FRED
NETWORK (5),AM=CMS,DEST=FD
NETWORK (6),AM=CMS,DEST=PAUL
NTPRINT (1,4),AM=CMS,DEST=FD
NTPRINT (2),AM=CMS,DEST=LISTING
NTPRINT (5),AM=CMS,DEST=PAUL
```

the following CMS files are produced:

```
CMWKF01 FRED A1  
FILE CMWKF05 A1  
CMWKF06 PAUL A1
```

```
MY REPORT D1  
MY REPORT A1  
CMWKF05 PAUL A1
```

The temporary file `CMVRT02 LISTING m` is printed and subsequently deleted (where *m* denotes the filemode of the minidisk that had the most free space).

Example 2:

```
NTPRINT (1),AM=CMS,DEST=UEXLOCAL
```

produces the CMS file:

```
CMVRT01 UEXLOCAL m
```

and the CMS command `UEXLOCAL` is issued with the file ID as parameter. If, for example, a Rexx procedure of this name exists, it can determine for which printer it was invoked by using `arg fn ft fm`.

Example 3:

Destinations can also be defined dynamically using the `DEFINE WORK FILE` statement. In addition, `DEFINE WORK FILE` can be used to specify input from, or output to, a Rexx stem:

```
DEFINE WORK FILE n 'STEM rexxstem.'
```

When opening an input file, Natural uses the value of `rexxstem.0` to determine the number of records to read. It then reads records from `rexxstem.1` to `rexxstem.max` (with `max = rexxstem.0`) before returning end-of-data.

For an output file, Natural writes `rexxstem.1` to `rexxstem.n`, and sets `rexxstem.0` to *n* when the work file is closed.

27

BS2000/OSD Environment

- Related Topics 162
- Other Natural Functions for BS2000/OSD-Specific Purposes 163

This part contains special considerations that apply when running Natural under the operating system BS2000/OSD.

- **Natural Shared Nucleus under BS2000/OSD** Explains the use of a common shared Natural nucleus, which is possible batch mode and under the TP monitors TIAM and UTM.
- **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.
- **Siemens Terminal Types Supported** Provides information on the various types of Siemens terminals that are supported by Natural under BS2000/OSD
- **Function Key Support with 9750 Devices** Describes the specific Natural function key assignments that are supported for Siemens devices of type 9750.
- **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/OSD.
- **RPC Server Front-End** Describes the RPC server front-end for an RPC batch server environment under BS2000/OSD with the print file/work file server NATPWSV2.

Related Topics

See also:

- *Using Natural with TP Monitors*
- *Natural under UTM*
- *Natural under TIAM*
- *Natural in Batch Mode under BS2000/OSD*

Other Natural Functions for BS2000/OSD-Specific Purposes

Natural provides the following functions for BS2000/OSD-specific purposes:

- **P-Key Utility**
Supports the loading of programmable P keys on Siemens 975X terminals (under UTM and TIAM).
- **Swap Pool Manager**
Controls the use of the Natural swap pool (under UTM and under CICS).

These functions are part of the Natural utility SYSTP.

28

Natural Shared Nucleus under BS2000/OSD

- Rules for Using a Natural Shared Nucleus 166

This document contains the rules that apply when you use a Natural shared nucleus, which is possible in **Batch** mode, under the TP monitors TIAM and/or UTM.

The following topic is covered:

Concerning UTM, see also *Several Applications with one Common Natural* in the *Natural TP Monitor Interfaces* documentation.

Rules for Using a Natural Shared Nucleus

With a Natural shared nucleus under BS2000/OSD, the rules given below apply:

1. The shared Natural nucleus is linked *without* the corresponding reentrant parts of the batch, TIAM and UTM drivers (these modules must be linked to the front-end part of the corresponding application).

Example: The name of the shared Natural nucleus is NATSHARE.

```
/EXEC $TSOSLNK
MOD NATSHARE,XREF=YES,MAP=Y,XDSEC=Y, SORT=Y
LINK-SYMBOLS *KEEP
INCLUDE NATINV,libname
.
.
INCLUDE NATURAL,libname
.
.
INCLUDE NATLAST,libname
BIND
/SETSW ON=1
LIB NATURAL.USER.MOD,BOTH
PAR O=Y
ADDR *OMF
END
/SETSW OFF=1
```

2. Batch, TIAM and UTM application-specific Natural parameter modules are also linked to the front-end part of the corresponding application. In addition, the shared Natural nucleus can contain a common Natural parameter module, for example, for `CSTATIC` entries. The name chosen for the linked Natural nucleus is also identical with the name of the global common memory pool into which Natural is loaded. This name is to be used as operand for the following keyword parameters:

NUCNAME	in the macros NAMBS2, NAMTIAM and NATUTM
NAME	in CMPSTART and ADDON (BS2STUB)

Example:

```
NRTSTART NAMTIAM CODE=FRONT,
        NUCNAME=NATSHARE
        PARMODE=(31, ABOVE),
        .
        .
NUTFRONT NATUTM APPLNAM=NATUTM,
        .
        .
        NUCNAME=NATSHARE
        PARMODE=(31, ABOVE)
```

- The shared Natural nucleus is started by the program [CMPSTART](#).

Example:

```
/EXEC (CMPSTART, NATURAL.MOD)
NAME=NATSHARE, SIZE=2MB, POSI=ABOVE, ADDR=250, SCOP=GLOBAL
PFIY=YES, LIBR=NATURAL.USER.MOD
```

- The link to the shared Natural nucleus is created in the batch, TIAM or UTM applications through the generation of the macro [BS2STUB](#); refer to [CMPSTART Program](#).

Example:

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

- The front-end part of the applications must contain the reentrant part of the corresponding driver (NAMBS2 CODE=RENT, NAMTIAM CODE=RENT or NURENT).

Examples:**Front-end Part of NAMTIAM:**

```
/EXEC $TSOSLNK                                /* Front part of NAMTIAM
PROG NATURAL,LOADPT=X'1000000',XREF=YES
TRAITS RMODE=ANY,AMODE=31
INCLUDE NRTSTART,libname                    /* Front part of NAMTIAM
INCLUDE NRTRENT,libname                    /* Reentrant part of NAMTIAM
INCLUDE NRTSTUB,libname                   /* BS2STUB
INCLUDE NRTPARM,libname                   /* Natural Parameter Module
```

UTM Front-end Part:

```
/EXEC $TSOSLNK                                /* UTM Front-end part
PROG NUTvrs,FILENAM=NATUTM,LOADPT=X'1000000',XREF=YES
TRAITS RMODE=ANY,AMODE=31
INCLUDE KDCNUT,libname                    /* UTM KDCROOT
INCLUDE NUTSTART,libname                 /* NATUTM
INCLUDE NUTRENT,libname                 /* NURENT
INCLUDE NUTSTUB,libname                 /* BS2STUB
INCLUDE NUTPARM,libname                 /* Natural Parameter Module
INCLUDE SWPPARM,libname                 /* Swap Pool Parameter Module
```

where *libname* is the name of the library and *vrs* stands for version, release and system maintenance level of the product.

29 Refresh of Natural Load Pool

■ Prerequisites/Restrictions	170
■ Procedure	170
■ Keyword Parameters for the Program PREFRESH	171

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.

The following topics are covered:

Prerequisites/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 UTM 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 (reentrant) 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.
TIM2=20	The default value is 20 seconds.

30 Optimization of Message Handling

- Screen Output Handling 176
- Restoring the Screen Content 176

The following topics are covered:

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'`

31

Siemens Terminal Types Supported by Natural

▪ Type 9748	178
▪ 975n Series	178
▪ Type 9763M	179

This document contains information on how Natural supports Siemens terminal types.

The following types are supported:

- 974 *n*
- 975 *n*
- 976 *n*
- telex devices

Type 9748

At present, there are significantly different data stations of the type 9748. Depending on the age of the device, better support can be provided in 9750, 9755 or 9756 mode. Devices from older series of this type should be defined as 9750 because of the limited number of fields per line.

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 UTM, 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.

32

Function Key Support with 9750 Devices

- Key Assignment 182
- Modes for Key Assignment 182

The following topics are covered:

Key Assignment

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

As current Siemens terminal device types 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:

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).

33 Common Memory Pools

- Global Common Memory Pools 186
- Local Common Memory Pools 190

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

The following topics are covered:

Global Common Memory Pools

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

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



Note: In the following, *vrs* stands for version, release, system maintenance level of the product.

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.

0	The common memory pool is not ready (in creation mode).
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 JVSUFEX (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.

JVSUFIX - Suffix of the Job Variable Name

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

JVSUFIX=xxxxxxxx	Maximally 8 characters for the second part of the job variable name.
JVSUFIX=.SAG.JV	This is the default value.

Example:

```
NAME=EDTvrsgA,TYPE=EDT,JV=YES,JVSUFIX=.SAG##JV . . . . .
```

The Jobvariable name is EDTvrsgA.SAG##JV.

Operator Commands

These operator commands terminate a global common memory pool:

```
/INTRtsn,STOP
```

or

```
/INTRtsn,END
```

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

```
/INTRtsn,DPRM
```

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

```
/INTRtsn,DUMP
```

Examples:

- **To start a global load pool (shared nucleus)**

```

/.NATSHRE LOGON
/OPTION DUMP=YES
/SYSFILE SYSDTA=(SYSCMD)
/SYSFILE SYSOUT=LST.NATSHARE
/EXEC (CMPSTART,NATvrs.MOD)
NAME=NATSHARE,SIZE=2MB,POSI=ABOVE,ADDR=250,SCOP=GLOBAL
PFI=YES,ALNK=NO,LIBR=NATvrs.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 (PFI=YES).

```

- **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'.

```

- **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 SYSDTA=(PRIMARY)
/LOGOFF N

```

COMPEND Program

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

Example:

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

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/OSD:

- [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:

<code>name BS2STUB PARMOD=nn,PROGMOD=xxx</code>

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 - 24/31 Bit Addressing Mode

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 | PFIK | 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 ACCS=READ, the user ID must be authorized for the BS2000/OSD CSTMP macro in the user catalog (JOIN command with C-M=YES).
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.

ADDR= <i>number</i>	<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 AUTOLNK function is deactivated.
ALNK=YES	By default, the AUTOLNK 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 PREFIX 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/OSD 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
```

34 Calling Dynamically Reloadable 3GL Programs in a Natural Application

- Storage Allocation Rule 198
- Thread-Creation Rule 198
- Address-Mode Dependencies 198

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

The following topics are covered:

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 `NAAADDR` 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 UTM partial programs which are not 31-bit enabled from Natural/UTM driver via PENDING](#)

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. 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.	AMODE=24
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 UTM Partial Programs which are not 31-bit enabled from Natural/UTM 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 UTM KDCS interface, Natural switches back to the 24-bit address mode, and when control is returned to the UTM driver, a switch-back occurs to the 31-bit address mode.

35

Print File/Work File Server NATPWSV2

- Setup 202
- Operation 203

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

The following topics are covered:

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:

```
/EXEC $TSOSLNK
PROG PWKSRV, FILENAM=E.NATPWSV2, LOADPT=X'1000000', XREF=YES
TRAITS RMODE=ANY, AMODE=31
INCLUDE NATPWSV2, NATURAL.NATvrs.MOD
INCLUDE PWKSTUB, USERLIB
BIND
```

where:

vrs stands for the current version of Natural for Mainframes and
USERLIB stands for the user-specific 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
```

Terminating the Print File/Work File Server

The print file/work file server can be terminated by way of P1 eventing, using the program CMPEND. 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.NATvrs.MOD,-  
/      ELEMENT=CMPEND)  
PWK#POOL      <== name of the print file/work file control pool  
/SET-JOB-STEP  
/ASSIGN-SYSDTA TO=*PRIMARY  
/EXIT-PROCEDURE
```

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

36

RPC Server Front-End

- Setup 206

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

The following topic is covered below:

Setup

For the generation of the Natural RPC batch server, the front-end 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 front-end 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 front-end part of the Natural RPC batch server:

```
/EXEC $TSOSLNK
PROG SERVER, FILENAM=BATCH.SERVER, LOADPT=X'1000000', XREF=YES
TRAITS RMODE=ANY, AMODE=31
INCLUDE NATSFED2, NATvrs.MOD      Must be bound as first module
INCLUDE RPCSFE, NATvrs.MOD       RPC front-end stub
INCLUDE SERVFRNT, USERLIB        Front-end part of Natural batch driver
INCLUDE SERVRENT, USERLIB        Reentrant part of Natural batch driver
INCLUDE RPCSTUB2, USERLIB        BS2STUB (see above)
INCLUDE SWPRMSRV, USERLIB        Swap pool parameter module
INCLUDE NATPRMSV, USERLIB        Natural parameter module
INCLUDE ADAUSER, ADAvrs.MOD
INCLUDE SSFB2C, ADAvrs.MOD
BIND
```

where:

vrs stands for the current version of Natural for Mainframes or Adabas for Mainframes and

USERLIB stands for the user-specific 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 reentrant part of the Natural RPC batch server:

```
/EXEC $TSOSLNK
MOD RPCSERV,XREF=Y,MAP=Y,XDSEC=Y, SORT=YES
TRAITS RMODE=ANY,AMODE=ANY
LINK-SYMBOLS *NOESD
INCLUDE NATINV,NATvrs.MOD      Must be bound as first module
INCLUDE NATURAL,NATvrs.MOD    Natural nucleus
COMMENT NATWKFB2,NATvrs.MOD   Is replaced by
INCLUDE NATFSTB2,NATvrs.MOD   print file/work file server stub
.                               All other
.                               moduls
INCLUDE NATLAST,,NATvrs.MOD
BIND
```

where *vrs* stands for the current 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,
AUTORPC=ON,TRYALT=OFF
),
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 NATvrs.EDIT.WORKFILE, LINK=CMEDIT
/FILE NATvrs.SERVER.ROLLFILE, LINK=PAMNAT, SHARUPD=YES
/FILE SERVER.MSG, LINK=P10           Is required for the server messages
/FILE ADAvrs.MOD, LINK=DDLIB
/FILE ADAPARM, LINK=DDLNKPAR
/FILE EXXvrs.LIB, LINK=BLSLIB01     Broker
/FILE EXXvrs.LIB, LINK=ETBLIB      Load library
/SYSFILE SYSDTA=SERVERPARMS
/EXEC BATCH.SERVER
/LOGOFF N
```

where *vrs* stands for the current 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.

37

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 CMS** Provides special considerations that refer to Natural in batch mode under CMS.
- **Natural in Batch Mode under BS2000/OSD** Provides special considerations that refer to Natural in batch mode under the operating system BS2000/OSD.
- **Natural in Batch Mode (All Environments)** Contains general considerations that apply when running Natural in batch: Adabas datasets, sort datasets, subtasking session support for batch environments.

See also *Batch Mode* in the section *Profile Parameters Grouped by Function* for an overview of the Natural profile parameters that apply if Natural is used in batch mode.

38

Natural in Batch Mode under z/OS

- General Information about the Natural z/OS Batch Mode Interface 214
- Natural z/OS Generation Parameters 214
- Datasets Used by Natural in z/OS Batch Mode 217

This document contains special considerations that refer to Natural in batch mode under the operating system z/OS.

The following topics are covered:

For considerations that refer to Natural in batch mode generally, see also:

- [Adabas Datasets](#)
- [Sort Datasets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

General Information about the Natural z/OS Batch Mode Interface

The Natural z/OS batch mode interface NATOS consists of a number of service routines interfacing with the z/OS operating system.

NATOS is supplied as a source module and can be customized to meet your requirements; see also *Installing Natural under z/OS*. You can either assemble and link NATOS to the Natural nucleus or you can run it separately, connecting with a shared nucleus.

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](#).

Natural z/OS Generation Parameters

The NTOS macro contains several generation parameters to change Natural for z/OS batch mode interface's internal defaults.

These parameters are: [ABEXIT](#) | [LBPNAME](#) | [LE370](#) | [SUBPOOL](#) | [TI0BSZ1](#) | [TI0BSZ2](#) | [USERID](#)

ABEXIT - Abend Processing

This parameter specifies the mode of abend processing within Natural.

ABEXIT=ESTAE	Natural intercepts all abends and issues the appropriate error messages. This is the default value.
ABEXIT=SPIE	Only program checks (SOCx abends) are intercepted as they used to be with Natural Version 2.1.
ABEXIT=NONE	Natural does not intercept any abends or program checks at all. This value corresponds to profile parameter DU=FORCE. The setting ABEXIT=NONE is not recommended because some functions which require the abend interception will not work any longer. The usage of profile parameter MT will cause an abend U0322 instead of error NAT0953 when the CPU time limit is reached.

LBPNAME - Sharing of Local Buffer Pools

This parameter controls the sharing of the local buffer pools when running multiple Natural sessions within the same region. It defines the name of the shared buffer pool environment and is used to locate the shared local buffer pool.

LBPNAME= <i>name</i>	<i>name</i> can be 1-8 characters long.
LBPNAME=	The default value is none, that is, the local buffer pools are not shared.

When running multiple Natural sessions in a z/OS batch or TSO environment concurrently, each session allocates storage for a separate local buffer pool. Except for the Natural z/OS batch mode server, the local buffer pools are not shared by default, that is, if the different sessions use the same Natural objects, these have to be loaded once for each session separately. If *name* is specified, all Natural sessions will share the same local buffer pool.

LE370 - Use of IBM Language Environment

This parameter specifies whether Natural is to run in the IBM Language Environment (LE).

LE370=YES	You can call external subprograms according to the IBM calling conventions.
LE370=NO	You can only call main programs of the Language Environment. This is the default value. This means a new LE enclave is created and terminated for each CALL statement.
LE370=POSIX	You can call external subprograms according to the LE calling conventions with POSIX semantics, that is, the LE is initialized with runtime option POSIX(ON).

LE370=AMODE24	Support of external subprograms linked in addressing mode 24. LE is initialized with options ALL31=(OFF) and STACK=(, , BELOW). Specify AMODE24 if one or more external subroutines are linked in AMODE 24. Parameter value AMODE24 enables support for external subprograms linked in either AMODE 24 or AMODE 31. If parameter value AMODE24 is omitted, an error is raised if an external subprogram is to be called that is linked in AMODE 24.
LE370=NOHDLR	No setting of an LE error handler is done by Natural during the call of LE subprograms. This means, if an unhandled error occurs during the execution of an LE subprogram, the LE enclave is terminated and so the Natural session is lost.

Multiple parameter values are enclosed in brackets, for example:

```
LE370=(YES,POSIX,AMODE24)
```

For more information about Natural running with the IBM Language Environment, refer to *Natural Execution - Miscellaneous Topics, LE Subprograms*.

SUBPOOL - Storage Subpool for GETMAIN Requests

This parameter defines the storage subpool for GETMAIN requests.

SUBPOOL=nnn	Possible value for nnn: "0" to "127".
SUBPOOL=0	The default value is "0".

TIOBSZ1 – Size of the Primary I/O Buffer for Batch Processing

This parameter specifies the size of the primary I/O buffer for batch processing. It will be allocated below the 16 MB line. For server processing, parameter TIOBSZ2 is used instead.

TIOBSZ1=nnnn	nnnn is the size of the primary I/O buffer in bytes. Possible values: 4096 - 16777216. The default value is 8192.
--------------	--

See also generation parameter TIOBSZ in *Natural TSO Interface Generation Parameters* in the *TP Monitor Interfaces* documentation.

TIOBSZ2 – Size of the Primary I/O Buffer for Server Processing

This parameter specifies the size of the primary I/O buffer for server processing. It will be allocated below the 16 MB line. For batch processing, parameter TIOBSZ1 is used instead.

TIOBSZ2=nnnn	nnnn is the size of the primary I/O buffer in bytes. Possible values: 4096 - 16777216. The default value is 65520.
--------------	---

See also generation parameter TIOBSZ in *Natural TSO Interface Generation Parameters* in the *TP Monitor Interfaces* documentation.

USERID - Content of System Variable *INIT-USER

This parameter specifies the content of the system variable *INIT-USER.

USERID=YES	The variable is set to either the user ID from the security access control block (ACEE) if a security package (as RACF or ACF2) is involved or the user parameter from the job card.
USERID=NO	The user ID is the job name. This is the default value.

The content of *INIT-USER can be changed by the user ID exit NATUEX1 during session initialization. For more information, see *Configuring Natural, Natural User Exits, NATUEX1 - User Exit for Authorization Control*.

Datasets Used by Natural in z/OS Batch Mode

The following datasets are required if certain functions are used during a Natural z/OS batch mode session:

Dataset	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

Dataset	Explanation
STEPLIB	Load Library for External Modules
CMWKFnn	Work Files 01-32

These datasets 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 dataset 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, please refer to *Installing the Software AG Editor* in the *Natural Installation* documentation.

CMHCOPY - Optional Report Output for Hardcopy

The default name of the hardcopy print output dataset 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 dataset. The default dataset name CMHCOPY implies CLOSE=FIN for the hardcopy print dataset, that is, after the dataset has been opened for output, any subsequent change of the hardcopy print output dataset name will not be honored. If a different name is defined at open time, the hardcopy dataset will be closed according to subparameter CLOSE of profile parameter PRINT for Print File 0.

During the session, the hardcopy dataset 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 dataset can be used to read data by the Natural `INPUT` statement rather than from the primary input dataset `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 dataset `CMPLOG` is available, the evaluated dynamic profile parameters are written to this dataset during session initialization. If dataset `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.

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 Dataset

`CMPRMIN` can be used as a dynamic parameter dataset 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 datasets 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 dataset 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` dataset. To define the Natural components that are to be traced, the profile parameter `TRACE` is required.

If dataset `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 dataset 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 shared 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 datasets 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 dataset, the defaults are defined by the `WORK` profile parameter or the `NETWORK` macro in the Natural parameter module.

The work file dataset names can be overwritten by subparameter `DEST`.

39

Natural in Batch Mode under z/VSE

▪ NATVSE - Natural z/VSE Batch Mode Interface	224
▪ NTVSE Macro - Generation Parameters for Natural under z/VSE	224
▪ Natural Datasets Used under a z/VSE Batch Mode Session	230
▪ NATVSE Print and Work File Support for z/VSE Library Members	235
▪ NATVSE Dynamic Work File Allocation (DYNALLOC) Support	237
▪ Debugging Facilities for Natural under z/VSE	240
▪ NATVSE Attention Interrupts	244

This document contains special considerations that refer to Natural in batch mode under the operating system z/VSE.

The following topics are covered:

For considerations that refer to Natural in batch mode generally, see also:

- [Adabas Datasets](#)
- [Sort Datasets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

NATVSE - Natural z/VSE Batch Mode Interface

The Natural z/VSE batch mode interface NATVSE consists of a number of service routines interfacing with the z/VSE operating system.

NATVSE is supplied as a source module and can be customized to meet your requirements; see also *Installing Natural under z/VSE*, where you can set the generation parameters. You can either assemble and link it to the Natural nucleus or you can run it separately, connecting with a shared nucleus.

NATVSE 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](#).

NTVSE Macro - Generation Parameters for Natural under z/VSE

The NTVSE macro contains several generation parameters (to be set in the NATVSE copy book) to change the NATVSE internal defaults.

These parameters are:

NAME | BUFSIZE | CANCEL | DSECTS | FILEID | FILMNGR | FILSCAN | FLUSH | IDUMP | LE370 | LIBRID
| MAXABND | RCSIZE | RJEUSER | SEGMENT | THDSIZE | USERID | WAITIME

NAME - Name of Relocatable Module

NAME specifies the name of the relocatable module to be created by the given assembly. Possible values:

NAME=xxxxxx	xxxxxx = name of the relocatable module to be created. Maximum length: 8 characters.
NAME=NATVSE	This is the default value.

BUFSIZE - Size of Natural I/O Buffer

BUFSIZE specifies the size of the Natural I/O buffer which is used for all input and output operations.

BUFSIZE=nnnn	nnnn specifies the size of the Natural I/O buffer in KB. Minimum value: 8 (KB).
BUFSIZE=8192	This is the default value.

CANCEL - Session Termination

CANCEL specifies how the Natural z/VSE interface is to proceed at session termination. Possible values:

CANCEL=YES	The job is cancelled by CANCEL or JDUMP macros, unless either Natural terminated normally or the session was terminated by the Natural TERMINATE statement. This is the default value.
CANCEL=NO	NATVSE always terminates with RETURN or EOJ macros with a set return code (the same effect is achieved when you set the Session Abend Flag UPSI XXXX1XX).



Note: CANCEL=YES is the default for compatibility reasons. However, it is recommended to specify CANCEL=NO, particularly to take advantage of VSE conditional job control facilities.

DSECTS - Listing of Operating System DSECTS

DSECTS specifies whether operating system DSECTS are to be listed. Possible values:

DSECTS=YES	Listing of operating system DSECTS takes place.
DSECTS=NO	Operating system DSECTS are not to be listed. This is the default value.

FILEID - Check of Label Information

FILEID specifies a string of up to 8 characters which is checked against the start of a DLBL or TLBL file ID. If it matches, this label information is ignored. Possible values:

FILEID=xxxxxxx	xxxxxxx = any character string which must be enclosed in apostrophes if it contains special characters.
FILEID='IGNORE'	This is the default value.

This is particularly helpful when DLBL or TLBL statements for CMWKFnn* and/or CMPRTnn* are supplied in the (partition) standard labels, but should not be used.

If, for example, a // DLBL CMPRT01, '...' statement is found, it is not possible to direct a WRITE(1) output to a printer SPOOL. To do so, use the JCS statement // DLBL CMPRT01, 'IGNORE' and a suitable printer assignment of the relevant SYSnnn.

FILMNGR - Management of Print or Work File in Natural

FILMNGR specifies how a print or a work file is to be managed in Natural. Possible values:

FILMNGR=YES	The fact that there is label information for a print or a work file and the fact that LABEL=OFF/ON is specified for an unlabelled work file indicates to Natural that this file is available. In particular, this is relevant if the Natural print and work files are to be managed by a file management system. This is the default value.
FILMNGR=NO	The logical unit number of the Natural print or work file must be assigned to the appropriate device type.

FILSCAN - Scanning of Print or Work Files

FILSCAN specifies whether print or work files are to be scanned. Possible values:

FILSCAN=YES	The Natural z/VSE interface scans the z/VSE label area for all Natural print and work files for which no specific file access method has been defined via Natural session parameters, as this may cause overhead. This is the default value.
FILSCAN=NO	Access to all Natural print and work files must be specified explicitly via session parameters in order to be “available”. This concentrates all file access efforts on the defined files.

FLUSH - Flush Card Input Files until EOF

FLUSH specifies how the Natural z/VSE interface is to proceed at session termination with the CMSYNIN/CMOBBIN card input files. Possible values:

FLUSH=YES	At session termination, the Natural z/VSE interface will read the SYSIN/SYSRDR/SYSIPT card input file until EOF, unless EOF had been encountered by Natural; this means that when driving the batch mode Natural session completely with STACK data, an extra “/” has to be provided in the JCL for a CMSYNIN/CMOBBIN null file. This is the default.
FLUSH=NO	No extra SYSIN/SYSRDR/SYSIPT card input (null) file is required, if the batch mode Natural session is completely driven with STACK data; the SYSIN/SYSRDR/SYSIPT card input file is then left as is, thus potentially resulting in INVALID STATEMENT operator prompts or job cancellation due to INVALID STATEMENT, when the Natural CMSYNIN/CMOBBIN had not been retrieved completely.

IDUMP - Dump Creation Mode

IDUMP specifies the kind of dump the Natural z/VSE interface is to produce.

Possible values:

IDUMP=YES	The Natural z/VSE interface will create dumps using the IDUMP macro. This is the default value.
IDUMP=NO	The Natural z/VSE interface will create dumps using the SDUMP macro.

LE370 - Use of IBM Language Environment

LE370 specifies whether Natural is to run in the IBM Language Environment. Possible values:

LE370=YES	The IBM Language Environment runtime environment is initialized on the initialization of the Natural session. You must specify "YES" if IBM Language Environment subroutine programs (dynamic or static) are to be called via Natural.
LE370=NO	The IBM Language Environment runtime environment is not initialized on the initialization of the Natural session. This is the default value.

LIBRID - Check of DLBL File ID Information

LIBRID specifies a string of up to 8 characters which is checked against the start of a DLBL file ID. If it matches, the remaining portion of that file ID is scanned for information specifying a library member in a z/VSE library or library chain. Possible values:

LIBRID=xxxxxxxx	xxxxxxxx = any character string of 8 characters length; must be enclosed in quotes if it contains any special characters.
LIBRID='LIBR: '	This is the default value.

MAXABND - Maximum Number of Abends

MAXABND specifies the maximum number of abends which NATVSE tolerates (that is, NATVSE intercepts the abend and invokes the Natural abend handler) until it assumes an unrecoverable abend situation or abend loop and terminates the Natural session abnormally by itself. Possible values:

MAXABND=nnnn	nnnn = maximum number of abends.
MAXABND=16	This is the default value.

RCSIZE - Default Roll Cache Size for a Server Environment

RCSIZE specifies the default roll cache size for a server environment for the case that the roll cache size is *not* passed with the Initialize Environment request. Possible values:

RCSIZE=nnnnnnnnn	nnnnnnnnn = default roll cache size in KB.
RCSIZE=0	This is the default value.

RJEUSER - User ID for Submission via XPCC Macro Requests

RJEUSER defines which user ID is to be set for submission via XPCC macro requests. Possible values:

RJEUSER=YES or RJEUSER=(YES, VSE)	RJEUSER=YES is the default value. The system variable *INIT-USER is used as the mandatory submission user ID.
RJEUSER=(YES, NAT)	The system variable *USER is used as the mandatory submission user ID.
RJEUSER=NO	The user ID R000 is used.

SEGMENT - Behavior at Output Spool File Close

SEGMENT specifies how the Natural z/VSE interface is to behave at CLOSE of an output SPOOL file (print or punch). Possible values:

SEGMENT=YES	A file close is accompanied by a POWER segment close unless CLOSE=FIN is in effect for that file.
SEGMENT=NO	The SPOOL file is closed without closing the POWER segment. This is the default value.

THDSIZE - Default Thread Size for a Server Natural Environment

THDSIZE specifies the default thread size for a server Natural environment for the case, that the thread size is *not* passed with the Initialize Environment request. Possible values:

THDSIZE=nnnnnnnnn	nnnnnnnnn = default thread size in KB.
THDSIZE=0	This is the default value.

USERID - Content of System Variable *INIT-USER

This parameter specifies the contents of the system variable *INIT-USER. Possible values:

USERID=YES	The following logic applies: if a z/VSE user ID is specified in JCL (// ID USER=xxx), this user ID is taken; otherwise, if a POWER from-user is specified in JECL (* \$\$ JOB FROM=xxx), this user id is taken; otherwise, the VSE job name is taken for the Natural user ID.
USERID=NO	The VSE job name is taken for the Natural user ID. This is the default value.

WAITIME - Time Limit for Session Roll-Out

WAITIME specifies a time limit in milliseconds. It applies to the CMROLL call in a Natural server environment: if the time interval passed in the CMROLL call is not less than the WAITIME interval, the session is rolled-out and the its thread is released, while the session is waiting. Possible values:

WAITIME=nnnnn	nnnnn = time limit in milliseconds.
WAITIME=1000	This is the default value.

Natural Datasets Used under a z/VSE Batch Mode Session

The following datasets are required if certain functions are used during a Natural z/VSE batch mode session:

Dataset	Explanation
CMEDIT	Software AG Editor Work File
CMHCOPY	Hardcopy Print Output
CMOBJIN	Input for Natural INPUT Statements
CMPLDG	Dynamic Profile Parameter Report Output
CMPRINT	Primary Report Output
CMPRMIN	Dynamic Profile Parameter Input
CMPTnn	Additional Reports 01-31
CMSYNIN	Primary Command Input
CMTRACE	External Trace Output
CMWKFnn	Work Files 01-32

These datasets are described below.

CMEDIT - Software AG Editor Work File

The **Software AG Editor** work file VSAM dataset 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*.

CMHCOPY - Optional Report Output for Hardcopy

The default name of the hardcopy print output dataset 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 dataset. The default dataset name `CMHCOPY` implies `CLOSE=FIN` for the hardcopy print dataset, that is, after the dataset has been opened for output, any subsequent change of the hardcopy print output dataset name will not be honored. If a different name is defined at open time, the hardcopy dataset 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`.

CMOBBJIN - Input for Natural INPUT Statements

CMOBBJIN 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 CMOBBJIN. 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 CMOBBJIN label information.

By default, the CMOBBJIN 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 CMOBBJIN.

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 CMOBBJIN.

CMPLOG - Optional Report Output for Dynamic Parameters

If profile parameter PLOG=ON is set and dataset CMPLOG is available, the evaluated dynamic profile parameters are written to this dataset during session initialization. If dataset 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 Dataset

CMPRMIN can be used as a dynamic parameter dataset 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 dataset, i.e. the same rules apply.

CMPRMIN is a sequential disk or a labelled tape dataset. 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
```

CMPRTnn - Additional Reports

CMPRT 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 CMPRT 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 CMPRT 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 dataset. 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 Natural z/VSE generation parameter `FILMNGR=YES` is specified 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 datasets. 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 `LIBRID` in *NTVSE Generation Parameters*) at the start of the file ID field in the `DLBL` statement signals that the Natural dataset actually is a z/VSE library member which is specified in the remainder of the file ID field.

The following specifications are possible:

<code>C=chain</code>	Specifies a library concatenation chain defined in JCL.
<code>S=library.sublib</code>	Specifies a specific sublibrary in a specific library.
<code>M=mbrname.mbrtype</code>	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 dataset 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 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
CAT= <i>catalog</i>	Triggers the usage for VSAM/SAM for the dynamic work file, where catalog 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 (CAT=IJSYSUC).
VOL= <i>volser</i>	If specified a // EXTENT information is generated with that 1 to 5 characters volume serial number; its content depends on the preceding DLBL information, see below.
DSN= <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 xxyyyyz,'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:

```
xxyyyyz
```

where:

- xxyyyy* 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 MORE 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,POOL01,,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 session parameter (`UPSI=1XXXXXX`, for example). This is useful if `UPSI` settings in 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 XXX1XXXX	Storage Freeze Flag
UPSI XXXXX1XX	Session Abend Flag
UPSI XXXXXX1X	Abend Exit Flag
UPSI XXXXXX1	Formatted Dump-Only 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 X1XXXXXX - 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 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 XXXXX1XX - Session Abend Flag

By default, a Natural session is cancelled if crucial errors have occurred (`NAT9nnn` termination messages except `NAT9995` and `NAT9987`). When this flag is set, Natural does *not* cancel, but terminates “normally” just passing the Natural return code to the job control.

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.

UPSI XXXXXX1 - Formatted Dump-Only Flag

With `DU=ON`, the NATVSE abend exit routine creates a snap dump of the Natural session when a program check abend occurs (and the `UPSI XXXXXX1X` flag is not set).

- The failed instruction, the program check code (`S0Cn`), the general registers, the currently active routine, the offset of the failed instruction within this routine and the absolute (PSW) address are displayed together with Registers 0 to 15 of the currently assigned save area (CSA).
- In addition, the non-reentrant Natural z/VSE driver, all areas `GETMAINED` by Natural and all Natural programs in the buffer pool are dumped.
- Then control is passed to Natural for a clean session termination.
- Finally the job terminates via a z/VSE `JDUMP` macro resulting in a dump containing the whole partition.

Since in many cases the dynamic Natural session areas are relevant for debugging only, the dump of the static session areas can be suppressed by setting this `UPSI` flag.

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 sampljob
// 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,'...',... labelled 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 labelled tape
// ASSGN SYSnnn,uuu 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`.

40 Natural in Batch Mode under CMS

- Running Natural in Batch Mode under CMS 246

This document contains special considerations that refer to Natural in batch mode under the operating system CMS.

For considerations that refer to Natural in batch mode generally, see:

- [Adabas Datasets](#)
- [Sort Datasets](#)

Running Natural in Batch Mode under CMS

If you invoke Natural with the dynamic parameter `BATCH`, batch mode is entered.

Natural does not communicate with the terminal, but takes its input from the dataset whose DD name is `CMSYNIN` and sends its output to the dataset whose DD name is `CMPRINT`. These datasets are described below.

`FILEDEF` commands for these DD names must be issued before invoking Natural in batch mode.

If `BATCH` is specified in combination with other dynamic parameters, `BATCH` must be the first parameter as shown in the example below:

```
FILEDEF CMPRINT PRINTER
FILEDEF CMSYNIN DISK BATCH INPUT A
NATvr BATCH, FNAT=(10,13), FUSER=(132,12)
```

In this example, *vr* stands for the current version and release number.

For more examples, see `NATBATCH EXEC` and `NATINPL EXEC`.

CMPRINT - Primary Report Output

`CMPRINT` is used for the primary output report resulting from `DISPLAY`, `PRINT` and `WRITE` statements in a Natural program.

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.

The number of characters actually processed per line is determined by the current setting of the profile 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.

41 Natural in Batch Mode under BS2000/OSD

- Files and System Files Used by Natural in BS2000/OSD Batch Mode 248
- Keyword Parameters 250
- BS2000/OSD Job Variables 259

This document contains special considerations that refer to Natural in batch mode under the operating system BS2000/OSD.

The following topics are covered:

See also *Natural under BS2000/OSD Batch Mode Error Messages*.

For considerations that refer to Natural in batch mode generally, see:

- [Adabas Datasets](#)
- [Sort Datasets](#)
- [Subtasking Session Support for Batch Mode Environments](#)

Files and System Files Used by Natural in BS2000/OSD Batch Mode

The following optional sequential files or system files are used during a Natural under BS2000/OSD batch mode session:

Link Name	System File	Explanation
CMPRMIN		Dynamic Parameter Dataset
	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
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 - SYLST99` 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/OSD 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` | `TRACE` | `USERID` | `WRITE`

ADACOM

Possible values:	ADACOM=ADAUSER, ADACOM=ADABAS, ADACOM=ADALNK
Default value:	ADACOM=ADALNK

This parameter applies to the generation of the front-end part. It determines which Adabas link module is to be used. Possible values:

ADACOM=ADAUSER	The module ADAUSER is linked to the front-end part (Adabas versions lower than 7.1).
ADACOM=ADABAS	The modules ADAUSER and SSFB2C are linked to the front-end part (Adabas Version 7.1 and higher).
ADACOM=ADALNK	The module ADALNK is linked to the front-end part (Adabas versions lower than 7.1) or the modules ADALNK, ADAL2P and SSFB2C are linked to the front-end part (Adabas Version 7.1 and higher).

ADDBUFF

Possible values:	1 to 8
Default value:	None

This parameter applies to the generation of the front-end part.

It determines the additional number of pages (4 KB units) for the terminal I/O buffer.

APPLNAM

Possible values:	<i>name</i>
Default value:	NATBS2

This parameter applies to the generation of the front-end part.

name 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.

CODE

Possible values:	FRONT/RENT
Default value:	FRONT

This parameter applies to the generation of both the front-end and reentrant parts.

It determines which part of the Natural BS2000/OSD interface is to be generated.

CODE=FRONT	indicates the generation/assembly of the front-end part.
CODE=RENT	indicates the generation/assembly of the reentrant part.

DELETE

Possible values:	ON/OFF
Default value:	ON

This parameter applies to the generation of the reentrant part.

DELETE=ON	The setting of the profile parameter DELETE in the Natural parameter 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.
DELETE=OFF	A dynamically loaded non-Natural program once loaded is kept for the duration of the whole Natural session.

DYNPAR

Possible values:	SYSDTA/SYSIPT/FILE/NO
Default value:	NO

This parameter applies to the generation of the front-end part.

DYNPAR=NO	No dynamic parameters are read.
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

Possible values:	YES/NO/CRTE
Default value:	NO

This parameter applies to the generation of the reentrant part.

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 front-end:</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 front-end:</p> <pre> INCLUDE ITOINITS,SYSLNK.ILCS RESOLVE,SYSLNK.ILCS </pre>
ILCS=NO	Standard processing applies.

JV

Possible values:	ON/OFF
Default value:	OFF

This parameter applies to the generation of the front-end part.

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/OSD installation does not include the Siemens product "Job Variables," this parameter must be set to OFF; otherwise assembly errors in the NAMBS2 compilation occur.

LF

Possible values:	X'zz'
Default value:	X'25'

This parameter applies to the generation of the front-end part.

With this parameter, you specify the control character to be used for line advance when printing on the local printer.

LINK

Possible values:	<i>name</i> (<i>name,name,...</i>)
Default value:	none

This parameter applies to the generation of the front-end part.

The *name(s)* 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.

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

Possible values:	<i>name</i> (<i>name,name,...</i>)
Default value:	none

These parameters apply to the generation of the front-end 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. The syntax is analogous to that of LINK.

Examples:

```
NAMBS2 LINK=(PROG1,PROG2,...),
LINK2=(PROG54,...)
NAMBS2 LINK=(PROG1,PROG2,PROG3,PROG4)
```

NUCNAME

Possible values:	<i>name</i>
Default value:	NB2RENT

This parameter applies to the generation of the front-end 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).

PARMOD

Possible values:	<i>(nn,loc)</i> <i>nn</i> : 24/31 <i>loc</i> : BELOW/ABOVE
Default values:	(31,ABOVE)

This parameter applies to the generation of both the front-end and reentrant parts.

The first part of this parameter (*nn*) is used to define an addressing mode (24-bit or 31-bit mode) for the Natural BS2000/OSD application.

31-bit mode is required if the Natural buffer pool, the reentrant part of the Natural BS2000/OSD 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 front-end part location of the Natural BS2000/OSD application. If you load the front-end part of the application above 16 MB, this must be defined in the front-end part's link procedure as follows:

```
LOADPT=*XS
```

or

```
LOADPT=X 'address'
```

Example:

```
/EXEC TSOLINK
PROG NAT230, FILENAM=NAT230, LOADPT=*XS, ...
TRAITS RMODE=ANY, AMODE=31
INCLUDE ...
/* PARMOD=(nn,loc) MUST BE IDENTICAL IN THE FRONT-END AND REENTRANT PARTS
```

REQMLOC

Possible values:	RES/BELOW/ABOVE
Default value:	RES

This parameter applies to the generation of both the front-end and reentrant parts 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.

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.

The REQMLOC parameter corresponds to the LOC parameter of the BS2000/OSD system macro REQM.

SYSDTA

Possible values:	PRIMARY/SYSCMD
Default value:	PRIMARY

This parameter applies for the generation of the front-end part.

SYSDTA=PRIMARY	After reading of dynamic parameters from SYSDTA, SYSDTA is set to SYSFILE SYSDTA=(PRIMARY).
SYSDTA=SYSCMD	After reading of dynamic parameters from SYSDTA, SYSDTA is set to SYSFILE SYSDTA=(SYSCMD).

TERM

Possible values:	PRGR/STEP
Default value:	PRGR

This parameter applies to the generation of the front-end part.

TERM=PRGR	The Natural batch application will be terminated.
TERM=STEP	The system additionally executes the next SET-JOB-STEP command.

TRACE

Possible values:	<i>nn, ll</i> <i>nn: 01 to 99</i> <i>ll: 71 to 132</i>
Default value:	99,71

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. *nn* is the number for the *SYSLSTnn* trace file and *ll* is the maximal length in characters of a trace print record.

If any external Natural trace function is active, the trace records will be written to *SYSLSTnn*. 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

Possible values:	YES/SYSTEM/NO/USER
Default value:	USER

This parameter applies to the generation of the front-end part.

USERID=SYSTEM or USERID=YES	The Natural user ID is created by using the BS2000/OSD 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/OSD job name has been specified with the LOGON command, the Natural user ID is created as with USERID=SYSTEM or YES.

WRITE

Possible values:	SYSOUT/SYSLST
Default value:	SYSLST

This parameter applies to the generation of both the front-end and reentrant parts.

This parameter controls whether output produced by Natural is written to *SYSOUT* or *SYSLST*.

BS2000/OSD Job Variables

The Natural batch mode driver uses the BS2000/OSD 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/OSD 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
```


42 Natural in Batch Mode (All Environments)

- Adabas Datasets 262
- Sort Datasets 262
- Subtasking Session Support for Batch Mode Environments 262

This document contains general considerations that apply when running Natural in batch mode.

The following topics are covered:

Adabas Datasets

Adabas datasets 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 datasets.

Sort Datasets

Sort datasets 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 datasets `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](#)



Note: With Natural for CMS, subtasking is not supported.

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 datasets (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 datasets can be undefined or dummy datasets; 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](#)
- [Startup Parameter Area](#)

- [User Parameter Area](#)

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 datasets (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 dataset 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 dataset name
CMOBJIN	Object input dataset name
CMPRINT	Standard output dataset name
CMPRMIN	Dynamic parameter input dataset name
CMPLLOG	Dynamic parameter output dataset name

CMTRACE	Trace output dataset 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 dataset 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 dataset means that this dataset 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 *SYSnnn* 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.

43

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/OSD.

- [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/OSD](#)

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 [Buffer Pools](#) in the section *Profile Parameters Grouped by Function*.

44 Natural Buffer Pool - General

- Natural Buffer Pool Principle of Operation 270
- Buffer-Pool Monitoring and Maintenance 275
- Natural Global Buffer Pool 278

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 and Global Buffer Pools](#)
- [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, TIAM and VM/CMS, 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 and Global Buffer Pools

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, TSO or CMS 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 VM/CMS environment, a global buffer pool can be installed as a writeable Discontiguous Shared Segment that is dynamically attached to the user's virtual machine when Natural/CMS is invoked; see also the sections *Installing Natural under CMS* and *Preparing the VM System for Natural* in the *Natural Installation* documentation.

In a BS2000/OSD environment, a global buffer pool is a common memory pool, see [Natural Global Buffer Pool under BS2000/OSD](#).

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 not available with z/VM. It is used only for Natural programming 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/OSD 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/OSD), 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 no 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` 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/OSD (refer to [Global Buffer Pool under BS2000/OSD](#)).

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.

45

Natural Global Buffer Pool under z/OS

▪ Using a Natural Global Buffer Pool	282
▪ Operating the Natural Global Buffer Pool	282
▪ Sample NATGBPvr Execution Jobs	284
▪ Localization	286

This document describes purpose and usage of a Natural Global Buffer Pool (GBP) under the operating system z/OS.

The following topics are covered:

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](#)

See also:

- [Natural Global Buffer Pool Manager Messages](#) in the *Natural Messages and Codes* documentation

Using a Natural Global Buffer Pool

Definition

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.

Operating the Natural Global Buffer Pool

The following topics are covered below:

- [Installing the Natural GBP Operating Program](#)
- [Setting up the Natural Global Buffer Pool](#)
- [Starting the Natural GBP Operating Program](#)

- [Stopping the Natural GBP Operating Program](#)

Installing the Natural GBP Operating Program



Note: *vr*s stands for version, release and system maintenance level of the product.

The global buffer pool is operated by the program `NATGBPvr` which must be executed from an authorized library.

During the installation of Natural, the modules `NATGBPvr` is linked into an APF-authorized library.

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 [Authorized Services Manager](#) is mandatory. The Authorized Services Manager 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 Global Buffer Pool

The functions available from `NATGBPvr` 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 `NATGBPvr` has not been terminated.

`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 `SYSDIN` 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 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, `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:

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 `vrs` or `vr` stands for the relevant version, release, system maintenance level numbers. For further 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:

```
//GBPSTR2
//* Read commands from SYSIN1:
//*
//* Start 3 global buffer pools (subsystem ID Nvrs) with name
//*   NATGBP1 - size=1024KB and a cache with size 2048KB
//*   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, send 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 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:

```
//SYSLIN DD *  
SETCODE AC(1)  
SETOPT PARM(REUS=RENT)  
INCLUDE NATLIB(NATGBPMG)  
INCLUDE SMALIB(NATGBPRM)  
INCLUDE SMALIB(NATGBPTX)  
INCLUDE NATLIB(NATBPMGR)  
NAME NATGBPvr(R)  
/*
```

46 Natural Global Buffer Pool under z/VSE

■ Using a Natural Global Buffer Pool	288
■ Operating the Natural Global Buffer Pool	289
■ Sample NATGBPvr Execution Jobs	290
■ Localization	292

This document describes purpose and usage of a Natural Global Buffer Pool (GBP) under the operating system z/VSE.

The following topics are covered:

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](#)

See also:

- [Natural Global Buffer Pool Manager Messages](#) in the *Natural Messages and Codes* documentation

Using a Natural Global Buffer Pool

Definition

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

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

Installing the Natural GBP Operating Program

No installation procedure required. The global buffer pool is operated by program NATGBP vr which is contained in and executed from the Natural load library.

Setting Up the Natural Global Buffer Pool

The functions available from NATGBP vr 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 MSG xx with xx being the z/VSE partition ID) unless NATGBP vr has not been terminated.

NATGBP vr 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=<dlbl-name>, where <dlbl-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 <dlbl-name> indicates to NATGBP vr 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 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, `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:

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* stands for the relevant version, release, system maintenance level numbers. 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 3 global buffer pools (subsystem id Nvrs) with name
/*   NATGBP1 - size=1024KB and a cache with size 2048KB
/*   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 sleeping job by MSG partition-id
/* and enter 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
/*
```

47 Common Natural GBP Operating Functions under z/OS and z/VSE

- Global Buffer Pool Manager Parameter Module 294
- Global Buffer Pool Operating Functions 294
- Global Buffer Pool Function Parameters 296
- Examples of NATBUFFER Specifications 301

This document provides a summary of those operating functions of the Natural global buffer pool which are identical under z/OS and z/VSE.

The following topics are covered:

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.

DELCACHE - 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 dataset. 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
- **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

This parameter is mandatory (except for the **TERMINATE** function). It specifies the name of the global buffer pool to be created.

BPNAME = <i>name</i>	<i>name</i> is the 8-byte name of the global buffer pool. If the specified name is shorter than 8 bytes, blanks will be appended to it.
-----------------------------	---

For the functions **DELCACHE** and **FSHUT**, you may supply a value of "*" to process all buffer pools for the specified Natural subsystem.

BPLIST - Name of Preload List

This parameter specifies the name of the preload list.

BPLIST = <i>name</i>	<i>name</i> is the 8-byte name of the preload list. If the specified name is shorter than 8 bytes, blanks will be appended to it.
-----------------------------	---

BPCSIZE - Buffer Pool Cache Size

This parameter specifies the amount of storage (in KB) used to allocate a data space for the buffer pool cache.

BPCSIZE = <i>size</i>	<p><i>size</i> is the amount of storage (in KB) used to allocate a data space for the buffer pool cache. The valid range is 100 - 2097148.</p> <p>The cache size can also be specified in units of MB or GB, e.g. by specifying 10M for 10 MB.</p> <p>If the BPCSIZE parameter is omitted (or set to zero), the buffer pool is not supplied with a cache.</p> <p>Note: A cache is only supported for buffer pools of TYPE=NAT.</p>
------------------------------	---

CONFIRM - FSHUT Confirmation

This parameter controls the **FSHUT** behavior if there are still active objects in the buffer pool.

CONFIRM=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.
CONFIRM=N	FSHUT is forced without interaction with the operator.

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

This parameter is ignored when the task does not own a buffer pool cache.

IDLE=nn	<i>nn</i> is 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. The default setting is 60 seconds.
---------	--

IDLE is a “global” parameter, i.e. once specified, **IDLE** will also apply to subsequent commands, without your having to specify it again.

 **Note:** 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

This parameter controls which algorithm is to be used for allocating storage in the Natural buffer pool.

METHOD=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.
METHOD=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.

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

This parameter specifies the size and the mode of the buffer pool, and the text block size.

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 is 4.</p>
<i>size, mode</i> and <i>tsize</i> have to be specified in the sequence shown above.		

If NATBUFFER is not specified, the default values will be used. See also [Examples of NATBUFFER Specifications](#) below.

RESIDENT - Behavior after Function Execution

This parameter specifies the behavior of the GBP operating program after the specified function has been executed. The following values are possible:

RESIDENT=Y	The GBP operating program will remain active after executing the specified function and await further commands. Once specified, RESIDENT=Y will also apply to subsequent commands, without your having to specify it again. (To stop the GBP operating program, you use the TERMINATE function.)
RESIDENT=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, RESIDENT=N is ignored and the task is not terminated.
RESIDENT=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, RESIDENT=A works in the same way as RESIDENT=N. When the task owns a buffer pool cache, RESIDENT=A works the same way as RESIDENT=Y, but switches automatically to RESIDENT=N, 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>

RESIDENT is a “global” parameter, i.e. once specified, RESIDENT will also apply to subsequent commands until explicitly specified/overwritten.

SUBSID - Natural Subsystem ID

This parameter specifies the ID of the Natural subsystem.

SUBSID= <i>id</i>	<p><i>id</i> is the 4-byte ID of the Natural subsystem.</p> <p>Once specified, SUBSID will also apply to subsequent commands, without your having to specify it again.</p> <p>The default value is NAT<i>v</i>, where <i>v</i> is the first digit of the current Natural version.</p>
-------------------	---

SUBSID is a “global” parameter, that is, once specified, SUBSID will also apply to subsequent commands until explicitly specified/overwritten.

For the functions [DELCACHE](#), [FSHUT](#) and [SHOWBP](#), you may supply a value of "*" to process all buffer pools for the specified Natural subsystem.

For further information on the Natural subsystem, see [Natural Subsystem \(z/OS\)](#) or [Natural Subsystem \(z/VSE\)](#).

TYPE - Type of Buffer Pool

This parameter specifies the type of the buffer pool. Possible values are:

TYPE=NAT	Natural buffer pool (this is the default).
TYPE=SORT	Sort buffer pool.
TYPE=DLI	DL/I buffer pool.
TYPE=EDIT	Editor buffer pool.
TYPE=MON	Monitor buffer pool.
TYPE=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.

48

Natural Global Buffer Pool under BS2000/OSD

- Using a Natural Global Buffer Pool under BS2000/OSD 304
- Establishing the Global Buffer Pool under BS2000/OSD 304
- Administering the Global Buffer Pool under BS2000/OSD 305

This document describes purpose and usage of a Natural Global Buffer Pool (GBP) under the operating system BS2000/OSD.

In the examples below, the notation *vrs* or *vr* stands for the relevant version, release, system maintenance level numbers. For further information on product versions, see Version in the *Glossary*.

The following topics are covered:

Using a Natural Global Buffer Pool under BS2000/OSD

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 UTM 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/OSD

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/OSD 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/OSD

Once global buffer pool is active, it is administered via the operator console.

The following BS2000/OSD console commands are available (where *tsn* is the TSN of the buffer pool task):

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.

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.

So that you can terminate global buffer pools via a program, too, the program CMPEND is provided:

Example:

```

/PROC C
/SYSFILE SYSDTA=(SYSCMD)
/EXEC (CMPEND,NATvrs.MOD)
name /* name of the global buffer pool
/SYSFILE SYSDTA=(PRIMARY)
/ENDP

```


49 Natural Swap Pool

This part provides information on the Natural swap pool which is available when you are using one of the following TP monitors:

- **CICS** (where the Natural swap pool is optional)
- **UTM** (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 UTM**
- **Terminating the Global Swap Pool**

Related Topics:

- *Natural Swap Pool under CICS* in the *Natural TP Monitor Interfaces* documentation
- *Using the Natural Swap Pool under CICS* in the *Natural TP Monitor Interfaces* documentation

- *Natural Swap Pool under UTM in the Natural TP Monitor Interfaces* documentation
- *Error Messages from the Natural Swap Pool Manager Valid under CICS and UTM in the Messages and Codes* documentation

50 Purpose of a Natural Swap Pool

- Purpose of a Natural Swap Pool 310
- Benefits of Using a Natural Swap Pool 310
- Swap Pool Structure 311

This document describes the purpose, benefits and structure of a Natural swap pool.

The following topics are covered:

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 <code>NATPARM</code> , that is, the <code>(NT)BPI</code> settings in effect for the Natural session initializing the Natural CICS environment are taken.
UTM	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.
UTM	The Natural swap pool is necessary under UTM. See also <i>Natural Swap Pool under UTM</i>

51 Natural Swap Pool Operation

- Users are On their Way to Natural - No Session Start 314
- Users are Returning from Natural 314

The following situations are explained:

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.

52 Natural Swap Pool Initialization

- Swap Pool Initialization Control 318
- Swap Pool Initialization Parameters 319

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`.

The following topics are covered:

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 UTM:**
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 `NATPARM`, that is, the `(NT)BPI` settings in effect for the Natural session initializing the Natural CICS environment are taken.

53

Dynamic Swap-Pool Reorganization

- Requirements for Dynamic Swap-Pool Reorganization 322
- Statistics Tables 322
- Swap-Pool-Reorganization Plus Table 322
- Swap-Pool-Reorganization Minus Table 323
- Parameters for Swap-Pool Reorganization 323
- Checking for the Necessity of Swap-Pool Reorganization 324
- Flow of Dynamic Swap-Pool Reorganization 324
- Start of Dynamic Swap-Pool Reorganization 325

This document describes the prerequisites, process, control and start of a dynamic swap pool reorganization.

The following topics are covered:

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.

54 Defining the Natural Swap Pool

- Environment-Specific Requirements 328
- Keyword Parameters of Macro NTSWPRM 328

This document describes the TP monitor environment-specific requirements that apply and the keyword parameters can be used to define the Natural swap pool.

The following topics are covered:

Environment-Specific Requirements

The following environment-specific requirements apply:

■ **Under UTM:**

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.

Keyword Parameters of Macro `NTSWPRM`

The following keyword parameters can be used to define the Natural swap pool details:

`LABEL` | `DSPCONT` | `DSPLIFE` | `SWPFILE` | `MAXSIZE` | `SWPFACT` | `SWPINIT` | `SWPLSWP` | `SWPPWRD` | `SWPSDIF` | `SWPSLSZ` | `SWPTFIX` | `SWPTIM1` | `SWPTIM2` | `SWPUSER` | `NOVPA` | `NOVPW` | `WAITMS` | `WRITMS` |

LABEL - Name of Swap-pool Parameter Module

This parameter defines the CSECT name of the swap-pool parameter module.

<code>LABEL=nnnnnnnn</code>	The name <code>nnnnnnnn</code> may be 8 characters at maximum.
<code>LABEL=NATSWPRM</code>	The default setting is the name of macro <code>NTSWPRM</code> .

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.

DSPCONT= <i>nnn</i>	<i>nnn</i> must be in the range of 1 to 480.
DSPCONT=10	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 of 1 to 60.
DSPCONT=5	The default value is 5 (minutes).

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.

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 of 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.

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= n	Possible values for n are 0 to 4. n determines the size of a “unit” as follows: 0 corresponds to 2 KB. 1 corresponds to 4 KB. 2 corresponds to 8 KB. 3 corresponds to 16 KB. 4 corresponds to 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](#).

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 of 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. Possible values 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 of 20 to 998.
	<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 of 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
    
```

This parameter will be ignored if the swap-pool initialization data could be read from the Natural system file.

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.

<code>SWPTIM2=nn</code>	<code>nn</code> must be in the range from 1 to 99 (minutes)
<code>SWPTIM2=2</code>	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.

<code>SWPUSER=nn</code>	<code>nn</code> must be in the range from 1 to 99.
<code>SWPUSER=20</code>	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.

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 of 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 of 1 to 999.
NOVPW=15	The default value is 15 (waits).

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 of 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 of 1 to 999.
WRITMS=10	The default value is 10 (milliseconds).

55

Natural User Area Size Considerations

▪ Using the MAXSIZE Parameter	336
▪ Defining the Size of the Individual Natural Buffers	336
▪ Possible Error Messages	336
▪ Displaying the Aggregate Size of All Buffers	337
▪ Calculating the Maximum Size	337

This document describes how to manage the size of the Natural user area and the size of the individual Natural buffers.

The following topics are covered:

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`.

56

Swap Pool Data Space

- Using ESA Data Space in Addition 340
- ESA Data Space Slot Size Adjustment 340

This document describes how to extend the Natural Swap Pool capacity by generating ESA Data Space.

The following topics are covered:

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 `UTM`, you can define the slot size by setting the `NATUTM` macro keyword parameter `ROLLTSZ` adequately.
- If you are using the TP monitor `CICS`, the Data Space slot size will automatically take the size of the longest thread.

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 `NATPARM`, that is, the `(NT)BPI` settings in effect for the Natural session initializing the Natural `CICS` environment are taken.

57

Global Restartable Swap Pool under UTM

- Purpose of a Natural Global Swap Pool under UTM 342
- Installing a Natural Global Swap Pool under UTM 342
- Starting a Natural Global Swap Pool under UTM 343
- Displaying Information about the Global Swap Pool 343

This document describes how to install and operate a Natural global swap pool in a Natural under UTM environment.

The following topics are covered:

Purpose of a Natural Global Swap Pool under UTM

If all tasks of a Natural under UTM 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 UTM 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 UTM

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 UTM 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 UTM 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 UTM 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 UTM 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 UTM

A Natural global swap pool must be started with program `CMPSTART`. It can be used from a maximum of five Natural under UTM 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:

```
/INTR tsn,DPR
```


58

Terminating the Global Swap Pool under UTM

■ Termination Using Console Commands	346
■ Abnormal Termination with Dump	346
■ Termination by Program	347

This document describes the ways in which a Natural global swap pool can be terminated under UTM.

The following topics are covered:



Caution: Before the swap pool is terminated, the Natural under UTM application that uses it must be terminated.

Termination Using Console Commands

▶ To terminate the global swap pool normally

- 1 Issue the console command:

```
/INTRtsn,STOP
```

- 2 or issue the console command:

```
/INTRtsn,END
```

Abnormal Termination with Dump

▶ To terminate the global swap pool abnormally, producing a dump

- Issue the console command:

```
/INTRtsn,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
```


59

Natural 3GL CALLNAT Interface

This part contains information about the Natural 3GL CALLNAT Interface with which Natural enables 3GL programs to invoke and execute Natural subprograms.

- [Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions](#)
- [Natural 3GL CALLNAT Interface - Usage, Examples](#)

60 Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions

▪ Purpose of 3GL CALLNAT Interface	352
▪ Prerequisites	353
▪ Restrictions	354

This document describes the purpose of the 3GL CALLNAT interface and its prerequisites and restrictions.

The following topics are covered:

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.

Under CMS, Natural provides the function `CALLNAT` to execute Natural subprograms from a Rexx program. For details, refer to [Natural under VM/CMS](#).

Availability

The interface is available in batch mode under the following operating systems:

- z/OS,
- z/VSE,
- z/VM,
- BS2000/OSD;

and for the following TP-monitor environments:

- CICS,
- Com-plete,
- IMS TM,
- TIAM,
- TSO,
- 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

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. In most cases, it is necessary to link the 3GL program to the interface module (for example, `NATXCAL`; see below).

The samples `XNATGC2` 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:

NATXCAL	To be used if the 3GL module is either linked to Natural or loaded dynamically and then invoked by a branch and link instruction (batch, CMS, Com-plete, IMS TM, TIAM, TSO, UTM).
NATXCAL4	To be used if the 3GL module is called via the <code>INTERFACE4</code> option of the <code>CALL</code> statement. It provides the <code>INTERFACE4</code> Natural Callnat Interface as well as the <code>INTERFACE4</code> Callback Functions. For further information on the <code>INTERFACE4</code> functionality, see the <code>CALL</code> statement documentation
NCIXCALL	To be used in a CICS environment if the 3GL module has been invoked using <code>EXEC CICS LINK</code> ; <code>NCIXCALL</code> is delivered in source code to be compiled with your CICS macros. See also <i>Installing the Natural CICS Interface</i> in the Natural Installation documentation.
NCIXCPRM	To be used in a CICS environment to build the parameter address list used as <code>COMMAREA</code> for the subsequent <code>EXEC CICS LINK</code> command.

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.

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 UTM

IMS TM and UTM 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).

61 Natural 3GL CALLNAT Interface - Usage, Examples

- Usage 358
- Sample Environments 361

This section describes the usage of the 3GL CALLNAT interface and describes a number of sample 3GL CALLNAT environments.

The following topics are covered:

Usage

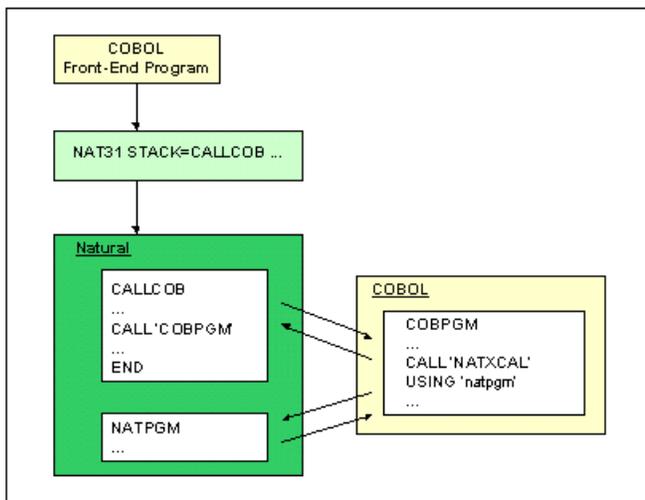
The following topics are covered:

- [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 *natpgm* 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 are not possible.

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 `NCvrn` 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 CICSLIB(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

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

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`.

62

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* in the *Natural Utilities* documentation
- *Installing the Software AG Editor* in the *Natural Installation* documentation
- *Software AG Editor* in the *Natural Editors* documentation

63 Editor Work File

- Editor Work File Structure 368
- Editor Work File under z/OS, z/VSE and BS2000/OSD 369
- Using the Batch Format Utility 370
- Formatting during Initialization 370
- Maintaining the Editor Work File under z/OS and z/VSE 370
- Maintaining the Editor Work File under BS2000/OSD 371
- Editor Work File under VM/CMS 372
- Editor Work File under Complete/SMARTS 372

This document describes structure, use and maintenance of the editor work file under the various operating systems. The following topics are covered:

See also:

- *SYSEDT Utility* in the *Natural Utilities* documentation
- *Installing the Software AG Editor* in the *Natural Installation* documentation.
- *EDBP - SAG Editor Buffer Pool Definitions* in the *Natural Parameter Reference* documentation
- *Software AG Editor* in the *Natural Editors* documentation

Editor Work File Structure

The editor work file is a relative record dataset with fixed length records. It is divided into three parts:

- Control Record
- Work Record
- 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.

You can modify the control record by using the **Generation Parameters** function of the *SYSEDT Utility*.

For buffer pool warm restarts, the parameters are read from the control record.

Work Record

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/OSD

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 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 datasets 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/OSD; default size is 4096).

The total number of editor work file records depends on the allocated dataset space for the editor work file.

There are two alternative ways of formatting the editor work file:

- offline by using the batch format utility,
- online during buffer pool initialization.

Using the Batch Format 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 batch format utility to specify editor buffer pool parameters by means of 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 Editor Work File Batch 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. Run the Editor Work File Batch Utility with the new file after having added:

```
PARM=COPY
```

in the EXEC JCL card and a

```
//CMCOPY DD...    under z/OS or  
//DLBL CMCOPY...  under z/VSE
```

card for the renamed editor work file `CMCOPY` to be copied into the new work file `CMEDIT`.

7. Check the job log for potential errors.
8. Reallocate and (if necessary) reopen the editor work file.
9. Use the `SYSED` utility (see the Natural *Utilities* documentation) to check if 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/OSD

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 (see also *Installing the Software AG Editor* in the Natural *Installation* documentation) 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
```

Editor Work File under VM/CMS

One editor work file corresponds to one VM/CMS user. A work file is created during the first editor initialization and has the default name CMEDIT DATA A1.

It must be large enough to contain the editor sessions of the VM/CMS user. Name, size and number of work file records are specified in the editor buffer pool parameter EDBP or by macro NTEDBP in the Natural parameter module NATPARM. For optimum performance, a multiple of 800 is recommended.

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.

64 Editor Buffer Pool

- Purpose of the Editor Buffer Pool 374
- Obtaining Free Blocks 375
- Initializing the Editor Buffer Pool 375
- Restarting the Editor Buffer Pool 376
- Editor Buffer Pool Parameters 376
- Buffer Pool Initialization for Multi-User Environments 376

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.

The following topics are covered:

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 an object (for example, a file member) 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/OSD only) or an auxiliary (*EDPSIZE*) buffer pool. In multi-user environments (CICS, IMS TM, UTM), the editor buffer pool is shared by all editor users of either the same region (local pool) or more than one region (global pool). Under CMS, the buffer pool is always a local one. A global buffer pool cannot be shared by Com-plete and other regions due to the separate SD editor work file under Com-plete.

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 dataset, 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.

Restarting the Editor Buffer Pool

The `SYSEDT` utility 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 `NATPARM` 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`.

65

Natural Net Data Interface NATNETTO

▪ Natural Net Data Driver Functional Description	380
▪ General Message Layout	381
▪ Layout of Header	381
▪ Format Buffer Layout	385
▪ Value Buffer Layout	390
▪ Attribute Buffer	391

This document provides information on the Natural Net Data Interface and the `net data` protocol definition.

See also *Installing the Natural Net Data Interface* in the *Installation* 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 UTM.

"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	Eyecatcher
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 (normaly 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 aplha 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 99999999999999999999

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
  0000001234.5678AAAAAABBBBBCCCCDDDDZ99999999999999999999
                                FMA20/08100024FOA20/
38102024FMA15/08102024FOA20/38101624FOA20/38102024FOA79/70000035.000000

```

Figure 2: Net-Data Stream Generated from NETM002 Execution

Where NATvrs stands for *version, release, system maintenance level* of the current Natural version.

Configuration Settings: Fixed format, format buffer + extended format buffer, attribute buffer option, cursor position represented as field number.

66

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.

67 Natural as a Server under z/OS

- Functionality 396
- Natural Nucleus Installation in a Server Environment 397
- Print and Work File Handling with External Datasets in a Server Environment 397

This document applies under z/OS only. It covers the following topics:

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 Remote Procedure Call (RPC)* 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 frontend.

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 Remote Procedure Call (RPC)* 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 `NATPARM` 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 Datasets 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 `NTWORK` 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 dataset 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 dataset 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 workfiles 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 dataset 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 dataset name, or
- to define an output spool class.

If a DD name is specified, the access method checks whether the dataset is allocated. If not, an error is issued. The dataset can be allocated by any Natural program using the `USR2021N` subprogram supplied in library `SYSEXT`.

If a physical dataset name or a spool file class is specified, the access method itself allocates the dataset dynamically during the execution of the `DEFINE . . .` statement. To ensure a unique DD name is used, `DEST=*` should be predefined in the `NATPARM` file. 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.

68 Natural as a Server under z/VSE

▪ Functionality	400
▪ Natural Nucleus Installation in a Server Environment	401
▪ Print and Work File Handling with External Datasets in a Server Environment	401

This document applies under z/VSE only. It covers the following topics:

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 Remote Procedure Call (RPC)* 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 frontend.

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 Remote Procedure Call (RPC)* 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 **NATPARM** 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 Datasets 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 **CMPT nn** and **CMWKF nn**). For normal Natural batch processing, these files are defined in JCL by a logical file and a physical dataset 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 dataset 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 dataset name.

If a physical dataset name or a spool file class is specified, the access method itself allocates the dataset dynamically during the execution of the `DEFINE . . .` statement. To ensure an unique file name is used, `DEST=*` should be predefined in the `NATPARM` file. This avoids any file name conflicts.

69 Natural as a Server under CICS

▪ Functionality	404
▪ Natural CICS Interface Installation in a Server Environment	404
▪ Restrictions	405

This document applies under CICS only. It covers the following topics:

See also:

- *Natural under CICS*
- *Natural Remote Procedure Call (RPC)*

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 and the execution of remote procedure calls (see *Natural Remote Procedure Call (RPC)*).

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 frontend.

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 backend 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 backend program.
5. Care should be taken when using the `NCIPARM` terminal ID variable `&TID` 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.

70

Natural Execution - Miscellaneous Topics

This part provides general information on Natural execution.

- [Asynchronous Processing](#)
- [Double-Byte Character Sets](#)
- [Input/Output Devices](#)
- [Back-End Program Calling Conventions](#)
- [Natural 31-Bit Mode Support](#)
- [LE Subprograms](#)
- [External SORT](#)

For an explanation of the terms used in this document, see the *Glossary*.

71 Asynchronous Processing

- Identifying Asynchronous Natural Sessions 410
- Handling Output of an Asynchronous Natural Session 410
- Handling Unexpected or Unwanted Input 411
- Other Profile Parameter Considerations 411

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.

The following topics are covered:

Related Topics:

- *Asynchronous Natural Processing under CICS*
- *Asynchronous Natural Processing under Com-plete/SMARTS*
- *Asynchronous Transaction Processing under UTM*

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 `TTYPE` and by any `SET CONTROL 'T=xxxx'` statement; see also profile parameter `TTYPE` 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. In asynchronous mode, the setting of this parameter is forced to OFF to avoid problems when changing over to NEXT mode.
ENDMSG	The 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. In asynchronous mode, the setting of this parameter is forced to OFF to avoid problems when changing over to NEXT mode.

PC	In asynchronous mode, the setting of this parameter is forced to OFF to avoid problems.
PLOG	Dynamic parameter logging is executed by sending all parameters line by line to the SENDER destination.
PROGRAM	If a standard backend 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 backend logic.

72 Double-Byte Character Sets

- Natural Profile Parameter SOSI 414
- Output Format Specification 414
- Parameter Definitions for DBCS Support 414
- Editor Profile Options 415
- Input Data Check 415
- Output Data Adjustment 416
- Natural Stack Data 416
- Application Programming Interfaces for DBCS Handling 416

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 and covers the following topics:

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.

Parameter	Explanation
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 member (`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.

73

Input/Output Devices

- Terminal Support 420
- Light Pen Support 420
- Printer Support 422

This document provides some additional information on input/output devices supported by Natural.

The following topics are covered:

Terminal Support

Natural supports a wide variety of terminal types for the use with IBM and Siemens 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 `TTYPE` so that Natural can activate the appropriate converter routine to operate a specific type of terminal.

Links to related topics:

- [*NATDVCE - Terminal-Device Specification Table*](#)
- [*Parameters Affecting Terminal Communication*](#)
- [*NATCONFIG Module*](#) (various I/O translation topics)
- [*Siemens Terminal Types Supported by Natural*](#)
- [*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).
' ' (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 (PM=C) '....'  
....
```

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
F1 E3 C5 E2 E3
1 T E S T
4E E3 C5 E2 E3
+ T E S T 5A .. ..
..
Üæ . . .

```

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..I
40 F0 C6 C9 D9 E2 E3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 (hex)
   0 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 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 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
.....

```

74 Back-End Program Calling Conventions

- Back-End Program Calling Conventions (Batch Mode) 426
- Special Considerations under CICS 427
- Special Considerations under IMS TM 427
- Sample Back-End Programs 427

This document describes the conventions that apply to invoking a back-end program.

Notes:

- This section does not apply to BS2000/OSD; refer to *Calling Non-Natural Programs* and *Calling UTM Chained Partial Programs* in the *Natural TP Monitor Interfaces* documentation, section *Natural under UTM*.
- 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.

The following topics are covered:

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 are 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 parameter COMAMSG=NO is set in the Natural/CICS generation macro (NCIPARM), only the termination data are 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) / SYSLST (z/VSE) and the operator console; potential termination data are printed on SYSPRINT/SYSLST in dump format.

75

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.

76 LE Subprograms

- Support of IBM LE Subprograms 432
- Enabling Natural Support of LE Subprograms 432
- Passing LE Runtime Options 432
- LE Abend Handling 434

This document 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.

The following topics are covered:

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 LE subprograms from Natural:

1. When installing Natural, the corresponding driver must be generated with option `LE370=YES`. For LE enablement of Natural under CICS, see *Natural under CICS, Natural CICS Interface and IBM Language Environment (LE)* (in the *Natural TP Monitor Interfaces* 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

- [Passing LE Runtime Options under z/OS Batch and TSO](#)
- [Passing LE Runtime Options under z/VSE Batch](#)

- [Passing LE Runtime Options under IMS TM](#)

Passing LE Runtime Options under z/OS Batch and TSO

You have two 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 `CEEOPTS` input data set in your JCL. With the use of `CEEOPTS` the LE run-time options are also available to all subtasks. The use of `CEEOPTS` is especially required with a Natural RPC server in batch mode.

Example:

```
//CEEOPTS DD *  
POSIX(ON)  
/*
```

Passing LE Runtime Options 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= / , . . . '
```

Passing LE Runtime Options 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 modul 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 informations.

77 External SORT

▪ Support of External SORT	436
▪ Special Considerations for z/OS	436
▪ Special Considerations for z/VSE	436
▪ Special Considerations for BS2000/OSD	437

This document provides information on using external SORT programs with Natural.

The following topics are covered:

Support of External SORT

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.

Natural supports all external SORT programs that comply with the SORT interface documented in the relevant IBM manuals (for z/OS, z/VSE and CMS) and Siemens manuals (for BS2000/OSD).

The requirements (for example, space and datasets) 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 datasets 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/OSD

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 dataset 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.

Index
