Natural

Operations

Version 9.2.5

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.2.5 and all subsequent releases.

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

Copyright © 1979-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

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

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

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

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

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

Document ID: NATMF-OPERATIONS-925-20261009

Table of Contents

Preface ... ix
1 About this Documentation ... 1
Document Conventions ... 2
Online Information and Support ... 2
Data Protection ... 3
I Configuring Natural ... 5
2 Linking Natural Objects to the Natural Nucleus ... 7
Benefits ... 8
ULDOBJ Utility ... 9
Using ULDOBJ to Generate an Object Module ... 9
Additional Considerations for Linking Subroutines ... 11
Operating System Dependency of Object Module Generation ... 11
Example of Linking a Natural Object to the Natural Nucleus ... 11
3 Natural User Exits ... 15
NATUEX1 - User Exit for Authorization Control ... 16
NATSREX2 and NATSREX3 - User Exits for Sort Processing ... 17
NATUSKnn - User Exit for Computation of Sort Keys ... 18
NATPM - User Exit for Inverted Output ... 19
NREXPG - User Exit for NATRJE ... 20
USR0070P - User Exit for Editor Profiles ... 21
USR2002P - User Exit for Help Window Text Strings ... 21
USR2003P - User Exit for Main Menu ... 21
4 Natural User Access Method for Print and Work Files ... 23
NATAMUSR Module Description ... 24
NATAMUSR Module Installation ... 24
Invoking the Third Party Product ... 24
5 Natural System Files ... 25
Natural Scratch-Pad File ... 26
6 Natural Text Modules and Macros ... 29
Function and Usage of Text Modules ... 30
NATTEXT - Natural Keyword Definitions ... 30
NATTXT2 - Output Text, Keywords and User Termination Messages (Mixed Case) ... 31
NATTXT2U - Output Text, Keywords and User Termination Messages (Uppercase) ... 33
NATTXT3 - Text Fragments for Placeholders in Natural Error Messages ... 34
NTERMSG - Natural Termination Messages and Return Codes ... 35
7 Natural Configuration Tables ... 37
NATCONFG - Natural Configuration Tables ... 38
General Overview of Macros Used by NATCONFG ... 38
NTDVCE - Terminal-Device Specification Table ... 39
NTMSG - Message Log Table Definitions ... 40

NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus 40
NTCPAGE - Code Page Definitions 41
Code Page Support 42
Output Devices Supported 43
Translation Tables 44
Upper-/Lower-Case Translation 47
CMULT Entry 48
Output Translation 48
Input Translation 49
Code Translation of DBCS Data 49
NTTZ - Time Zone Definitions 49
8 Natural Storage Management 55
Thread and Non-Thread Environments 56
Buffer Types 56
Fixed Buffers 57
Variable Buffers 57
Customization of Buffer Characteristics 57
II Profile Parameter Usage 59
9 Natural Parameter Hierarchy 61
Natural Parameter Hierarchy Overview 62
General Rules for Parameter Usage 62
Natural Parameter Module 63
Predefined Dynamic Parameter Sets 64
Predefined User Parameter Profiles 64
Dynamic Parameter Entry 64
Natural Security Definitions 65
Session Settings for Profile Parameters 65
Program/Statement Level Settings 65
Development Environment Settings 66
Examples of Parameter Evaluation 66
10 Assignment of Parameter Values 69
Sources for Parameter Value Assignment 70
Static Assignment of Parameter Values 71
Dynamic Assignment of Parameter Values 72
Session Parameters for Runtime Assignment of Parameter Values 73
11 Building a Natural Parameter Module 75
NTPRM Parameter Macro 76
Additional Macros in the Natural Parameter Module 77
Example of Macros in the Natural Parameter Module 79
III z/OS Environment 81
12 Natural under z/OS 83
Natural Subsystem 84
TP Monitor Interfaces 84
Interfaces to Database Management Systems 84
Natural in Batch Mode under z/OS 85

Natural as a Server under z/OS ... 85
13 Authorized Services Manager under z/OS ... 87
ASM Overview ... 88
ASM System Requirements ... 89
Starting the ASM ... 91
ASM Operator Commands ... 96
Resetting the Coupling Facility Structure for ASM ... 97
ASM Messages, Condition Codes and Abend Codes ... 98
14 Natural Roll Server Functionality ... 101
Natural Roll Server - Overview ... 102
Roll Server in a Single z/OS System ... 103
Roll Server in a z/OS Parallel Sysplex Environment ... 104
Roll File and LRB ... 106
15 Natural Roll Server Operation ... 109
Roll Server System Requirements ... 110
Formatting the Roll File ... 112
Starting the Roll Server ... 115
Roll Server Messages, Condition Codes and Abend Codes ... 122
Return Codes and Reason Codes of the Roll Server Request ... 123
Operating the Roll Server ... 123
Resetting the Coupling Facility Structure ... 125
Roll Server Performance Tuning ... 125
Roll Server User Exits ... 126
IV Natural in Batch Mode ... 129
16 Natural in Batch Mode under z/OS ... 131
Natural z/OS Batch Interface ... 132
Driver Parameters for z/OS Batch ... 132
Data Sets Used by Natural in z/OS Batch Mode ... 132
17 Natural in Batch Mode (All Environments) ... 139
Adabas Data Sets ... 140
Sort Data Sets ... 140
Subtasking Session Support for Batch Mode Environments ... 140
V Natural Buffer Pools ... 145
18 Natural Buffer Pool - General ... 147
Natural Buffer Pool Principle of Operation ... 148
Buffer-Pool Monitoring and Maintenance ... 153
Natural Global Buffer Pool ... 156
19 Natural Global Buffer Pool under z/OS ... 157
Using a Natural Global Buffer Pool ... 158
Prerequisites ... 158
Operating the Natural Global Buffer Pool ... 158
Global Buffer Pool Manager Parameter Module ... 160
Global Buffer Pool Operating Functions ... 161
Global Buffer Pool Function Parameters ... 163
Examples of NATBUFFER Specifications ... 170

Sample NATGBPvr Execution Jobs ... 171
Localization ... 173
Messages ... 173
VI Message Buffer Pool ... 175
20 Message Buffer Pool ... 177
Purpose ... 178
Prerequisites ... 178
Operating the Message Buffer Pool ... 178
Sample NATMBPvr Execution Jobs ... 180
Message Buffer Pool Operating Functions ... 181
Function Parameters ... 182
Messages ... 184
VII System Spool Access ... 185
21 System Spool Access ... 187
Purpose ... 188
Prerequisite ... 188
Using the Write-to-Spool Feature ... 188
VIII Natural 3GL CALLNAT Interface ... 193
22 Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions ... 195
Purpose of 3GL CALLNAT Interface ... 196
Prerequisites ... 196
Restrictions ... 198
23 Natural 3GL CALLNAT Interface - Usage, Examples ... 201
Usage ... 202
Sample Environments ... 206
IX Operating the Software AG Editor ... 209
24 Editor Work File ... 211
Editor Work File Structure ... 212
Editor Work File under z/OS ... 213
Using the Editor Work File Formatting Utility ... 214
Formatting during Initialization ... 214
Maintaining the Editor Work File ... 214
Editor Work File under Complete/SMARTS ... 215
25 Editor Buffer Pool ... 217
Purpose of the Editor Buffer Pool ... 218
Obtaining Free Blocks ... 219
Initializing the Editor Buffer Pool ... 219
Restarting the Editor Buffer Pool ... 220
Editor Buffer Pool Parameters ... 220
Buffer Pool Initialization for Multi-User Environments ... 220
X Natural as a Server ... 223
26 Natural as a Server under z/OS ... 225
Functionality under z/OS ... 226
Natural Nucleus Installation in a Server Environment ... 227

Print and Work File Handling with External Data Sets in a Server Environment ... 227
27 Natural as a Server under CICS ... 229
Functionality under CICS ... 230
Natural CICS Interface Installation in a Server Environment ... 230
Restrictions ... 231
XI Natural Execution - Miscellaneous Topics ... 233
28 Natural 31-Bit Mode Support ... 235
29 Support and Use of Natural and Non-Natural Objects ... 237
Support for Natural Objects from Previous Natural Versions ... 238
Back-End Program Calling Conventions ... 238
LE Subprograms ... 240
External Sort Programs ... 243
30 Input/Output Devices ... 245
Terminal Support ... 246
Light Pen Support ... 246
Printer Support ... 247
31 Double-Byte Character Sets ... 251
Natural Profile Parameter SOSI ... 252
Output Format Specification ... 252
Parameter Definitions for DBCS Support ... 252
Editor Profile Options ... 253
Input Data Check ... 253
Output Data Adjustment ... 254
Natural Stack Data ... 254
Application Programming Interfaces for DBCS Handling ... 254
Alternate Text Module NATTXT2U ... 255
32 Asynchronous Processing ... 257
Identifying Asynchronous Natural Sessions ... 258
Handling Output of an Asynchronous Natural Session ... 258
Handling Unexpected or Unwanted Input ... 259
Other Profile Parameter Considerations ... 259

Preface

This documentation contains information for operating Natural in a z/OS environment.

This documentation is organized under the following headings:

Configuring Natural	Describes how to link Natural objects to the Natural nucleus. Provides information on Natural user exits, Natural user access method for print and work files, Natural system files, Natural text modules, Natural configuration tables, and Natural storage management.
Profile Parameter Usage	Provides an overview of the hierarchical structure of the different levels on which Natural parameters can be set. Explains how values can be assigned to profile parameters statically, dynamically and at runtime. Describes how to build a Natural parameter module.
z/OS Environment	Contains an overview of special considerations that apply when you are running Natural under z/OS online or in batch mode. Describes the functions and the operation of the Authorized Services Manager (ASM). Explains the functions of the Natural Roll Server. Provides information on the Roll Server system requirements, operation, performance tuning and restart capability.
Natural in Batch Mode	Contains considerations that apply when running Natural in batch (Adabas data sets, sort data sets, subtasking session support for batch environments), and specifically when running Natural in batch mode under z/OS.
Natural Buffer Pools	Contains information about the various storage management functions that are available to a Natural administrator under z/OS.
Message Buffer Pool	Provides information on how to start and operate the message text buffer pool.
System Spool Access	The Write-to-Spool feature enables Natural users to write reports to the system spool directly. It can be used in any Natural environment (Com-plete, TSO, CICS, IMS TM, batch, etc.) and uses the Entire System Server view `WRITE-SPOOL`. This output may then be processed by any software which expects output in JES or POWER spool (for example, Entire Output Management).
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 as a Server	Describes the use of Natural as a Server and the Natural Server Monitor.
Natural Execution - Miscellaneous Topics	Provides information on Natural 31-bit mode support, support and use of Natural and non-Natural objects, input/output devices, double-byte character sets and asynchronous processing.

Notation *vrs* or *vr*

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

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

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

Online Information and Support

Product Documentation

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

Product Training

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

Tech Community

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

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

Product Support

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

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

Data Protection

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

I Configuring Natural

This part provides information on Natural configuration.

Linking Natural Objects to the Natural Nucleus
Natural User Exits
Natural User Access Method for Print and Work Files
Natural System Files
Natural Text Modules and Macros
Natural Configuration Tables
Natural Storage Management

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

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

2 Linking Natural Objects to the Natural Nucleus

- Benefits 8
- ULDOBJ Utility 9
- Using ULDOBJ to Generate an Object Module 9
- Additional Considerations for Linking Subroutines 11
- Operating System Dependency of Object Module Generation 11
- Example of Linking a Natural Object to the Natural Nucleus 11

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

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

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

Benefits

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

- **Better Performance**
  The objects are executed from the nucleus and not from the Natural buffer pool. This saves space in the buffer pool and also results in fewer database calls. (If Natural cataloged objects are not linked to the Natural nucleus, they are stored in a database file, for example Adabas, and the actual code must be loaded from this file into the buffer pool before it can be executed.)
- **Consistency**
  As an object which is linked to the Natural nucleus is always executed from the nucleus, there is no effect if the cataloged object from which it was derived is deleted or changed in the Natural system file. Thus, during each TP-monitor session, the status of the object remains unchanged. A new version of an object which is linked to the nucleus can be obtained by unloading it with `ULDOBJ` (see below), relinking the new version to the Natural nucleus and refreshing the Natural module. (Refreshing implies that a new copy of a module is loaded into the TP monitor region.)
- **Global Error Handling**
  If a cataloged object fetches another program to handle errors (for example, by using the Natural system variable `*ERROR-TA`), and the error-handling program cannot be loaded into the buffer pool, the original error might be missed and any subsequent error may mask the first error and lead to confusion. To prevent this situation, you can link a user-written global error-handling program to the nucleus.

ULDOBJ Utility

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

When an environment-independent Natural nucleus is used, the generated object module has to be linked to the environment-*in*dependent part of the nucleus.

Using ULDOBJ to Generate an Object Module

To invoke the ULDOBJ utility

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

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


                    Specify parameters below ....

                              Object ...... ________   (Enter '.' to exit)
                              Library ..... SYSMISC_
                               OP System ... z/OS____
```

2 Specify and confirm the following parameters:

Object	The name of the cataloged object to be processed. The object can be a program, subprogram, subroutine, helproutine 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. The name of the operating system must be z/OS.

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

- the object type (Program, Subprogram, Subroutine, Helproutine, Map, Adapter); see *Objects for Natural Application Management* in the *Programming Guide*.
- the name of the cataloged object processed;

- the programming mode (S = structured mode, R = reporting mode);
- the name of the library containing the cataloged object;
- the name of the operating system for which the object deck was generated;
- the size of the cataloged object and optimized code (if applicable);
- the Natural version of the cataloged object (see *Version* in the *Glossary*);
- statistics about the last cataloging of the object, including user and terminal IDs.

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

To terminate the `ULDOBJ` utility

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

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

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

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

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

A `NAME` control statement is generated as the last card of the object module. It specifies the replace function. For example:

```
NAME TEST (R)
```

`TEST` is the name of the cataloged object.

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.

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

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

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

```
//JOBCARD 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 `NATCONFG` in the `NSTATIC` table for linked Natural programs:

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

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

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

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

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

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

3 Natural User Exits

- NATUEX1 - User Exit for Authorization Control 16
- NATSREX2 and NATSREX3 - User Exits for Sort Processing 17
- NATUSKnn - User Exit for Computation of Sort Keys 18
- NATPM - User Exit for Inverted Output 19
- NREXPG - User Exit for NATRJE 20
- USR0070P - User Exit for Editor Profiles 21
- USR2002P - User Exit for Help Window Text Strings 21
- USR2003P - User Exit for Main Menu 21

A Natural user exit is an 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.

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 the following 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.)
6	Address of work area (6KB length)

The values of addresses 1 to 5 can be modified by the user exit. Address 6 points to an area of 6KB that can be used as working storage.

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

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

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

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

NATSREX2 and NATSREX3 - User Exits for Sort Processing

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

The two user exits can be used with Natural's own sort program as well as with an external sort program. The exits are activated automatically when they are linked to the nucleus and so their addresses get resolved. Since 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.

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

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

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

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

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

NATUSKnn - User Exit for Computation of Sort Keys

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

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

You can write a NATUSK*nn* user exit in any programming language that provides a standard CALL interface. The character string specified with SORTKEY will be passed to the user exit. The user exit has to be programmed so that it converts “incorrectly sorted” characters in this string into corresponding “correctly sorted” characters. The converted character string is then used in the Natural program for further processing.

For the conversion, NATUSK*nn* may use the translation table NTUTAB1 of the configuration module NATCONFG; this means that NTUTAB1 may have to be adjusted accordingly.

NATUSK*nn* is called using standard calling conventions:

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

The parameter list contains the following addresses:

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

NATUSK*nn* 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.

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

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

NATPM - User Exit for Inverted Output

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

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

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

Inversion Logic

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

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

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

Field User Exit

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

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

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

NREXPG - User Exit for NATRJE

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

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

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

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

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

USR0070P - User Exit for Editor Profiles

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

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

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

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

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

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

USR2002P - User Exit for Help Window Text Strings

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

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

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

USR2003P - User Exit for Main Menu

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

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

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

4 Natural User Access Method for Print and Work Files

- NATAMUSR Module Description 24
- NATAMUSR Module Installation 24
- Invoking the Third Party Product 24

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

NATAMUSR Module Description

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

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

NATAMUSR Module Installation

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

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

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

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

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

Invoking the Third Party Product

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

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

5 Natural System Files

- Natural Scratch-Pad File 26

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

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

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

Natural Scratch-Pad File

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

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

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

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

- Recordings
- Screen Captures - NATPAGE
- File Maintenance

Related Topic:

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

Recordings

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

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

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

Screen Captures - NATPAGE

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

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.

6 Natural Text Modules and Macros

- Function and Usage of Text Modules 30
- NATTEXT - Natural Keyword Definitions 30
- NATTXT2 - Output Text, Keywords and User Termination Messages (Mixed Case) 31
- NATTXT2U - Output Text, Keywords and User Termination Messages (Uppercase) 33
- NATTXT3 - Text Fragments for Placeholders in Natural Error Messages 34
- NTERMSG - Natural Termination Messages and Return Codes 35

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

Function and Usage of Text Modules

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

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

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

NATTEXT - Natural Keyword Definitions

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

Modifying NATTEXT

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

The following rules apply:

- A keyword value for a NTKEY or NTALT macro can be changed by replacing the current keyword value with the desired value.
- A keyword or alternative keyword can be disabled by replacing the keyword value with the character "%".
- The position of each NTKEY and NTALT macro within the module is fixed and must not be shifted. Additional NTKEY and NTALT macros must not be inserted.
- Synonyms can be assigned for any keyword or alternative keyword using the NTSYN macro. One or more NTSYN macros can be inserted after a NTKEY or NTALT macro. The NTSYN macro includes one parameter, which is the value to be used as the synonym. If the synonym contains embedded blanks, the entire value must be enclosed in apostrophes.

Example of Modifying the NATTEXT Module

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

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

NATTEXT **before** modification:

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

NATTEXT **after** modification:

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

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

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

- Standard Natural Output Texts
- Keywords and Alternative Keywords for Natural System Commands and Utilities

- User-Written Termination Messages

Standard Natural Output Texts

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

- the literal Page used in the standard output page header;
- the name of each month as used in the Natural system variable *DATG (Gregorian date), date edit masks (L), and the name of each day as used in date edit masks (N);
- the ENTER INPUT DATA message and the skeleton error messages for error numbers 1104, 1105 and 1106 (used during online input processing);
- the error message used for system file open failure (which cannot be retrieved from the system file); an error number of the form NAT8*xxx* (where *xxx* is the decimal Adabas response code) is added to this error message by Natural;
- the constants More, Top and Bottom used in windows for position information to be displayed in text form;
- the table to define reports and report handling for reports greater than 33.

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

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

Keywords and Alternative Keywords for Natural System Commands and Utilities

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

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

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

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

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

User-Written Termination Messages

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

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

Example:

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

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

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

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

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

NATTXT3 - Text Fragments for Placeholders in Natural Error Messages

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

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

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

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

Example:

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

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

Trying to execute the object NOTEXIST leads to following result:

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

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

:1: was replaced by the text fragment Program.

The text fragment was declared in module NATTXT3 as follows:

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

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

NTERMSG - Natural Termination Messages and Return Codes

Natural has a number of standard session termination messages (NAT99...) that are delivered in macro NTERMSG and can be modified there (for example, to translate them 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 has to be re-assembled and linked.

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

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

With the profile parameter TS set to ON, the termination messages are translated to upper case using the upper case translation table NTUTAB1 as supplied in the NATCONFG 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.

7 Natural Configuration Tables

- NATCONFG - Natural Configuration Tables 38
- General Overview of Macros Used by NATCONFG 38
- NTDVCE - Terminal-Device Specification Table 39
- NTMSG - Message Log Table Definitions 40
- NTSTAT - Definition of Natural Objects Linked to the Natural Nucleus 40
- NTCPAGE - Code Page Definitions 41
- Code Page Support 42
- Output Devices Supported 43
- Translation Tables 44
- Upper-/Lower-Case Translation 47
- CMULT Entry 48
- Output Translation 48
- Input Translation 49
- Code Translation of DBCS Data 49
- NTTZ - Time Zone Definitions 49

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

See also ***Input/Output Devices Supported***.

NATCONFG - Natural Configuration Tables

The NATCONFG module contains the Natural configuration tables.

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

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

The NATCONFG 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 NATCONFG

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

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

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

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

For further details, see *Translation Tables*.

NTDVCE - Terminal-Device Specification Table

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

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

With the NTDVCE macro, it is possible to add new terminal drivers to Natural to specify modifications of the terminal-specific input/output or lower-to-upper case translation tables. Other information which can be specified is the frame character, the position of the message line, whether screen optimization is to be on or off, as well as various flags in the IOCB. In addition, the terminal specification can be routed to an existing driver by using other translate tables or can hook into a driver routine.

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

Note that the translation tables can be modified by the same macros dynamically or within the **Natural parameter module**.

NTMSG - Message Log Table Definitions

The macro NTMSG is used to define Natural messages which shall be written to the operator console or to the job message log (if available). A defined message will be written in addition, that is, the usual Natural processing remains unchanged. To find the log message definition table, locate label NATMSGT in NATCONFG. 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
object-name	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 NATCONFG. 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 NATCONFG, an error message is issued.

NTCPAGE Macro Syntax

The syntax of the NTCPAGE macro is as follows:

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

NTCPAGE Macro Parameters

Parameter	Description
IANA	This parameter is required under all operating systems. It specifies the standard name of the code page. Maximum length: 64 characters.
CCSID	This is a required parameter. It specifies the coded character set identification; that is, a numeric value with up to 5 digits. **Examples:** 1141 German EBCDIC code page 62243 Hebrew/Latin (ISO 8859) code page
ALIAS	This parameter is optional. It specifies the code page alias name. Maximum length: 32 characters.
PHC	This parameter is optional. It specifies the place holder character. Length: 2 bytes hexadecimal.

Parameter	Description	
ECS	This parameter is optional. It specifies the key number of the code page in Entire Conversion Service (ADAECS), which is used by Adabas.	
MULTI	This parameter is optional. It specifies whether the code page is a single-byte code page or a multi-byte or ASCII code page. Possible values:	
	ON	The code page is a single-byte code page; for example, IBM01140. The code page can be used as a Natural session code page. The session code page is defined by the Natural profile parameter CP.
	OFF	The code page is a multi-byte code page or ASCII code page. It cannot be used as a Natural session code page. Any attempt to use this code page results in initialization message NAT7019. This is the default setting.
	VALID	The code page is a multi-byte code page, but can be used as a Natural session code page. For example, IBM-939 is a Japanese EBCDIC code page that contains DBCS characters.

Examples:

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

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

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

Code Page Support

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

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

Output Devices Supported

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

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

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.
PC	All models and sizes which support standard data stream and/or extended data stream.

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 `NATCONFG`.

The following tables are provided:

Macro	Table
`NATSCTU`	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. **Important:** This table must never be changed.
`NATCPTAB`	Optional *single-byte* code page conversion accelerator tables. 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. The following code pages are supported by the delivered `NATCPTAB`: `IBM01140`

Macro	Table
	IBM01141 IBM01145 IBM01146 IBM01147 ASCII It is possible to add new entries by using the NTCPCNV macro. For each conversion direction, an entry is needed that contains the IANA name of the source code page, the IANA name of the target code page and optionally a blank character, a substitution character and a place holder character, followed by a complete list of character mappings.
NTSCTAB	The table which defines the properties of characters ■ used in mask definitions for the MASK option, ■ recognized as delimiters in the EXAMINE and SEPARATE statements. This table can be used to define upper-case attributes, lower-case attributes, special characters, hexadecimal characters and numeric characters. To modify this table, use the macro NTSCTAB in the Natural parameter module or the corresponding dynamic profile parameter SCTAB. If the CP profile parameter is set to a value other than OFF, the modification is ignored and the table is adjusted according to the code page used for the Natural session. See also *Translation Tables* in the *Unicode and Code Page Support* documentation.
NTTAB	The standard (primary) output translation table used for screen or printer output. Basically this table is used to translate all characters below X'40', that is from the space character to the question mark (X'00' is not translated). This guarantees that all terminal-control characters are translated before output and no control escape sequences can influence the screen output. Special characters (X'FE' and X'FF') which could influence the screen output are translated to the question mark (?). If nothing else is specified, all Natural output data are translated with NTTAB. To modify this table, use the macro NTTAB in the Natural parameter module or the corresponding dynamic profile parameter TAB. The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), and the table is adjusted by ICU according to the code page used at session start. Furthermore, all characters below X'40' are translated to the question mark (?) as described above. A character is excluded from this translation if either of the following conditions is true: ■ The character is explicitly translated to the same character. ■ The character is one of the logical shift-out/shift-in characters specified with the SOSI profile parameter (see the *Parameter Reference* documentation), and the specified code page is not an MBCS code page.

Macro	Table
NTTAB1	The alternative (secondary) output translation table for the secondary character set used when the Natural parameter PM is set to C. 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. To modify this table, use the macro NTTAB1 in the Natural parameter module or the corresponding dynamic profile parameter TAB1. The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), and the table is not used.
NTTAB2	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. To modify this table, use the macro NTTAB2 in the Natural parameter module or the corresponding dynamic profile parameter TAB2. The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), and the table is not used.
NTTABS	This table defines all valid characters that can be used in Natural variable names; it is used for the Natural syntax processor. It also defines all valid characters that can be used in the first position of a Natural variable name. In addition, it defines whether the variable is a global variable, a non-database variable or a source-code variable. If a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), the table is adjusted by ICU according to the code page used at session start.
NTUTAB1	The sample user-specific translation table for input translation from lower to upper case. In addition, this table performs the translation specified with the statement EXAMINE TRANSLATE INTO UPPER CASE. To modify this table, use the macro NTUTAB1 in the Natural parameter module or the corresponding dynamic profile parameter UTAB1. The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), and the table is not used.
NTUTAB2	The sample user-specific translation table which performs the translation specified with the statement EXAMINE TRANSLATE INTO LOWER CASE. To modify this table, you can use the macro NTUTAB2 in the Natural parameter module or the corresponding profile parameter UTAB2. The modification is ignored if a code page is specified using profile parameter CP (CP=ON, CP=AUTO or CP=*code-page*), and the table is not used.

Macro	Table
NTLANG	The language-code table, which defines which language number is assigned to which language code in the system variable *LANGUAGE.
NTTABL	The SYS* output translation table, which is controlled by the Natural profile parameter TS. With TS=ON, this table is used to translate output produced by programs located in Natural SYS* libraries (except modifiable fields) from Latin lower case to upper case. 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. To modify this table, use the macro NTTABL in the Natural parameter module or the corresponding dynamic profile parameter TABL. If Natural is running with an MBCS code page (for example, CP='IBM-939'), the table is not used, but translation is performed via ICU according to the current locale settings.
WRDFCUC1 WRDFCUC2 WRDFCSP2	The DBCS translation tables used to translate double-byte characters into Latin characters and vice versa. **Important:** These tables have to be activated explicitly, for example, for Far East countries.

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	Lower-case translation is switched off, which means that global upper-case translation is in effect. 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.)
%U	Upper-case translation is globally on. 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.
EXAMINE TRANSLATE	Upper-/lower-case translation can also be performed with the EXAMINE TRANSLATE statement. By default, EXAMINE TRANSLATE translates to upper case by using the translation table NTUTAB1, and to lower case by using the translation table NTUTAB2.

CMULT Entry

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

Output Translation

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

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

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

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

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

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

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

secondary translation table NTTAB1 is used. The data are simply translated by the primary translation table NTTAB.

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

Input Translation

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

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

Code Translation of DBCS Data

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

NTTZ - Time Zone Definitions

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

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

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

Reference point for automatic switching to and from summertime is the current machine time, which is UTC (GMT) time. Depending on the time period the current machine time is in, the current

local time is determined. The support of automatic switching to and from summertime is currently for years in the range from 2002 to 2041.

The following topics are covered below:

- NTTZ Macro Considerations and Restrictions
- NTTZ Macro Syntax
- NTTZ Macro Parameters
- Example of NTTZ Macro

NTTZ Macro Considerations and Restrictions

The following considerations and restrictions apply:

1. **Time Format**

   The basic time format is:

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

   or:

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

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

2. **UTC versus Local Time**

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

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

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

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

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

   Concurrent use of profile parameter `TD=`*`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=value;                                                    *
     TDON=value,                                                    *
     TDOFF=value,                                                   *
     SWTON=value,                                                   *
     SWTOFF=value,                                                  *
     DSTONvalue,                                                    *
     DSTOFF=value
```

NTTZ Macro Parameters

ZONE | TDON | TDOFF | SWTON | SWTOFF | DSTON | DSTOFF

ZONE - Time Zone Name

ZONE=*value* specifies the time zone name which can be referenced with the TD parameter. The first occurrence of a name will be selected.

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

TDON - Difference of Local Daylight Saving Time to UTC Time

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

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

Notes:

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

TDOFF - Difference of Local Zone Time to UTC Time

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

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

Note: This parameter corresponds to the parameter SWTOFF.

SWTON - Time when Daylight Saving Time Starts

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

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

SWTOFF - Time when Daylight Saving Time Ends

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

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

DSTON - Date when Daylight Saving Time Starts

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

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

Notes:

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

DSTOFF - Date when Daylight Saving Time Ends

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

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

Notes:

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

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 delivered NATCONFG module.

8 Natural Storage Management

- Thread and Non-Thread Environments 56
- Buffer Types 56
- Fixed Buffers 57
- Variable Buffers 57
- Customization of Buffer Characteristics 57

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

Thread and Non-Thread Environments

There are two different types of storage environments:

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

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

Upon certain events (terminal I/Os and long waits), the thread storage may be compressed and rolled out to external storage (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 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 NATCONFG, 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 NATCONFG by NTBUFID macro definitions.

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

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

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

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

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

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

Example of a buffer characteristics definition:

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

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

II Profile Parameter Usage

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

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

Related Topics:

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

9 Natural Parameter Hierarchy

- Natural Parameter Hierarchy Overview 62
- General Rules for Parameter Usage 62
- Natural Parameter Module 63
- Predefined Dynamic Parameter Sets 64
- Predefined User Parameter Profiles 64
- Dynamic Parameter Entry 64
- Natural Security Definitions 65
- Session Settings for Profile Parameters 65
- Program/Statement Level Settings 65
- Development Environment Settings 66
- Examples of Parameter Evaluation 66

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.

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	■ Development Environment Settings ■ Program/Statement Level Settings ■ Session Parameter Settings ■ Natural Security Definitions
Dynamic during Session Start	■ Dynamic Parameter Entry ■ Predefined User Parameter Profiles ■ Predefined Dynamic Parameter Sets ■ Alternative Natural Parameter Module
Static	Natural Parameter Module

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

General Rules for Parameter Usage

The following general rules apply:

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

Example:

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

The resulting value is `ESIZE=100`.

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

Natural Parameter Module

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

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

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

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

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

- Alternative Natural Parameter Module

Alternative Natural Parameter Module

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

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

Usage Restrictions

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

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

Predefined Dynamic Parameter Sets

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

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

See also the profile parameter SYS.

Predefined User Parameter Profiles

You can use the Natural utility 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 Natural parameter module to make analog settings. This, however, will prohibit the use of the profile parameter DYNPARM.

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

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

Natural Security Definitions

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

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

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

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

Session Settings for Profile Parameters

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

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

See also ***Session Parameters for Runtime Assignment of Parameter Values*** or SET GLOBALS.

Program/Statement Level Settings

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

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

Development Environment Settings

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

Examples of Parameter Evaluation

The examples below are based on the following parameter settings:

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

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

Example 1: No Dynamic Parameters

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

Example 2: PARM=ALTPARM

Resulting Values	Origin
DATSIZE 50	ALTPARM
Others: Default	ALTPARM

Example 3: `SYS=A`

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

Example 4: `PARM=ALTPARM,SYS=A`

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

Example 5: `PARM=ALTPARM,SYS=B`

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

Example 6: `SYS=A,PROFILE=MYPROF`

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

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

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

Example 8: `PROFILE=MYPROF,SYS=A`

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

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

Resulting Values	Origin
`DATSIZE 50`	`ALTPARM`
Others Default	`ALTPARM`

10 Assignment of Parameter Values

- Sources for Parameter Value Assignment 70
- Static Assignment of Parameter Values 71
- Dynamic Assignment of Parameter Values 72
- Session Parameters for Runtime Assignment of Parameter Values 73

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

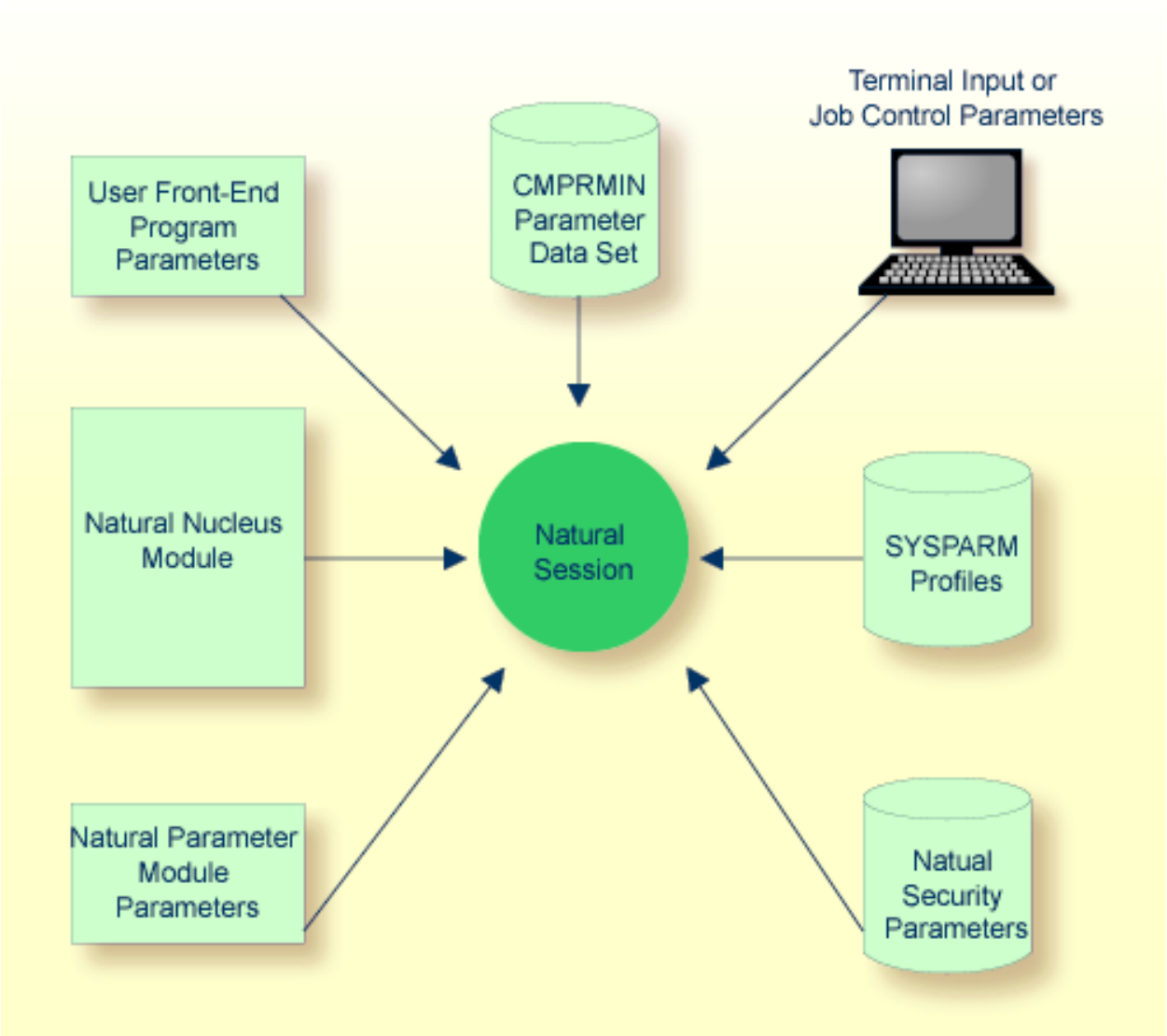
For details of the individual profile parameters, refer to the *Parameter Reference* documentation.

Sources for Parameter Value Assignment

The values for profile parameters are taken from three sources:

1. **Static assignments**
   Profile parameters specified in the parameter macro `NTPRM` and **additional parameter macros** contained in the **Natural parameter module**.
2. **Dynamic assignments**
   Parameters specified for the Natural session execution. These parameters override the static assignments and are valid for the current Natural session. Dynamic parameters can be passed by a front-end program, the parameter data set (`CMPRMIN`), session-initialization JCL, terminal input or Natural Security. In addition, it is possible to overwrite certain parameters by Natural program statements.
3. **Session parameters**
   Parameters specified with the system command `GLOBALS` (or a `SET GLOBALS` statement) within the current Natural session. The parameters override static and dynamic assignments.

Illustration of the Natural Parameter Assignment:



Static Assignment of Parameter Values

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

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

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

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

Static:	`macro-name keyword1=value,keyword2=value1,value2,...`
Dynamic:	`parameter-name=(keyword1=value,keyword2=value1,value2,...)`

Example:

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

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

See also the section ***Building a Natural Parameter Module***.

Dynamic Assignment of Parameter Values

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

Example:

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

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

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

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

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

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

Example:

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

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

Example for z/OS in Batch Mode:

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

Session Parameters for Runtime Assignment of Parameter Values

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

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

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

Example:

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

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

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

Example:

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

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

11 Building a Natural Parameter Module

- NTPRM Parameter Macro 76
- Additional Macros in the Natural Parameter Module 77
- Example of Macros in the Natural Parameter Module 79

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

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

To build a Natural parameter module using SMA jobs

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

   - Your `NTPRM` macro contains `CSTATIC` entries.
   - Your Natural parameter module contains an `NTCSTAT` macro.

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

NTPRM Parameter Macro

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

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

Additional Macros in the Natural Parameter Module

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

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

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

See also ***Example of Macros in the Natural Parameter Module***.

Naming Conventions and Overview of Macros

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

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

The following is an overview of available macros:

Macro	Purpose
NTALIAS	Defines external alias names for the modules linked to the Natural nucleus. Corresponding dynamic profile parameter: RCALIAS.
NTBPI	Assigns buffer pools to Natural sessions.
NTCCTAB	Defines printer-control sequences.
NTCFICU	Enables Unicode and code page support.
NTCMPO	Specifies compilation options.
NTCOMP	Specifies configuration settings for the Natural Com-plete/SMARTS Interface (Natural under Com-plete/SMARTS).
NTCSTAT	Defines the modules to be linked to the Natural nucleus. Corresponding dynamic profile parameter: CSTATIC.
NTDB	Defines database types and options for databases.
NTDB2	Specifies configuration settings for Natural for Db2.
NTDBGAT	Allows debugging of external Natural applications.
NTDS	Defines the sizes of storage buffers.
NTDYNP	Controls the use of dynamic profile parameters. Corresponding dynamic profile parameter: DYNPARM.

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

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

See also *Example of Macros in the Natural Parameter Module*.

Example of Macros in the Natural Parameter Module

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

```
NTPRM FNR=8,                          System File for NTPRM *
      DBID=001,                       Database ID for NTPRM *
      FNAT=(001,8),                   Natural System File   *
      FUSER=(001,9),                  Natural User File     *
      FDIC=(001,11),                  Predict System File   *
      FSEC=(001,10),                  Natural Security File *
      FREG=(001,52),                  Registry System File  *
      ESIZE=128,                      User Extension Area   *
      SLOCK=SPOD,                     Source Locking        *
      THSIZE=0,                       Thread Size           *
      UCONMAX=0,                      Max. Session Number   *
      CSTATIC=(CMMSG,                 Static. Modules Links *
      NSPPFUNC),                      Dummy Static. Module  *
      LE=OFF,                         Record Limit Error    *
      RECAT=OFF,                      Allow Stow of Macros  *
      PROFILE=,                       Profile Batch         *
      ADANAME=ADABAS,                 Adabas Link Routine   *
      ADASBV=OFF,                     Form. Buffer not Pass.*
      DFOUT=S,                        Output Format of Date *
      DFSTACK=S,                      Date Format for Stack *
      NUCNAME=NATvrsSH,               Natural Nucleus Name  *
      AUTO=OFF,                       Automatic Logon       *
      PC=ON,                          PC Connection         *
      LS=250,                         Default Line Size     *
      PS=80,                          Default Page Size     *
      STACK=OFF,                      Initial Natural Cmds. *
```

```
               ET=OFF                          END/BACKOUT TRANSACT.
*-----------------------------------------------------------------------*
         NTDB2 BTIGN=ON,                       Ignore Trans. Error   *
               CONVERS=ON,                     Convers. Mode CICS    *
               CONVRS2=OFF,                    Convers. Mode2 CICS   *
               DB2PLAN=PQANDBvr,               Plan Name             *
               DB2SSID=DB2A,                   Subsystem ID          *
               DB2XID=ON,                      Global Transaction ID *
               DDFSERV=CMFSERV,                DD Name File Server   *
               DELIMID=OFF,                    Delimited Identifiers *
               MAXLOOP=10,                     Nested Program Loops  *
               MAXSTMT=10,                     Dynamic SQL Statements*
               NNPSF=OFF,                      Set Positive Sign     *
               NSBHOST=IBM2.HQ.SAG,            NSB Server Host Name  *
               NSBPORT=7311,                   NSB Server TCP/IP Port*
               PSCIGN=OFF,                     Positive SQLCODEs     *
               REFRESH=OFF,                    Refresh Setting       *
               RETRYPO=10,                     Positioning Retries   *
               RWRDONL=ON,                     Delimited Identifiers *
               STATDYN=NEVER                   Static Dynamic Switch
*-----------------------------------------------------------------------*
         NTOSP ABEXIT=ESTAE,                   Abend Processing      *
               LBPNAME=' ',                    Local Shared Buffer   *
               LEHDLR=ON,                      LE Error Handler      *
               SUBPOOL=0,                      Subpool for GETMAIN   *
               TIOBSZ=(8,64),                  Primary I/O Buffer    *
               USERID=OFF                      Init-User Job Name
*-----------------------------------------------------------------------*
         NTVSAM BTSUPP=ON,                     BACKOUT TRANSACTION   *
               CLSUPP=ON,                      Close Call at Session *
               DDMCHK=OFF,                     Support of DDM        *
               DDSWITE=0,                      Maximum Entries DLBLY *
               DFBE=10,                        Decoded Format Buffer *
               DFBN=100,                       Format Buffer Entries *
               ENADIS=OFF,                     Enable Disabled Files *
               ENAUNE=OFF,                     Enable Unenabled Files*
               ETSUPP=ON,                      END TRANSACTION       *
               FORMAT=ON,                      Record Formatting     *
               KEYLGH=126,                     Length of VSAM Keys   *
               OPSUPP=OFF,                     Dynamic Open Calls    *
               PATH=CHECK,                     Path Processing       *
               PSIGNF=OFF,                     Compiler Option PSIGNF*
               RETRY=(OFF,OFF),                Retry ON ERROR Clause *
               RLS=OFF,                        Record-Level Sharing  *
               ROLLSIZ=550,                    Session Status Info.  *
               SFILE=ON,                       Support of VSAM Files *
               TAFE=10,                        Maximum No. DDMs      *
               TAFN=50,                        Maximum No. DDM Fields*
               TIMEOUT=0,                      Timeout RLS Request   *
               TSAE=20,                        READ/FIND Statement   *
               TVS=OFF,                        Support of DFSMSTVS   *
               UPDL=32768                      Size of Update Table
```

III z/OS Environment

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

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

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

12 Natural under z/OS

- Natural Subsystem 84
- TP Monitor Interfaces 84
- Interfaces to Database Management Systems 84
- Natural in Batch Mode under z/OS 85
- Natural as a Server under z/OS 85

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 `NAT`*v*, where *v* is the first digit of the current Natural version.

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

TP Monitor Interfaces

For information on the TP monitor interfaces that are available with Natural under z/OS, refer to the following sections in the *TP Monitor Interfaces* documentation:

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

Interfaces to Database Management Systems

Except for the 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 following sections in the *Database Management System Interfaces* documentation:

- *Natural for Db2*

- *Natural for VSAM*

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***.

13 Authorized Services Manager under z/OS

- ASM Overview 88
- ASM System Requirements 89
- Starting the ASM 91
- ASM Operator Commands 96
- Resetting the Coupling Facility Structure for ASM 97
- ASM Messages, Condition Codes and Abend Codes 98

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

ASM Overview

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

The following authorized functions are provided:

- communicating Natural buffer pool administration messages,
- write-access to global buffer pools in system key,
- writing SMF records,
- holding Natural session information in the Session Information Pool (SIP),
- executing authorized system services for IBM zIIP (System z Integrated Information Processor) support,
- executing authorized system services for z/OS shared memory objects.
- executing authorized services using the RACROUTE interface of the z/OS Security Server (RACF or any other external security product)

The first three functions are always available, whereas the SIP is needed and used when Natural is running with SYSPLEX enabled. For more information about the parameters you must specify to keep session information records (SIRs) in the SIP, see the CICSPLX and SIPSERV parameters under *Natural CICS Generation Parameters*. The SIP which holds the SIRs in SIP slots 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. This is necessary if your systems programmer specified `VSM ALLOWUSERKEYCSA(NO)` in `SYS1.PARMLIB(DIAG`*xx*`)`; see also ***Allocation of the Natural GBP*** in the section *Natural Global Buffer Pool under z/OS*.
- Natural under CICS is used in a z/OS Parallel Sysplex environment (SIP function required).
- Natural under IMS TM is used in terminal-oriented, non-conversational mode (with the SIP function).
- Natural under IMS TM is used, with the Accounting function writing SMF records.
- Natural under IMS TM is used, with the Broadcasting function.

- Natural under IMS TM is used, with the Monitor function.
- Enablement of zIIP support is required.
- Enablement of the Shared Memory Objects File Server (FSSM) of Natural for Db2 is required.
- Enablement of the ACEE caching (`SECURITY_CACHING=YES`) of the Natural Development Server.

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

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

Note concerning Natural/CICS:

The CICS System Recovery Table should include the z/OS system abend code `0D6`.

ASM System Requirements

This section describes the ASM system requirements.

- APF Authorization
- System Linkage Index
- CF Structure
- XCF Signaling Paths

APF Authorization

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

System Linkage Index

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

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

CF Structure

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

The size of a CF structure can be calculated using the IBM web utility "CFSizer", which can be found here:**https://www.ibm.com/support/pages/cfsizer**

There, you must choose "XCF" in the selection box, there where you see "Choose one".

You get a new window, which is pre-filled with "Number of systems" = 8 and "CLASSLEN" = 956 (= 936 + 20 -> CONA + CONALOCKATTR)

See macro `IXLYCONA` for those size:

```
01 SIZE:
*            CONA          -- X'03A8' bytes = 936
*            CONALOCKATTR  -- X'0014' bytes =  20
*            CONALISTATTR  -- X'0028' bytes =  40
*            CONACACHEATTR -- X'001C' bytes =  28
```

Authorized Server needs CONA + CONALISTATTR = 936 + 40 = 976 bytes.

Once you select "Number of systems" = n (LPARs), and you press "Submit", you get something like (5 LPARs with 936 bytes):

Function	Type	Structure Name	INITSIZE	SIZE
XCF	LIST	IXC.....	18M	19M

followed by a number of lines with JCL.

XCF Signaling Paths

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

Starting the ASM

You start the ASM either as a batch job or as a started task by executing module `NATASMvr`, where *vr* represents the relevant product version. You can specify parameters in the JCL `EXEC` statement, in a parameter file, or in both. A parameter specified in the `EXEC` statement overwrites the corresponding parameter in the parameter file.

It is recommended that you use a parameter file (see ***Parameters in the Parameter File***) because the parameters that control SIP timeout processing and future parameters can only be specified in the parameter file. Existing JCL will continue to execute unchanged.

This section covers the following topics:

- Parameters in the JCL EXEC Statement
- Parameters in the Parameter File

Parameters in the JCL EXEC Statement

In 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,Update-ECSA-D
```

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

Parameter	Possible Values	Default Value	Comment
subsystem-id	4-byte non-blank string	`NATv`	The specified value must match the value of the Natural profile parameter `SUBSID` (*v* is version). **Note:** With Natural under CICS, refer to the `CICSPLX` parameter in the NCMDIR macro for setting the appropriate subsystem ID.

Parameter	Possible Values	Default Value	Comment
XCF-group-name	any valid XCF group name	none	The name of the XCF group for Signaling services. An asterisk ("*") will produce the name "NAT" followed by the subsystem-name. It is, however, indicated that the name of a XCF group should **always** be specified.
CF-structure-name	any valid CF structure name	none	Optional, only needed if SIP is used. The name of the CF structure used for the SIP function. When *XCF-group-name* is specified while *CF-structure-name* remains empty, you will only be able to use the buffer pool propagation.
number-of-SIP-slots	1 - 2147483647	none	Optional, only needed if SIP is used. The number of slots to be allocated if the CF structure has not yet been allocated. If omitted or specified as 0, the entire structure will be used for as many slots as it can hold.
SIP-slot-size	256, 512, 1024, 2048 or 4096	1024	The specified value is ignored if a CF structure has already been allocated.
message-case	UCTRAN or blank	blank	Specify UCTRAN if the Authorized Services Manager is to issue all its messages in upper case.
Update-ECSA-D	ECSADUPD or blank	blank	Update an older ECSA-Directory entry without the need for an IPL.

Parameters in the Parameter File

The parameter file is a physical sequential file (DSORG=PS) that is allocated with LRECL=80 and RECFM=FB. In your JCL, specify this file with DDNAME ASMPARM.

Parameters in the parameter file are specified as *name=value* pairs. Specify one parameter per line starting in Column 1. The *name=value* pair is terminated by the first blank, and the rest of the line is not examined. Lines starting with an asterisk (*) in Column 1 are treated as comments. Parameters are translated to upper case before they are processed.

Parameter	Possible Values	Default Value	Comment
SUBSID=*name*	4-byte non-blank string	NAT*v*	The specified value must match the value of the Natural profile parameter SUBSID (*v* is version). **Note:** With Natural under CICS, refer to the CICSPLX parameter in the NCMDIR macro for setting the appropriate subsystem ID.
XCFGROUP=*group-name*	any valid XCF group name	none	The name of the XCF group for Signaling services. An asterisk ("*") will produce the name "NAT" followed by the subsystem-name. It is, however, indicated that the name of a XCF group should **always** be specified.
STRUCTURE=*structure-name*	any valid CF structure name	none	Use only for the SIP function. The name of the CF structure used for the SIP function. When XCFGROUP is specified while STRUCTURE remains empty, you will only be able to use the buffer pool propagation.
NUMSLOTS=*number*	1 - 2147483647	none	Use only for the SIP function. 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.
SLOTSIZE=*size*	256, 512, 1024, 2048 or 4096	1024	Use only for the SIP function. The specified value is ignored if a CF structure has already been allocated.
MSGCASE=*case*	UPPER or MIXED	MIXED	Specify UPPER if the Authorized Services Manager is to issue all its messages in upper case.
NONACTIVITY=*hours*	1 - 999999	none	Use only for the SIP function. The number of hours a SIP session can be inactive before it is deleted. 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.

Parameter	Possible Values	Default Value	Comment
			This parameter can be changed using the `TIMEOUT` operator command (see *ASM Operator Commands*). Both the Authorized Services Manager and the Roll Server allow to specify a timeout value. If Natural is running in a SYSPLEX environment, set the same value for this parameter and the `non-activity-time` parameter of the Roll Server.
`TIMEOUTCHECK=`*hhmm*	`0000 - 2359`	none	Use only for the SIP function. The time of day that the timeout check is to be run. Sessions will be deleted if they have been inactive longer than the non-activity time specified with `NONACTIVITY`. This parameter can be changed using the `TIMEOUT` operator command (see *ASM Operator Commands*).
`TIMEOUTREPEAT=`*mmmm*	`0 - 1440`	none	Use only for the SIP function. The number of minutes between two timeout checks. If `TIMEOUTCHECK` is also specified, the first check is run at the time specified with `TIMEOUTCHECK`, and then repeated after *mmmm* minutes. If `TIMEOUTCHECK` is not specified, the first check is run *mmmm* minutes after Authorized Services Manager start. This parameter can be changed using the `REPEAT` option of the `TIMEOUT` operator command (see *ASM Operator Commands*).
`FSSMDST`*x*	See *How to Enable Tracing for Dynamic SQL Statements* in the *Database Management System Interfaces* documentation.	none	Parameters starting with `FSSMDSTx` are passed to NATFSSM, the Natural for Db2 part of the Authorized Services Manager (NATASM), to define a dynamic SQL trace buffer above the bar.

Parameter	Possible Values	Default Value	Comment
FSSM*xxxx*	See *Defining Size and Format of an FSSM* in the *Database Management System Interfaces* documentation.	none	Parameters starting with FSSMxxxx (xxxx must not be replaced with DSTx) are passed to NATFSSM, the Natural for Db2 part of the Authorized Services Manager (NATASM), to define the Natural for Db2 file server above the bar.
ECSADUPD=*option*	no or yes	no	Update an older ECSA-Directory entry without the need for an IPL.
TIMEOUT=*option*	VERBOSE or TERSE	VERBOSE	Displays or suppresses messages ASM0085 and ASM0086 during a TIMEOUT process.

Note: If TIMEOUTCHECK is specified and its value is earlier than the start time of NATASMvr, the next TIMEOUTCHECK value is calculated starting from the specified TIMEOUTCHECK value, as if one or more TIMEOUTCHECKs had already been executed. However, this is only possible starting with NATASM93.

Example:

Examples:

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

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

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

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

  Same as above, except SIP slots:

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

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

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

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

  The SIP service will not be available.

- ```
  //ASM EXEC PGM=NATASMvr,PARM='TST5'
  //ASMPARM  DD DISP=SHR,DSN=FB.SYSF.PARMS(ASMPARM1)
  ```

  File `FB.SYSF.PARMS(ASMPARM1)`:

  ```
  MSGCASE=M              Mixed case messages
  SUBSID=TST1
  XCFGROUP=HELGA         Message group for global buffer pool administration
  *SIP definitions:
  STRUCTURE=TSTSIP       SIP CF structure
  SLOTSIZE=256
  NUMSLOTS=200
  TIMEOUTCHECK=2135      Delete old sessions at 9:35 pm
  NONACTIVITY=2         Delete sessions that have been inactive for 2 hours or ↩
  more
  ```

  The SIP service is to build 200 SIP slots in CF structure `TSTSIP`, each having a size of 256 bytes. The Natural subsystem to be used is `TST5`, as the parameter on the `EXEC` statement overwrites the `TST1` subsystem specified in the parameter file.

ASM Operator Commands

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

Command		Description
`HELP`		Shows an overview of the available syntax.
`TERMinate` | `STOP`		Terminates the ASM.
`SNAP`		Debugging function. The ASM address space is dumped to `SYSUDUMP`.
`VLIST`		Displays the name, version and assembly time of modules that are linked to the ASM.
`TIMEOUT`	`NAT` *nnn*	Specifies or replaces the non-activity time parameter.
	`REPEAT` *mmmm*	Specifies or replaces the time interval in minutes in which the timeout check is to be run.
	`TOC` *hhmm*	Specifies or replaces the time of day of the timeout check.
	`OFF`	Disables timeout checking.
	`ON`	Reinstates timeout checking.
  ```

Command		Description
	NOW	Starts an immediate timeout check. Normal timeout check scheduling (if specified) remains in effect.
	TERSE	Suppresses the messages ASM0078 and ASM0080 during TIMEOUT NOW processing. Suppresses the messages ASM0085 and ASM0086 during a TIMEOUT processing. The message ASM0047 Operator command: TIMEOUT NOW is also suppressed.
	VERBOSE	Displays the messages ASM0078, ASM0080 and ASM0047 during TIMEOUT NOW processing. Displays the messages ASM0085 and ASM0086 during a TIMEOUT processing. This is the default setting.
	? (or no specification)	Displays the current timeout settings. The question mark (?) is optional and can be omitted.

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.

Resetting the Coupling Facility Structure for ASM

When a CICS or IMS TM region which uses the Session Information Pool (SIP) function of the Authorized Services Manager abends, the Authorized Services Manager might return an error for the SIP. For example, the SIP might be reported as full because session cleanup had not been performed before the region abended. To resolve such an error, delete the respective Coupling Facility structure:

1. Shut down all Authorized Services Managers which use the affected CF structure.
2. Issue the operator command SETXCF FORCE,STR,STRNAME=*structure-name*, where *structure-name* is the name of the CF structure used for the SIP function.
3. Restart all Authorized Services Managers.

ASM Messages, Condition Codes and Abend Codes

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

```
ASM0005 FBASMvr
```

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

The following condition codes are used:

Condition Code	Explanation
0	Normal completion.
4	Authorised server not available.
8	No space for work area.
12	Wrong parameter input.
16	Runtime error has occurred.
20	Subtask has failed.
24	Abend has occurred.
97	Buffer pool routine not linked.
98	Invalid buffer pool type.
99	No more storage.
>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 support.
U0104	`IXCMSGI` failed.	Abend Register 14 contains the reason code.
U0105	Message Exit could not obtain a Purge Task Request Block.	Contact support.
U0106	Work Space for `IXLCONN` could not be obtained.	Contact support.
U0251	Real “bad abend” indicator (no memory available).	Contact support.
U05*xx*	`IXLCONN` failed.	*xx* is the reason code.
U06*xx*	`IXLLIST WRITE` failed.	*xx* is the reason code.

Abend Code	Reason	Comment
U070*x*	STORAGE OBTAIN for subpool 245 failed.	Contact support.
U071*x*	STORAGE RELEASE for subpool 245 failed.	Contact support.
U0800	LXRES failed.	Contact support.
U0810	AXSET failed.	Contact support.
U0820	ETCRE failed.	Contact support.
U0821	ETCON failed.	Contact support.
U0830	IDENTIFY failed.	Contact support.

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 support.

14 Natural Roll Server Functionality

- Natural Roll Server - Overview 102
- Roll Server in a Single z/OS System 103
- Roll Server in a z/OS Parallel Sysplex Environment 104
- Roll File and LRB 106

See also ***Natural Roll Server Operation***.

Natural Roll Server - Overview

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

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

Before a roll out is performed, Natural compresses the thread into one contiguous buffer, and decompresses it after the roll-in is performed. Depending on the Natural version installed at your site, the CPU load of compression and decompression can be taken off the hosting TP system and moved to respective routines within the Roll Server. If you want to take advantage of the compression/decompression feature, install the appropriate Roll Server module described in the respective installation step in the section *Installing Natural on z/OS* in the *Installation for Natural on z/OS* documentation. In addition, Natural Batch for zIIP (extra product license required) must be available to the Natural nucleus (see the respective installation step) in *Installing Natural on z/OS*.

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

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

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

Roll Server in a Single z/OS System

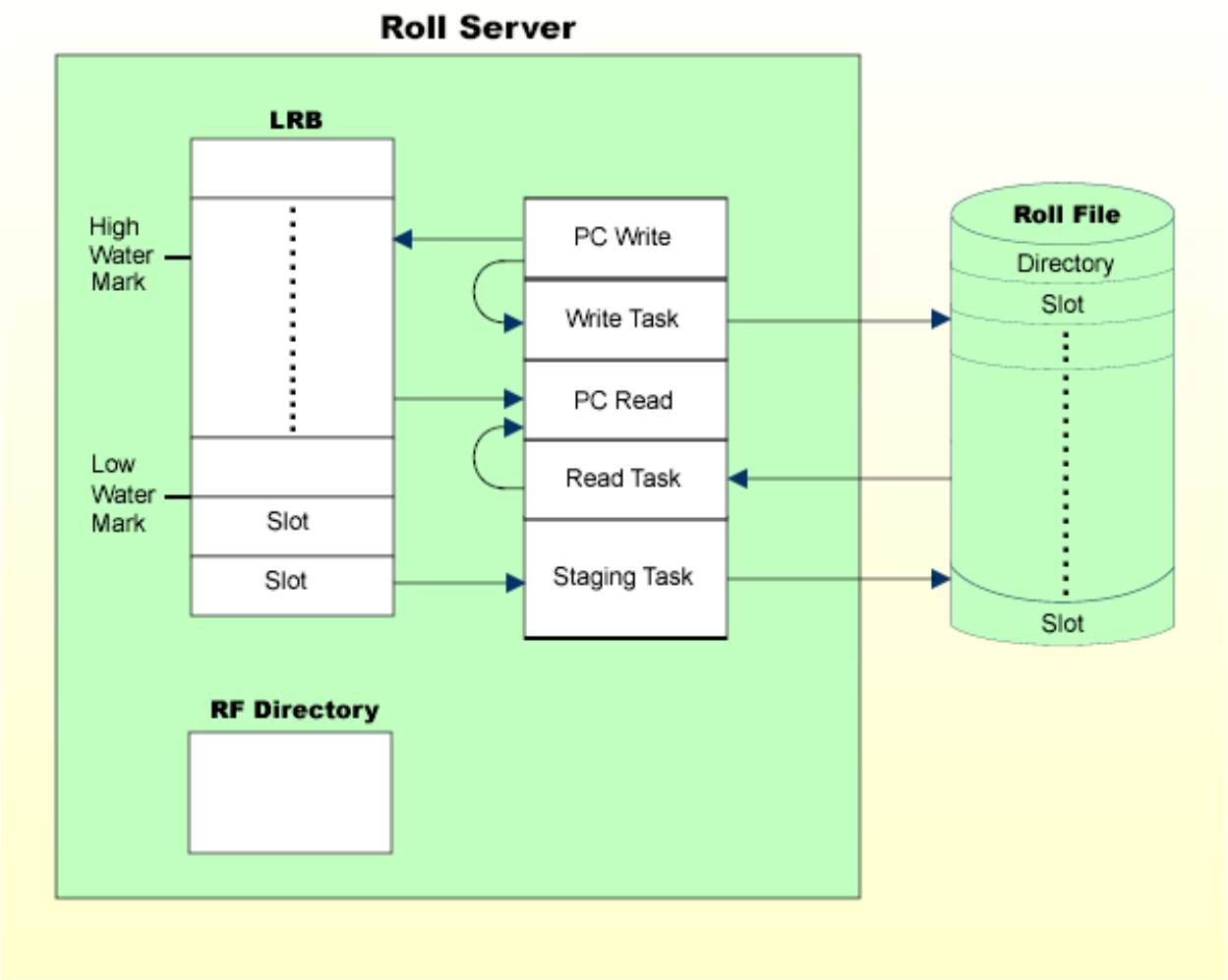
When the Roll Server receives a thread through a write request (before terminal output), it checks whether enough space is available in the local roll buffer (LRB). If there is, the thread is copied to the LRB. If not, the thread is written to the roll file. If the thread is larger than the roll file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

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

As the main and overflow slots must reside in the same roll file, it is not possible to allocate an overflow slot **outside** the LRB once the main slot could be allocated **within** the LRB. This would mean that the whole write operation should be repeated from the beginning, and the main slot should then be stored in a roll file, even if there is space for that main slot **in** the LRB, so that a subsequent overflow slot could be stored in the same roll file as the main slot.

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

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



Roll Server in a z/OS Parallel Sysplex Environment

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

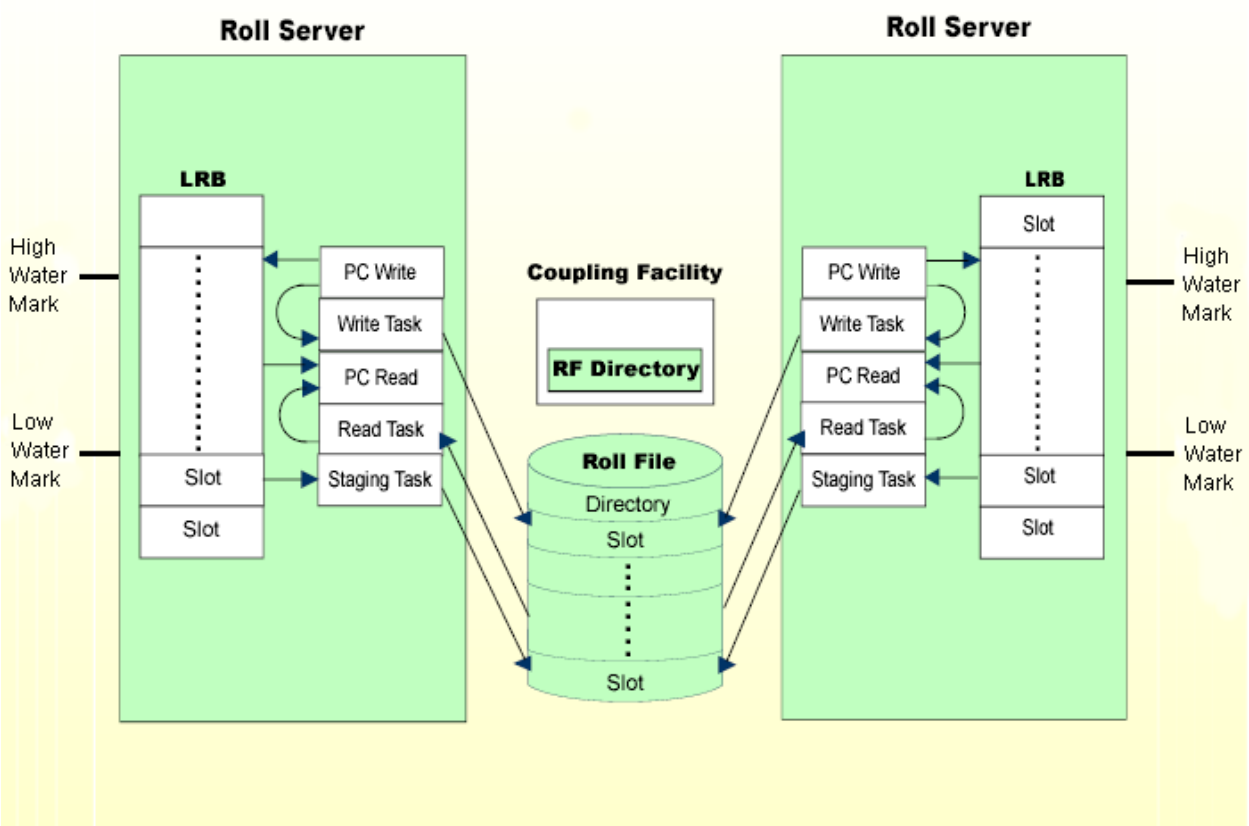
When the Roll Server receives a thread through a write request (before terminal output), it checks whether enough space is available in the local roll buffer (LRB). If there is enough space, the thread is copied to the LRB. If there is not enough space in the LRB, the thread is written directly to the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

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

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

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

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



Roll File and LRB

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

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

The local roll buffer acts as a cache for the roll file. The roll buffer is contained in a z/OS memory object and subdivided into fixed-length slots. The LRB slot size is identical to the slot size of the corresponding roll file. If the Roll Server is to run without a roll file, the LRB slot size must be specified as parameter on the JCL `EXEC` statement. See ***Natural Roll Server Operation***, ***Starting the Roll Server***.

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

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

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

15 Natural Roll Server Operation

- Roll Server System Requirements 110
- Formatting the Roll File 112
- Starting the Roll Server 115
- Roll Server Messages, Condition Codes and Abend Codes 122
- Return Codes and Reason Codes of the Roll Server Request 123
- Operating the Roll Server 123
- Resetting the Coupling Facility Structure 125
- Roll Server Performance Tuning 125
- Roll Server User Exits 126

See also ***Natural Roll Server Functionality***.

Roll Server System Requirements

This section describes the Roll Server system requirements.

- APF Authorization
- System Linkage Index
- Virtual Storage
- CF Structure
- XCF Signaling Paths

APF Authorization

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

System Linkage Index

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

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

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

Virtual Storage

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

CF Structure

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

The size of a CF structure can be calculated using the IBM web utility "CFSizer", which can be found here:**https://www.ibm.com/support/pages/cfsizer**

There, you must choose "XCF" in the selection box, there where you see "Choose one".

You get a new window, which is pre-filled with "Number of systems" = 8 and "CLASSLEN" = 956 (= 936 + 20 -> CONA + CONALOCKATTR)

See macro `IXLYCONA` for those size:

```
01 SIZE:
*            CONA          -- X'03A8' bytes = 936
*            CONALOCKATTR  -- X'0014' bytes =  20
*            CONALISTATTR  -- X'0028' bytes =  40
*            CONACACHEATTR -- X'001C' bytes =  28
```

Roll Server only uses CONA = 936 bytes.

Once you select "Number of systems" = n (LPARs), and you press "Submit", you get something like (5 LPARs with 936 bytes):

Function	Type	Structure Name	INITSIZE	SIZE
XCF	LIST	IXC.....	18M	19M

followed by a number of lines with JCL.

XCF Signaling Paths

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

If you want to specify your own XCF group name, use the `NATRSU24` user exit. For more information on this user exit, see ***NATRSU24 User Exit***.

Formatting the Roll File

To format the roll file, proceed as follows:

1. Allocate it as a physical, sequential data set with a fixed-record format.
2. Format it using module `NATRSRFI`.
3. If the roll server executes in a z/OS Parallel Sysplex environment, reset the Coupling Facility structure as described in the **Notes Concerning the Formatting or Resetting of Roll Files**.

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	
function	`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 *dd-name*.
	`LIST`	Print a list of session IDs contained in the roll file and their last activity. The only other parameter allowed is *dd-name*.
dd-name	The name of the `DD` statement under which the roll file has been specified.	
slot-size	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.	
number-of-slots	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.	

Parameter	Description
	Note that during formatting, the system abend code SB37 or SD37 (or S209 for a VSAM file) can be encountered. This abend is intercepted by the formatting routine and can be ignored.

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

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

or in tracks

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

The block size used is:

23476 for 3380 DASD

27998 for 3390 DASD

22928 for 9345 DASD

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

NATRSRFI Output

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

NATRSRFI Condition and Abend Codes:

The following condition codes are used:

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

The following user abend codes are used:

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

Examples:

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

Example 1:

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

Excerpt from resulting JESMSGLG:

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

Example 2:

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

Resulting `RFIPRINT`:

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

Notes Concerning the Formatting or Resetting of Roll Files

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

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

Starting the Roll Server

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

You can specify parameters in the JCL `EXEC` statement, in a parameter file, or in both. A parameter specified in the `EXEC` statement overwrites the corresponding parameter in the parameter file.

It is recommended that you use a parameter file. The parameter `TIMEOUTREPEAT` (see *Parameters in the Parameter File*) and future parameters can only be specified in the parameter file. Existing JCL will continue to execute unchanged.

This section covers the following topics:

- Parameters in the JCL EXEC Statement
- Parameters in the Parameter File

- Examples for Starting the Roll Server as a Batch Job

Parameters in the JCL EXEC Statement

Parameters in the JCL `EXEC` statement:

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

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

Parameter	Parameter name for parameter dataset	Possible Values	Default	Comment
subsystem-id	SUBSID	4-byte non-blank string	NAT*v*	The specified value must match the value of the Natural profile parameter SUBSID (*v* = *v*ersion number). **Note:** With Natural under CICS, refer to the ROLLSRV parameter in the NCMDIR macro for setting the appropriate subsystem ID.
number-of-roll-files	NUMFILES	0 - 5	1	In a z/OS non-Parallel Sysplex environment, the Roll Server can operate without a roll file, using only the in-storage Local Roll Buffer.
number-of-LRB-slots	NUMSLOTS	1 - 32767	none	The number of LRB slots multiplied by the slot size must not exceed 2 TB. 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: *number-of-roll-files * number-of-LRB-slots*
LRB-slot-size	SLOTSIZE	0..67108864	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.

Parameter	Parameter name for parameter dataset	Possible Values	Default	Comment
				If no roll files are used meaning the *number-of-roll-files* is zero, this parameter is mandatory.
CF-structure-name	STRUCTURE	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). Otherwise, using this parameter might cause unnecessary overhead
low-water-mark	LOWWATER	0 - 9	7 (single z/OS) 0 (sysplex)	Specifies the low water mark in steps of 10 percent of the number of LRB slots. LWM can not be higher than the current HWM. This value is always ignored and reset to 0 if *CF-structure-name* has a value, independently of what was specified.
high-water-mark	HIGHWATER	0 - 10	8 (single z/OS) 0 (sysplex)	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. HWM can not be lower than the current LWM. This value is always ignored and reset to 0 if *CF-structure-name* has a value, independently of what was specified.
non-activity-time	NONACTIVITY	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.

Parameter	Parameter name for parameter dataset	Possible Values	Default	Comment
				If this parameter is omitted, no timeout check will be executed. This parameter can be changed using operator command TIMEOUT, see below. Both the Authorized Services Manager and the Roll Server allow to specify a timeout value. If Natural is running in a SYSPLEX environment, set the same value for this parameter and the NONACTIVITY parameter of the Authorized Services Manager.
timeout-check-time	TIMEOUTCHECK	0000 - 2359	none	The time of day that the timeout check is to be run. Sessions will be deleted if they have been inactive longer than the non-activity time specified by the preceding parameter. If this parameter is omitted, no timeout check will be scheduled. This parameter can be changed using operator command TIMEOUT, see below.
message-case	MSGCASE	UCTRAN or blank	blank	Specify UCTRAN if the Roll Server is to issue all its messages in upper case.
repeat-timeout-check	TIMEOUTREPEAT	0000 - 1440	none	The number of minutes between two consecutive timeout checks. This allows for a timeout check to occur several times a day.

Parameters in the Parameter File

The parameter file is a physical sequential file (DSORG=PS). In your JCL, specify this file with `DDNAME RSMPARM`.

Parameters in the parameter file are specified as name=value pairs, for example, `NUMFILES=1`. Specify one parameter per line, starting in Column 1. The name=value pair is terminated by the first blank, and the rest of the line is not examined. Lines starting with an asterisk (*) in Column 1 are treated as comments. Parameters are translated to upper case before they are processed.

Parameter	Possible Values	Default	Comment
`SUBSID`	4-byte non-blank string	`NAT`*v*	The specified value must match the value of the Natural profile parameter `SUBSID` (*v* = *v*ersion number). **Note:** With Natural under CICS, refer to the `CICSPLX` parameter in the `NCMDIR` macro for setting the appropriate subsystem ID.
`NUMFILES`	`0 - 5`	`1`	In a z/OS non-Parallel Sysplex environment, the Roll Server can operate without a roll file, using only the in-storage Local Roll Buffer.
`NUMSLOTS`	`1 - 32767`	none	The number of LRB slots assigned for each roll file in use. (The total number of LRB slots is calculated by this formula: NUMFILES * NUMSLOTS)
`SLOTSIZE`	`0..67108864`	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.
`STRUCTURE`	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).
`LOWWATER`	`0 - 9`	`7` (single z/OS) `0` (Sysplex)	Specifies the low water mark in steps of 10 percent of the number of LRB slots. LWM can not be higher than the current HWM. This value is always ignored and reset to `0` if `STRUCTURE` has a value, independently of what was specified.
`HIGHWATER`	`0 - 10`	`8` (single z/OS) `0` (Sysplex)	Analogous to *`low-water-mark`* parameter. A value of `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.

Parameter	Possible Values	Default	Comment
			HWM can not be lower than the current LWM. This value is always ignored and reset to `0` if `STRUCTURE` has a value, independently of what was specified..
`NONACTIVITY`	`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 the operator command `TIMEOUT`, (see *Operating the Roll Server*).
`TIMEOUTCHECK`	`0000 - 2359`	none	The time of day that the timeout check is to be run. Sessions will be deleted if they have been inactive longer than the non-activity time specified by the preceding parameter. If this parameter is omitted, no timeout check will be scheduled. This parameter can be changed using operator command `TIMEOUT` (see *Operating the Roll Server*).
`TIMEOUTREPEAT`	`0 - 1440`	none	The number of minutes between two consecutive timeout checks. The first timeout check after Roll Server start is run after `TIMEOUTREPEAT` minutes. If `TIMEOUTCHECK` is also specified, the first timeout check is run at the time specified with `TIMEOUTCHECK`, and then every `TIMEOUTREPEAT` minutes. Specifying `0` or `1440` indicates no timeout repeat, and the timeout check is run only at the time specified with `TIMEOUTCHECK`.
`MSGCASE`	`UPPER` or `MIXED`	mixed case	`MSGCASE=UPPER` causes all messages to be displayed in uppercase letters. `MSGCASE=MIXED` causes all messages to be displayed in mixed case letters.
`TIMEOUT`	`VERBOSE` or `TERSE`	`VERBOSE`	Displays or suppresses the message RSM0096 during a `TIMEOUT` processing.

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".

Note: If `TIMEOUTCHECK` is specified and its value is earlier than the start time of NATRSMvr, the next `TIMEOUTCHECK` value is calculated starting from the specified `TIMEOUTCHECK` value, as if one or more `TIMEOUTCHECK`s had already been executed. However, this is only possible starting with NATRSM93.

Example:

Examples for Starting the Roll Server as a Batch Job

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

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

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

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

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

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

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

Roll Server Messages, Condition Codes and Abend Codes

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

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

where *vrs* represents the relevant product version.

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

Condition Codes of the Roll Server Started Task

The following condition codes are used:

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

User Abend Codes

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

The following user abend codes are used:

Abend Code	Reason
U0200	`IXLCONN` failed.
U0201	`IXLFORCE` failed
U0202	`IXLLIST` failed.
U0203	`IXLDISC` failed.
U0204	`IXCLEAVE` failed.
U0301	`IXLLIST` failed.
U0302	`IXCMSGO` failed.

Abend Code	Reason
U0401	IXLLIST failed.
U0501	IXLLIST failed.
U0800	LXRES failed.
U0810	AXSET failed.
U0820	ETCRE failed.
U0821	ETCON failed.
U0830	IDENTIFY failed 1.
U0831	IDENTIFY failed 2.
U0832	IDENTIFY failed 3.
U0833	IDENTIFY failed 4.
U0834	IDENTIFY failed 5.
U0835	IDENTIFY failed 6.

Return Codes and Reason Codes of the Roll Server Request

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

Operating the Roll Server

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

Command	Description
HELP	Shows an overview of the available syntax.
DIAGNOSE	Debugging function, only to be used when advised by support. This command does not have any function. Its intended future use is in connection with special Zaps to aid in diagnosing specific customer problems, as the need arises.
HWM *n*	Sets the LRB high water mark to *n* times 10 percent of the number of LRB slots (*n*=0-10). HWM can not be lower than the current LWM. HWM is always ignored and reset to 0 if STRUCTURE has a value. If *n* is not specified, the current low and high water marks are displayed.

Command	Description	
LWM *n*	Sets the LRB low water mark to *n* times 10 percent of the number of LRB slots (*n*=0-9). LWM can not be higher than the current HWM. LWM is always ignored and reset to 0 if STRUCTURE has a value. If *n* is not specified, the current low and high water marks are displayed.	
RESET	Resets interval counters.	
SNAP	Debugging function. The Roll Server's address space is dumped to SYSUDUMP.	
STATS	Writes Roll Server statistics to JESMSGLG using the WTO macro (ROUTCDE=11). Statistics include information about roll-out and roll-in activity, as well as roll file I/O.	
TERMinate | STOP	Stops the Roll Server. The roll file directory and all modified LRB slots are written to the roll file and the address space is terminated. The address space ID is no longer available until the next IPL. Statistics are written to JESMSGLG using the WTO macro (ROUTCDE=11). Statistics include information about roll-out and roll-in activity, as well as roll file I/O.	
TIMEOUT	NAT *nnn*	Specifies or replaces the non-activity time parameter.
	TOC *hhmm*	Specifies or replaces the time of day of the timeout check.
	OFF	Disables timeout checking.
	ON	Reinstates timeout checking.
	NOW	Starts an immediate timeout check. Normal timeout check scheduling (if specified) remains in effect.
	TERSE	Suppresses the messages RSM0072 and RSM0074 during TIMEOUT NOW processing. Message RSM0047 Operator command: TIMEOUT NOW is also suppressed. Suppresses the message RSM0096 during a TIMEOUT process.
	VERBOSE	Displays the messages RSM0072, RSM0074 and RSM0047 Operator command: TIMEOUT NOW. Displays the message RSM0096 during a TIMEOUT process. This is the default setting.
	REPEAT *hhmm*	Specifies or replaces the interval at which a timeout check occurs.
	? (or no specification)	Displays current timeout settings. The question mark (?) is optional and can be omitted.

Resetting the Coupling Facility Structure

When a TP Monitor or server region which uses the Roll Server abends, the Roll Server might return an error. For example, the Roll Server directory might be reported as full because session cleanup had not been performed before the region abended. To resolve such an error, delete the respective Coupling Facility structure:

1. Shut down all Roll Servers which use the affected CF structure.
2. Issue the operator command `SETXCF FORCE,STR,STRNAME=`*`structure-name`*, where *`structure-name`* is the name of the CF structure used for the Roll Servers.
3. Restart all Roll Servers.
4. Format the Roll Files.

If, under normal operation, the Roll Files or the LRB become full due to the number of concurrently active users, restart the Roll server and specify larger Roll Files or a larger LRB.

You can also monitor the usage of Roll Files with the `SYSTP` utility.

Roll Server Performance Tuning

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

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

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

- The number of direct writes.

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

  There are two possible reasons:

  1. No LRB slot available. Increase the LRB.
  2. The compressed thread was larger than a single LRB slot. Increase the LRB slot size.
- The number of direct reads.

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

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

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

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

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

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

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

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

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

Roll Server User Exits

The roll server has two user exits.

- `NATRSU14`
- `NATRSU24`

Sample source modules are delivered for these.

NATRSU14 User Exit

Specifies the roll file number to be used.

Entry calling conventions:

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

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

where *vrs* represents the relevant product version.

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

Return calling convention:

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

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

NATRSU24 User Exit

Specifies the XCF group name to be used.

Entry calling conventions:

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

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

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

IV 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 (All Environments)	Contains general considerations that apply when running Natural in batch: Adabas data sets, sort data sets, subtasking session support for batch environments.

See also *Batch Mode* in the section *Profile Parameters Grouped by Category* (*Parameter Reference* documentation) for an overview of the Natural profile parameters that apply if Natural is used in batch mode.

16 Natural in Batch Mode under z/OS

- Natural z/OS Batch Interface 132
- Driver Parameters for z/OS Batch 132
- Data Sets Used by Natural in z/OS Batch Mode 132

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

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

- **Adabas Data Sets**
- **Sort Data Sets**
- **Subtasking Session Support for Batch Mode Environments**

Natural z/OS Batch Interface

The Natural z/OS batch interface consists of the `NATOS` object module which is linked to the Natural nucleus during the installation procedure for base Natural as described in the *Installation for z/OS* documentation.

You can customize the Natural z/OS batch interface to meet your requirements by changing the parameter settings in the `NTOSP` macro in the Natural parameter module during the appropriate installation step.

`NATOS` is fully reentrant and can run above the 16 MB line. Multiple Natural sessions can be started in parallel within one batch region; see ***Subtasking Session Support for Batch Environments***.

Driver Parameters for z/OS Batch

For information on the driver parameters that are available for z/OS in batch mode, refer to the description of profile parameter `OSP` or parameter macro `NTOSP` in the *Parameter Reference* documentation.

Data Sets Used by Natural in z/OS Batch Mode

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

Data Set	Explanation
CMEDIT	Software AG Editor Work File
CMHCOPY	Hardcopy Print Output
CMOBJIN	Input for Natural INPUT Statements
CMPLOG	Dynamic Profile Parameter Report Output
CMPRINT	Primary Report Output
CMPRMIN	Dynamic Profile Parameter Input
CMPRT*nn*	Additional Reports 01-31
CMSYNIN	Primary Command Input
CMTRACE	External Trace Output
NATRJE	Job Submit Output
STEPLIB	Load Library for External Modules
CMWKF*nn*	Work Files 01-32

These data sets are described below.

For sequential data output sets, the default DCB RECFM/LRECL information is as follows:

RECFM=FBA and LRECL=133

CMEDIT - Software AG Editor Work File

The Software AG Editor work file VSAM data set is required if a local or global Software AG editor buffer pool is to be used.

If not defined in the JCL, the name of the Editor work file specified by subparameter DSNAME of profile parameter EDBP or parameter macro NTEDBP is used by Natural to do the dynamic allocation for the Editor work file.

Alternatively, profile parameter EDPSIZE can be used to run with an auxiliary editor buffer pool, which doesn't require an editor work file. For more information about the installation of the Software AG editor, refer to *Installing the Software AG Editor on z/OS* in the *Installation for z/OS* documentation.

CMHCOPY - Optional Report Output for Hardcopy

The default name of the hardcopy print output data set is CMHCOPY. It can be changed by one of the following:

- the subparameter DEST of profile parameter PRINT for Print File 0,
- the profile parameter HCDEST, which is an equivalent of PRINT=((0),DEST=...),
- the setting of the system variable *HARDCOPY during the session,
- the terminal command %H during the session.

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

During the session, the hardcopy data set can be released and reallocated (before open or after close) by the by dynamic allocation (via application programming interface USR2021N, see *SYSEXT - Natural Application Programming Interfaces*).

CMOBJIN - Input for Natural INPUT Statements

This data set can be used to read data by the Natural INPUT statement rather than from the primary input data set CMSYNIN.

The usage of CMOBJIN is controlled by the profile parameter OBJIN. The input record data length for Natural is determined by profile parameter SL. The maximum record length (LRECL) supported is 255. The record format (RECFM) can be fixed or variable.

CMPLOG - Dynamic Profile Parameter Report Output

If profile parameter PLOG=ON is set and data set CMPLOG is available, the evaluated dynamic profile parameters are written to this data set during session initialization. If data set CMPLOG is not available, the evaluated dynamic profile parameters are written to CMPRINT.

CMPRINT - Primary Report Output

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

The record format (RECFM) for CMPRINT is FBA. If no DCB information for LRECL is available from the data set or from the JCL, LRECL=133 will be used as default. If no DCB information for BLKSIZE is available from the data set or from the JCL, BLKSIZE will be 10 times the value of LRECL as default.

If not defined in JCL, CMPRINT will be allocated dynamically as

```
//CMPRINT DD SYSOUT=*
```

when the first record is to be written.

CMPRMIN - Dynamic Parameter Data Set

CMPRMIN can be used as a **dynamic parameter** data set to overcome the length restriction for the character string in the job control PARM keyword of the EXEC statement.

If available, this file is read during session initialization to get the dynamic profile parameters.

All input records from CMPRMIN are concatenated into one parameter string. Only the first 72 positions of each CMPRMIN record are significant. Trailing blanks at the end of each record are truncated; if the last non-blank character is a comma, all trailing blanks are truncated, otherwise just one blank is left as delimiter; no commas are inserted.

Additional dynamic parameters can be supplied using the job control PARM keyword. They are concatenated at the end of the parameter string which was built from the input of CMPRMIN, that is, these can be used to overwrite the parameters from CMPRMIN.

A comment starts with a slash asterisk /* and ends with an asterisk slash */ and can be anywhere in your data set. They may span over several lines in your file.

CMPRTnn - Additional Reports 01 - 31

These data sets can be used by Natural print file statements like WRITE (*nn*). If no DCB information (for example, RECFM, LRECL, BLKSIZE) is available, the defaults are defined by the PRINT profile parameter or the NTPRINT macro in the Natural parameter module. The print file names can be overwritten by subparameter DEST.

CMSYNIN - Primary Command Input

This data set is used to read command input and data requested by the Natural INPUT statement. The latter is controlled by the profile parameter OBJIN (see also CMOBJIN).

The input record data length for Natural is determined by profile parameter SL. The maximum record length (LRECL) supported is 255. The record format (RECFM) can be fixed or variable.

CMTRACE - Optional Report Output for Natural Tracing

If profile parameter ETRACE=ON is set or the equivalent terminal command %TRE+ was issued, any Natural trace output during the session is written to the CMTRACE data set. To define the Natural components that are to be traced, the profile parameter TRACE is required.

If data set CMTRACE is not available, it will be allocated dynamically as

```
//CMTRACE DD SYSOUT=*
```

when the first trace record is to be written.

NATRJE - Job Submit Output

This data set is used for the Natural job submitting utility. If it is not defined, it will be allocated dynamically as

```
//NATRJE DD SYSOUT=(A,INTRDR)
```

when the first job is submitted.

STEPLIB - Load Library for External Modules

STEPLIB is the default load library name for loading external modules, for example:

- the environment-independent nucleus (profile parameter NUCNAME),
- a separate Adabas link routine module (profile parameter ADANAME),
- the session back-end program (profile parameter PROGRAM),
- any external subprograms not linked to the **Natural parameter module**.

The load library name can be changed by profile parameter LIBNAM. The specified load library name must be defined by a DD statement in the JCL.

CMWKFnn - Work Files 01-32

These data sets can be used by Natural work file statements like `READ WORK` *nn* and `WRITE WORK` *nn*.

If no DCB information (`RECFM`, `LRECL`, `BLKSIZE`, etc.) is available in the JCL or in the VTOC entry for the data set, the defaults are defined by the `WORK` profile parameter or the `NTWORK` macro in the **Natural parameter module**.

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

17 Natural in Batch Mode (All Environments)

- Adabas Data Sets 140
- Sort Data Sets 140
- Subtasking Session Support for Batch Mode Environments 140

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

Adabas Data Sets

Adabas data sets must be specified only in single-user mode. They are identical to those required for the execution of any normal application program using Adabas. See the relevant Adabas documentation for detailed information on Adabas data sets.

Sort Data Sets

Sort data sets must be specified if a Natural program containing a `SORT` statement is to be executed during the Natural session.

The requirements are identical to those for execution of a normal COBOL or PL/1 application program that invokes the operating system sort program and can vary according to the sort program in use.

Natural does not require the intermediate data sets `SORTIN` and `SORTOUT`, but communicates with the sort program via the `E15` and `E35` user-exit routine interfaces.

Subtasking Session Support for Batch Mode Environments

- Purpose
- Prerequisites
- Functionality
- Starting a Natural Session
- Starting a Subtask
- Accessing the User Parameter Area

Purpose

With subtasking support, you can run multiple Natural batch mode sessions within one address space. This allows parallel processing within one address space, rather than executing subsequent job steps, and can increase throughput dramatically.

Typically, client/server applications and products would take advantage of this functionality, for example, the Natural remote procedure call. Multiple server subtasks can be started to communicate with remote clients.

Prerequisites

If you wish to restart the Natural nucleus, it must be linked as a reentrant module (linkage editor option `RENT`).

The Adabas link routine (`ADALNK`) must be generated with reentrancy support.

Functionality

You start a subtask by issuing a `CALL` statement from a Natural program. The new Natural session ("subtask") is started with an extended front-end parameter list. This list contains up to three parameter sets:

- dynamic Natural profile parameters,
- startup parameters,
- user parameters.

Variable names for standard I/O data sets (for example `CMPRINT`) and other parameters for the batch mode interface startup can be passed from the starting program in the startup parameter area. Standard I/O data sets can be undefined or dummy data sets; they can be owned by one session or shared by multiple sessions.

Furthermore, a `CALL` interface is provided for reading the user parameter area with a Natural program.

Starting a Natural Session

Extended Parameter List

The Natural batch mode interface without extended parameter list gets initial control from the operating system using standard linkage call. Register 1 points to an address with high-order bit on as the last address indicator. This address points to a halfword field containing the length of the following parameter area.

The extended parameter list contains up to three parameter addresses. This is indicated by the high-order bit in the last address which can be the first, second or third address. All parameter addresses point to a halfword field containing the parameter length of the following parameter area. Zero length indicates that there is no parameter area.

- The first parameter area contains the dynamic profile parameters for the Natural session.
- The second contains special startup parameters for the initialization of the batch mode interface.
- The third contains a user parameter area which can be accessed during the Natural session.

Startup Parameter Area

When multiple batch mode Natural (sub)tasks are running in the same region, by default these sessions access the very same Natural standard I/O data sets (such as `CMPRINT`, `CMSYNIN`, etc),

as there are no Natural profile parameters available to set these file names. Also by default the Natural system variables `*INIT-ID` and `*INIT-USER` are identical because of their definition for batch mode.

In order to provide unique standard I/O data set names and unique IDs for Natural subtask sessions the startup parameters in the extended parameter list can be used to overwrite the Natural system defaults. The Startup Parameter area is a table of pairs of 8-character fields:

- The first entry contains the 8-byte keyword to be replaced,
- the second entry contains the 8-byte replacement value.

Keywords and replacement values must be padded with trailing blanks, if necessary.

The following keywords are valid:

`CMHCOPY`	Permanent hardcopy destination
`CMSYNIN`	Command input data set name
`CMOBJIN`	Object input data set name
`CMPRINT`	Standard output data set name
`CMPRMIN`	Dynamic parameter input data set name
`CMPLOG`	Dynamic parameter output data set name
`CMTRACE`	Trace output data set name
`INITID`	Job step name (system variable `*INIT-ID`)
`MSGCLASS`	Spool class for dynamic allocation of `CMPRINT` and `CMTRACE` (z/OS only)
`NATRJE`	Job submission data set name (z/OS only)
`STEPLIB`	Program load library name (see also profile parameter `LIBNAM`, Name of Load Library, z/OS only)
`SUBPOOL`	z/OS storage subpool (0 - 127, right justified)
`USERID`	Initial user identification (system variable `*INIT-USER`)

The usage of these entries is optional and no particular sequence is required. A blank value for a data set means that this data set is not available or is empty.

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 STRPARM1 (A16) INIT <'CMPRINT SYSPRINT'>
02 STRPARM2 (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.

V Natural Buffer Pools

This part contains information about the various storage management functions that are available to a Natural administrator under the operating system z/OS.

Natural Buffer Pool - General

Natural Global Buffer Pool under z/OS

For a functional overview of the Natural buffer pool, see *Natural Buffer Pool* in the Natural *System Architecture* documentation.

For an overview of the Natural profile parameters that affect the Natural buffer pools, see *Storage Management* in the section *Profile Parameters Grouped by Category* in the *Parameter Reference* documentation.

18 Natural Buffer Pool - General

- Natural Buffer Pool Principle of Operation 148
- Buffer-Pool Monitoring and Maintenance 153
- Natural Global Buffer Pool 156

The buffer pool is a storage area into which Natural programs are placed in preparation for their execution. Programs are moved into and out of the buffer pool as Natural users request Natural objects. Conceptually, it serves a function similar to that of an operating system in loading programs in and out of a reentrant area. The Natural buffer pool is an integral part of Natural in all supported environments.

Natural Buffer Pool Principle of Operation

Natural generates reentrant Natural object code. A compiled program is loaded into the buffer pool and executed from the buffer pool. Thus, it is possible that a single copy of a Natural program can be executed by more than one user at the same time.

This section covers the following topics:

- Objects in the Buffer Pool
- Directory Entries
- Text Pool
- Buffer Pool Hash Table
- Buffer Pool Initialization
- Buffer Pool Search Methods
- Local Buffer Pool
- Global Buffer Pool
- Buffer Pool Cache

Objects in the Buffer Pool

Objects in the buffer pool can be programs, subprograms, maps and global data areas. Global data areas are placed in the buffer pool only for compilation. In this case, two objects with the same name are loaded in the buffer pool: the GDA itself and the corresponding symbol table.

Directory Entries

When a Natural object is loaded into the buffer pool, a control block called a directory entry is allocated to this object.

A directory entry contains such information as the name of the object, what library it belongs to, what database ID and Natural system file number the object was retrieved from, and some statistical information (for example, the number of users who are concurrently executing the program at a given point in time).

When a user executes a program, Natural checks the directory entries to see if the program has already been loaded into the buffer pool. If it is not already in the buffer pool, a copy of the program is retrieved from the appropriate Natural system file and loaded into the buffer pool.

When an object is loaded in the buffer pool, one or more other Natural objects which are currently not being executed may be deleted from the buffer pool in order to make room for the newly loaded object. When the new object is loaded, a new directory entry is created in order to identify this object.

When an object is deleted from the system file, it will also be deleted from the buffer pool as soon as it is no longer being used. When an object is newly cataloged or stowed, its old version will also be deleted from the buffer pool as soon as it is no longer being used; when it is requested for execution again, the new version will then be loaded from the system file into the buffer pool.

Text Pool

The actual object code of a program that is loaded into the buffer pool is placed into an area called the text pool and must be allocated as a contiguous piece of memory within this text pool. This text pool is divided into a number of 4 KB buffers. This is an arbitrary size and can be changed at the Natural administrator's discretion. When an object is loaded, one or more text buffers that are contiguous to each other are allocated to store the object code of the object.

Buffer Pool Hash Table

This section applies to buffer pools of `TYPE=NAT` only.

To speed up the search time for looking up an object in the buffer pool directory, a hash table is used. The number of entries in the hash table is twice the number of directory entries, rounded up to the next prime number. This will ensure that only half of the table is filled at any point in time and that the probability of collisions is near zero. As a consequence, the average number of tests to find an existing object in the hash table is theoretically less than 2.

The hash criterion is the eight character long program name. If more than one program name is hashed to the same location in the hash table, an overflow technique resolves the collisions.

The storage required for the hash table is roughly 16 bytes per text block. Thus, the available storage in the text pool is reduced by between 1.6% (1 KB text blocks) and 0.1% (16 KB text blocks).

Buffer Pool Initialization

In case of a global buffer pool the initialization occurs during start of the global buffer pool.

In case of a local buffer pool the initialization time varies depending on the environment.

- In batch mode and TSO 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 (see also ***Tip*** in the section *Preload List*).

Buffer Pool Search Methods

As mentioned earlier and explained below, there are different search methods for allocating space in the buffer pool.

To select a search method, use

- The Natural profile parameters `BPMETH` and `BPI`.
  Or the macro `NTBPI` in the Natural parameter module.
  Or the function parameter `METHOD` of the global buffer pool.

For a description of these parameters and the macro `NTBPI` refer to the Natural *Parameter Reference* documentation.

Below is information on the search methods:

- `METHOD=S`
- `METHOD=N`
- **Choosing Search Methods**

METHOD=S

`METHOD=S` indicates that a selection process is used as search algorithm for allocating storage in the buffer pool in order to obtain the space required to accomplish a new load.

The selection process used is a combination of search Algorithms 1 and Algorithm 2:

- **Algorithm 1**
  Search Algorithm 1 attempts to find storage in the buffer pool that is either free space or space occupied by an unused (active but not used) object.

  If free space of the exact object size required is found, the selection process ends immediately. Otherwise, the search continues by browsing the buffer pool from top to bottom and comparing the entries in the buffer pool for best size fit. Additionally, in the case of unused objects, the search also considers the last attached time of the object, that is, the time the object was last referenced at a load or locate.

  When the selection process has finished, either free space or the space of an unused object with a size greater than or equal to the size requested is selected. The rule of precedence that applies to the search is: free space is taken first and space of unused objects next. In the case of unused objects, the oldest objects are removed first.

  If the selection process of Algorithm 1 was not successful, Algorithm 2 is invoked.

- **Algorithm 2**
  Search Algorithm 2 starts if Algorithm 1 fails. Algorithm 2 starts searching from a position in the buffer pool which is passed by Algorithm 1 and attempts to combine two or more entities (free storage and/or space occupied by unused objects) in order to obtain the necessary storage for a new load. However, the age of the object is not taken into account.

  Algorithm 2 continues processing to the bottom of the text record section and, if necessary, wraps around to the top of the text record section to make one final pass from top to bottom. If space is still unavailable, Algorithm 2 fails, the object cannot be loaded and Natural issues a corresponding error message.

METHOD=N

`METHOD=N` indicates that the next available free or unused space is used in order to obtain the space required to accomplish a new load. Unused space is space that is occupied by an active but not used object.

The search for the next available space starts from a pointer that moves through the buffer pool in a wrap-around fashion. Any next available buffer pool entries that are free or contain unused objects can be used and possibly chained together to obtain the amount of storage requested.

If the bottom of the buffer pool is reached during an allocation request, the pointer is wrapped around to the top of the buffer pool and one final search is performed through the buffer pool from top to bottom. If the bottom of the buffer pool is reached again and the object cannot be loaded, the load fails and Natural issues a corresponding error message.

`METHOD=N` can especially be considered for large buffer pools in combination with the **buffer pool cache** function. For details, see also Choosing Search Methods below.

Choosing Search Methods

By default, `METHOD=S` is used. The advantage of this method is, that a diligent search is performed to allocate space, taking into account the size and the age of objects and guaranteeing that the most dispensable unused objects are removed from the buffer pool to provide space for a new load.

A disadvantage of `METHOD=S` can be the high CPU time that is consumed by the selection process when browsing the buffer pool from top to bottom.

The advantage of `METHOD=N` is the short selection process and, usually, little browsing that require less CPU time for allocating space. This can be significant to large buffer pools.

The disadvantage of `METHOD=N` is that objects are selected less carefully for removal from the buffer pool. To avoid an increase in Adabas I/Os for reloading removed objects, we recommend that you use `METHOD=N` in combination with the **buffer pool cache** function.

Local Buffer Pool

Using Natural as supplied on the installation tape, you are running a *local* buffer pool. This is a buffer pool area that is allocated in the same partition (or region or address space) of the particular environment in use.

For example, in a batch or TSO environment, each user has his/her own local buffer pool. In a TP monitor environment such as Com-plete, CICS or IMS TM, there is one buffer pool per TP monitor from which all TP users execute.

Global Buffer Pool

In a z/OS environment, a global buffer pool is allocated from CSA or ECSA storage. In such an environment, all TSO users, batch users and TP monitor users could be executing from one common global area.

For further information on the global buffer pool, see ***Natural Global Buffer Pool***.

Buffer Pool Cache

This section applies to global and local buffer pools of `TYPE=NAT`.

The buffer pool cache is available in conjunction with global and local buffer pools. It is used only for Natural objects like programs, subprograms, maps, etc.

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.

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 as a data space or as an "above the bar" 64-bit common memory object.
When a data space is created for a buffer pool (by specifying `C64=N`), the ownership is assigned to the creator task. If the creator task terminates, the system automatically deletes the data space. Therefore, a creator task will remain active for at least as long as the buffer pool cache it owns is in use, even if the value specified for `RESIDENT` is `N`.

When a memory object is created for a buffer pool (by specifying `C64=Y`), the ownership is assigned to the system (not the creator task). Because of that, the memory object is not deleted when the creator task terminates. If you want to keep the creator task active after it executes its function, you have to specify `RESIDENT=Y`.

Note for Local Buffer Pools: (z/OS only, not for Com-plete 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)***. Thus, "all global buffer pools" in this context means "all global buffer pools within the same subsystem".

In some environments, it is important that changes which occur in one (local or global) buffer pool are also propagated to all other global buffer pools - that is, the same changes are also automatically made in the other global buffer pool - so as to ensure the consistency of the buffer pools and the Natural applications being used. This is particularly important in a z/OS Parallel Sysplex environment.

For example, if a Natural program is newly cataloged in one z/OS image, the propagation will cause the program to be deleted from all other global buffer pools in the z/OS Parallel Sysplex environment, so that its new version has to be loaded from the system file when the program is to be executed again.

The following changes are propagated:

- an object is deleted from the buffer pool,
- the buffer pool's blacklist is modified,
- the buffer pool is re-initialized.

Changes can be propagated to all other global buffer pools within the current z/OS image, or within the entire z/OS Parallel Sysplex environment, or all other global buffer pools of the same name within the z/OS Parallel Sysplex environment.

The propagation does not affect those objects in another global buffer pool which are defined as resident in that buffer pool.

The propagation is activated and its scope controlled by the Natural profile parameter `BPPROP`.

Note: As the propagation is performed asynchronously and an object is only deleted from a buffer pool when it is not longer being used, it may take some time until the current version of an object is available in all buffer pools.

Propagation to other *local* buffer pools is not possible.

Performance Considerations

For general advice on performance-related issues regarding the buffer pool and the BP cache, see *Performance Considerations* in the section *SYSBPM Utility - Buffer Pool Management* of the Natural *Utilities* documentation.

Natural Global Buffer Pool

The Natural global buffer pool is an optional Natural component.

It is available for the z/OS (refer to ***Global Buffer Pool under z/OS***)

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 *local* buffer pool.
`SUBSID`	Specifies the ID of the Natural subsystem to be used (see profile parameter `SUBSID`).

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.

19 Natural Global Buffer Pool under z/OS

- Using a Natural Global Buffer Pool 158
- Prerequisites 158
- Operating the Natural Global Buffer Pool 158
- Global Buffer Pool Manager Parameter Module 160
- Global Buffer Pool Operating Functions 161
- Global Buffer Pool Function Parameters 163
- Examples of NATBUFFER Specifications 170
- Sample NATGBPvr Execution Jobs 171
- Localization 173
- Messages 173

This document describes purpose and usage of a Natural global buffer pool (GBP) under the operating system z/OS.

Using a Natural Global Buffer Pool

Purpose

The Natural global buffer pool is a segment of storage assigned from the z/OS extended common system area (ECSA) above 16 MB (or from CSA storage below, if requested), used by Natural to load and execute Natural programs.

Benefits

Using a global buffer pool, multiple Natural sessions under different TP monitors (multiple copies of CICS, TSO, IMS TM, etc.) and/or in multiple batch sessions share the same area - thus requiring less storage than would be required for a local buffer pool in each environment.

Prerequisites

The following prerequisites must be met if you want to use a global buffer pool:

1. The module `NATGBPvr` must have been linked into an Authorized Program Facility (APF) library; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
2. The global buffer pool must have been started; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.

Operating the Natural Global Buffer Pool

The global buffer pool is operated using the program `NATGBPvr` which must be executed from within an Authorized Program Facility (APF) library.

The following topics are covered below:

- Allocation of the Natural GBP
- Setting up the Natural GBP
- Starting the Natural GBP Operating Program

- Stopping the Natural GBP Operating Program

Note: In the following document, *vrs* or *vr* represents the relevant version of the product. For information on product versions, see Version in the *Glossary*.

Allocation of the Natural GBP

If the z/OS parameter `ALLOWUSERKEYCSA(YES)` has explicitly been specified in `SYS1.PARMLIB(DIAG`*xx*`)`, a Natural global buffer pool is allocated in user key, so that Natural sessions accessing a global buffer pool have write permission for that buffer pool.

If `ALLOWUSERKEYCSA(NO)` is in effect, a Natural global buffer pool is allocated in system key; therefore, Natural sessions accessing a global buffer pool do not have any write permission for that buffer pool. These Natural sessions call the Authorized Services Manager (ASM) to perform all buffer pool functions. As a consequence, installation of the ASM is mandatory. The ASM is not only called to load a Natural object into the buffer pool but also to maintain the use count of a Natural object if the execution of this Natural object is started or terminated. The calls to the Authorized Services Manager will increase Natural's resource consumption. The overhead is hard to predict and depends on the application profile (ratio of program calls to program execution time).

Setting up the Natural GBP

The functions provided by the operating program `NATGBP`*vr* are activated in that they are

- specified in a parameter card (`PARM=`),
- read from a file (see below),
- or supplied by the `MODIFY` operator command.

`NATGBP`*vr* expects the first command in the parameter field (`PARM=`) of the `EXEC` statement.

You may enter:

- one of these **functions**
- or a reference to an input file with `CF=`*dd-name*, where *dd-name* represents a DD name defined in the JCL. Only "card image" files are supported, that is, `RECFM=F,LRECL=80`, and only the first 72 bytes of the input record are honored. Every record included from the input file represents a command. Blank records or records prefixed with an asterisk (*) are ignored. A file is processed until End-Of-File (EOF). Example: `PARM='CF=SYSIN1'`

If the parameter field is not supplied or blank, the commands will be read from file `SYSIN` by default.

It is only possible to enter one function at a time at the console or one function per line using the command file, otherwise an error message will be returned.

Each command received, from the parameter card, from file input or from operator console input is shown on the operator console.

Starting the Natural GBP Operating Program

To start the program `NATGBPvr`, either start a started task or submit a job, which executes `NATGBPvr`.

Important: To ensure that the global buffer pool is retained after a system failure, the global buffer pool should be started automatically during machine IPL.

Stopping the Natural GBP Operating Program

After all commands are processed, the program `NATGBPvr` terminates, unless

- `RESIDENT=Y` was specified
- or a buffer pool with a cache was created.

`NATGBPvr` will return one of the following condition codes:

Condition Code	Explanation
0	All functions executed successfully.
20	An error has occurred; see the message log for details. See also the `CC` function parameter of the global buffer pool.

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:

- HELP - Shows an overview of the available syntax
- 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.

HELP - Shows an overview of the available syntax

This function prints a list of the available syntax commands and, where applicable, the default values of the function parameters.

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.

Note: A data space Cache can only be deleted by the task which owns it. An "above the bar" memory object Cache can be deleted by any task.

FSHUT - Shut Down Global Buffer Pool

The global buffer pool is shut down and the storage area, including buffer pool and cache storage, 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 can be used to set global parameters. It does not perform any buffer pool operation.

REFRESH - Re-initialize Global Buffer Pool

With the `REFRESH` command it is possible to re-initialize an already active buffer pool. As no storage allocation takes place, the buffer pool size and location (above or below 16 MB) remain unchanged. However, it is possible to change the text-block size (see `NATBUFFER` parameter).

You should use this function only if the Current Use Count (see *Fields for Buffer Pool Objects* in *SYSBPM Directory Information*) is equal to zero (see warning below) or if the buffer pool has been destroyed.

Caution: If you re-initialize the buffer pool while Natural objects are being executed by active sessions in this buffer pool, the results of the active sessions are unpredictable and Natural may even abend.

SHOWBP - Show Existing Buffer Pools

Displays all buffer pools currently existing.

TERMINATE - Terminate GBP Operating Program

The GBP operating program is terminated. This termination does *not* affect any active global buffer pool.

ZAPS - Display Zaps Applied to GBP

Displays all Zaps applied to the global buffer pool operating program.

Global Buffer Pool Function Parameters

The functions of the Natural GBP operating program can be controlled with the aid of parameters. These parameters can be specified in any sequence. They can be abbreviated so that they are still unique.

Note: If you like to start multiple global buffer pools with an associated cache, you are recommended to use a single job or (under z/OS only) a single started task and to supply the different `CREATE` commands in an input data set. See ***Example 4*** in the section *Natural Global Buffer Pool under z/OS*.

The following parameters are available:

- BPNAME - Name of Global Buffer Pool
- BPLIST - Name of Preload List
- BPCSIZE - Buffer Pool Cache Size
- C64 - Type of Buffer Pool Cache Storage
- CC - Count Condition Code
- CONFIRM - FSHUT Confirmation
- IDLE - Wait Time before Check
- METHOD - Search Algorithm for Allocating Space in Buffer Pool
- NATBUFFER - Buffer Size, Mode, Text Block Size
- RESIDENT - Behavior after Function Execution
- SUBSID - Natural Subsystem ID
- TYPE - Type of Buffer Pool

BPNAME - Name of Global Buffer Pool

BPNAME=*value* is required (except for the TERMINATE function). It specifies the name of the global buffer pool to be created.

Value	Explanation
8 bytes	The name of the global buffer pool. **Note:** If the specified name is shorter than 8 bytes, blanks will be appended to it.
*	For the functions DELCACHE and FSHUT, you may supply a value of "*". FSHUT shuts down all the buffer pools for the specified natural subsystems. If sub parameter CONFIRM is set to Y, you must also provide additional confirmation. DELCACHE deletes all data space caches owned by the task from the specified natural subsystem. To delete caches located in "above the bar" memory, you must also provide a complete *BPNAME* and *SUBSID*.

BPLIST - Name of Preload List

BPLIST=*value* specifies the name of the preload list.

Value	Explanation
8 bytes	The name of the preload list. **Note:** If the specified name is shorter than 8 bytes, blanks will be appended to it.

BPCSIZE - Buffer Pool Cache Size

`BPCSIZE=value` specifies the amount of storage used to allocate a Buffer Pool Cache.

Value	Explanation
`100 - 2097148`	When sub parameter `C64=N`, this value is the amount of allocated storage (in KB) for a data space for the buffer pool cache. The specified value is rounded to the next 4KB boundary.
`100 - 58720256` (max 57344MB)	When sub parameter `C64=Y`, this value is the amount of allocated storage (in KB) for an "above the bar" memory object for the buffer pool cache. The specified value is rounded to the next 1MB boundary.

Notes:

1. The cache size can also be specified in units of MB or GB, for example, by specifying 10M for 10 MB.
2. If the `BPCSIZE` parameter is omitted (or set to zero), the buffer pool is not supplied with a cache.
3. A cache is only supported for buffer pools of `TYPE=NAT`.
4. The sub parameter `C64` decides whether the cache is created in the data space or in the "above the bar" 64 bit common memory.

C64 - Type of Buffer Pool Cache Storage

`C64=value` determines the type of storage for the buffer pool cache. The following values are possible:

Value	Explanation
`Y`	The storage for the buffer pool cache will be an "above the bar" memory object (in 64-bit memory).
`N`	The storage for the buffer pool cache will be a data space. This is the default value.

Note: This parameter is applicable only if `BPCSIZE` is provided.

CC - Count Condition Code

`CC=value` determines whether a condition code is ignored when returned by a command executed by the global buffer pool manager.

Value	Explanation
Y	The condition code returned after command execution is counted for the condition code of the `NATGBPvr` job. This is the default value.
N	The condition code returned after command execution is ignored. This may lead to a job response code of zero although the command execution failed.

Note: This parameter is valid for all commands.

Example of Command Execution:

The global buffer pool `NATGBP1` is stopped and restarted with the following command sequence:

```
FSHUT BPN=NATGBP1,S=NAT9,CONFIRM=N,CC=N
CREATE BPN=NATGBP1,S=NAT9,N=(1024),M=S,BPC=4096,I=60
```

The `FSHUT` command usually returns a condition code of 20 when it executes and the buffer pool is not active. However, with `CC=N` set, any condition code is ignored. In this case, a job response code greater than zero is only returned if the following `CREATE` command fails.

CONFIRM - FSHUT Confirmation

`CONFIRM=value` controls the `FSHUT` behavior if there are still active objects in the buffer pool.

Value	Explanation
Y	A confirmation for the `FSHUT` function is required from the operator. The operator can decide to abort or to force the `FSHUT` function. This is the default value.
N	`FSHUT` is forced without interaction with the operator.

Note: This parameter is only valid for the `FSHUT` command it has been specified with, that is, `CONFIRM` has to be specified with each `FSHUT` parameter, and it does not apply to subsequent `FSHUT` commands.

IDLE - Wait Time before Check

`IDLE=value` is ignored when the task does not own a buffer pool cache.

Value	Explanation
Numeric	The number of seconds to elapse before the GBP operating program checks for each buffer pool cache if its associated buffer pool is still active; if not, that buffer pool cache is released; when the last buffer pool cache owned by the task has been released, the task terminates, unless `RESIDENT=Y` has been specified.
`60`	This is the default value.

Notes:

1. `IDLE` is a "global" parameter. Once specified, `IDLE` will also apply to subsequent commands, without your having to specify it again.
2. Under z/OS, the GBP operating program also checks the specified `IDLE` time value against the job's timeout value: the specified `IDLE` time value internally may reduce `IDLE` to prevent timeout abends (S322).

METHOD - Search Algorithm for Allocating Space in Buffer Pool

`METHOD=value` controls which algorithm is to be used for allocating storage in the Natural buffer pool.

Value	Explanation
`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 is the default value.
`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.

Note: This parameter is only valid for the `CREATE` function. If you want to change the allocation method, restart the buffer pool.

NATBUFFER - Buffer Size, Mode, Text Block Size

`NATBUFFER=(size,mode,tsize)` specifies the size and the mode of the buffer pool, and the text block size.

Syntax	Value	Explanation
`NATBUFFER=(size,mode,tsize)`	*size*	is the amount of storage (in KB) to be allocated. The pool size can also be specified in units of MB, by specifying 10M for 10 MB. For the Natural buffer pool (`TYPE=NAT`): ■ The possible size is in KB (256–2097148) or in MB (1M –2047M). ■ The default size is 2MB. For the other buffer pools: ■ The possible size is in KB (100–2097148) or in MB (1M –2047M). ■ The default size is 256 KB. The specified amount of storage is always rounded up to a multiple of 4 KB. Next to the storage specified by *size*, one page (4 KB) of write protected storage will be allocated for administrative purposes.
	mode	determines if the global buffer pool is to be allocated above or below 16 MB. Possible values are: `XA` = above (default), `BL` = below.
	tsize	determines the text block size (in KB). Possible values are: `1`, `2`, `4`, `8`, `12`, and `16`. The default value is `4`.
size, *mode* and *tsize* have to be specified in the sequence shown above.		

Note: If `NATBUFFER` is not specified, the default values will be used. See also ***Examples of NATBUFFER Specifications***.

RESIDENT - Behavior after Function Execution

`RESIDENT=value` specifies the behavior of the GBP operating program after the specified function has been executed. The following values are possible:

Value	Explanation
`Y`	The GBP operating program will remain active after executing the specified function and await further commands. Once specified, `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.)
`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.
`A`	The GBP operating program automatically decides how to behave after having processed all commands. It will terminate if ■ no further command is available and ■ no buffer pool with an associated cache exists that was created by this task. 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. This is the default setting.

Note: `RESIDENT` is a "global" parameter. Once specified, `RESIDENT` will also apply to subsequent commands until explicitly specified/overwritten.

SUBSID - Natural Subsystem ID

`SUBSID=value` specifies the ID of the Natural subsystem.

Value	Explanation
4 bytes	The 4-byte ID of the Natural subsystem. Once specified, `SUBSID` will also apply to subsequent commands, without your having to specify it again. The default value is `NATv`, where *v* is the first digit of the current Natural version.
`NATv`	This is the default value. *v* is the first digit of the current Natural version.

Notes:

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

2. For the functions DELCACHE, FSHUT and SHOWBP, you may supply *SUBSID*="*" to process a given buffer pool, regardless of the subsystem. Alternatively, you may supply BPNAME="*" to process all buffer pools from all subsystems. In this case, the FSHUT function will prompt for an additional confirmation, regardless of the value of the CONFIRM sub parameter.
3. For further information on the Natural subsystem, see ***Natural Subsystem (z/OS)***.

TYPE - Type of Buffer Pool

TYPE=*value* specifies the type of the buffer pool. Possible values are:

Value	Explanation
NAT	Natural buffer pool (this is the default).
SORT	Sort buffer pool.
EDIT	Editor buffer pool.
MON	Monitor buffer pool.
RNM	Review Natural Monitor buffer pool.

Examples of NATBUFFER Specifications

The following examples refer to the NATBUFFER parameter which is used to set buffer size, mode and text block size, the parameter name being abbreviated (N).

Example 1: To allocate a global buffer pool above 16 MB, with a size of 1 MB and a text block size of 1 KB, you specify:

```
N=(1000,,1)
```

or

```
N=(1M,,1)
```

Example 2: To allocate a global buffer pool above 16 MB, with a size of 10 MB and a text block size of 4 KB, you specify:

```
N=(10000)
```

or

```
N=(10M)
```

Example 3: To allocate a global buffer pool above 16 MB, with a size of 256 KB and a text block size of 4 KB, you specify:

```
N=(,,)
```

This is equivalent to omitting the `NATBUFFER` parameter altogether, as it causes the default values to apply.

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* represents the relevant product version. For information on product versions, see Version in the *Glossary*.

Example 1:

```
//GBPSTART JOB
//*
//* Starts a global buffer pool with the name NATvrGBP, a size of 1 MB and
//* a text block size of 4 KB. The global buffer pool is allocated above 16 MB.
//* The subsystem used is NATv.
//* After the allocation, the job GBPSTART terminates.
//*
//STEP EXEC PGM=NATGBPvr,PARM='BPN=NATvrGBP,N=(1M)'
//SETPLIB DD DISP=SHR,DSN=USER.APF.LINKLIB
```

Example 2:

```
//GBPRES JOB
//*
//* Starts a global buffer pool with the name GBP, a default size of
//* 100 KB and a text block size of 1 KB. The global buffer pool is allocated
//* below 16 MB. The subsystem used is SAGS.
//* After the allocation, the job GBPRES will wait for further commands.
//* Further commands may be entered using the MODIFY operator command:
//* F GBPRES,command-string
//*
//STEP EXEC PGM=NATGBPvr,PARM='BPN=GBP,N=(,BL,1),S=SAGS,R=Y'
```

Example 3:

```
//GBPSTOP
//*
//* Stops the global buffer pool GPB if it contains no active objects. If it
//* does contain active objects, the operator console will prompt for a reply.
//* Depending on the reply, the shutdown will be forced (Y) or aborted (N).
//* The subsystem used is NATv.
//*
//STEP EXEC PGM=NATGBPvr,PARM='FSHUT,BPN=GPB'
```

Example 4:

```
//GBPSTRT2
//* Read commands from SYSIN1:
//*
//* Start two global buffer pools (subsystem ID Nvrs) with names
//*   NATGBP1 - size=1024KB and a cache with size 2048KB, and
//*   NATGBP2 - size=2048KB without cache.
//* Display all buffer pools of subsystem ID Nvrs.
//*
//* Note: The job does not terminate by itself, but stays resident and waits
//*       for operator commands because it owns the data space allocated for
//*       buffer pool NATGBP1.
//*
//* To shut down the buffer pools, send the operator command MODIFY with
//* parameter CF=SYSIN2 to execute the corresponding FSHUTs.
//*
//STEP EXEC PGM=NATGBPvr,PARM='CF=SYSIN1'
//SYSIN1  DD *
CREATE,BPN=NATGBP1,S=Nvrs,N=(1M),BPC=2M
CREATE,BPN=NATGBP2,S=Nvrs,N=(2M)
SHOWBP S=Nvrs
//SYSIN2 DD *
FSHUT,BPN=NATGBP1,S=Nvrs
FSHUT,BPN=NATGBP2,S=Nvrs
SHOWBP S=Nvrs
//*
```

Localization

The module NATGBPTX is delivered in source form. It contains all error messages in English in mixed case. The messages can be translated into other languages as required. In this case, the "new" NATGBPTX source module has to be assembled and the module NATGBP*vr* has to be relinked.

To issue the global buffer pool messages including their variable parts in upper case, the global buffer pool parameter module NATGBPRM has to be assembled with the UCTRAN parameter set to YES, and the module NATGBP*vr* has to be relinked.

To relink the module NATGBP*vr*, use the following JCL:

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

Messages

Refer to *Natural Global Buffer Pool Manager Messages* in the Natural *Messages and Codes* documentation

VI Message Buffer Pool

20 Message Buffer Pool

- Purpose 178
- Prerequisites 178
- Operating the Message Buffer Pool 178
- Sample NATMBPvr Execution Jobs 180
- Message Buffer Pool Operating Functions 181
- Function Parameters 182
- Messages 184

This part describes the use of the message buffer pool.

Note: The message buffer pool is available under z/OS.

Purpose

The message buffer pool is a cache memory which is used to store the Natural system messages and the user texts.

Before an error message is output, Natural first checks whether the corresponding message text is available in the message buffer pool. If so, this text is output. Otherwise, the error message would be read from the database, and would be stored in the message buffer pool.

The message buffer pool is available only as a global buffer pool. Its use is optional, and is controlled by the Natural profile parameter `BPI` or the corresponding macro `NTBPI`. When used, the message buffer pool is allocated in a data space.

Prerequisites

The following prerequisites must be met if you want to use the message buffer pool:

1. The module `NATMBP`*vr* must have been linked into an Authorized Program Facility (APF) library; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
2. The message buffer pool must have been created and started; see the corresponding step in *Installing Natural on z/OS* in the *Installation for z/OS* documentation.
3. The keyword subparameter `TYPE` of profile parameter `BPI` or macro `NTBPI` must be set to `MSG`.

Operating the Message Buffer Pool

The message buffer pool is operated by the program `NATMBP`*vr* which must be executed from within an Authorized Program Facility (APF) library.

The following topics are covered below:

- Setting up the Message Buffer Pool
- Starting the Message Buffer Pool Operating Program

- Stopping the Message Buffer Pool Operating Program

Note: In the following document, *vrs* or *vr* represents the relevant version of the product. For information on product versions, see *Version* in the *Glossary*.

Setting up the Message Buffer Pool

The **functions** available from NATMBP*vr* (see also ***Function Parameters***) are activated in that they are

- provided by a parameter card (PARM=),
- read from a file (see below),
- or supplied by the MODIFY operator command unless NATMBP*vr* has not been terminated.

NATMBP*vr* expects the first command in the parameter field (PARM=) of the EXEC statement.

You may enter:

- one of the functions described in the section ***Common Message Buffer Pool Operating Functions***,
- or a reference to an input file with CF=<*dd-name*>, where <*dd-name*> represents a DD name defined in the JCL.

  Only "card image" files are supported; that is, RECFM=F,LRECL=80, and only the first 72 bytes of the input record are honored.

  Every record included from the input file represents a command.

  Blank records or records prefixed with an asterisk (*) are ignored.

  A file is processed until End-Of-File (EOF).

  Example: PARM='CF=SYSIN1'

If the parameter field is not supplied or blank, the commands will be read from file SYSIN by default.

It is only possible to enter one function at a time at the console, or one function per line using the command file, otherwise an error message will be returned.

Each command received from parameter card, from file input or from operator console input is displayed on the operator console.

Starting the Message Buffer Pool Operating Program

To start the program NATMBP*vr*, either start a started task or submit a job which executes NATMBP*vr*.

Stopping the Message Buffer Pool Operating Program

The program NATMBP*vr* is stopped by using the TERMINATE function (see ***Common Message Buffer Pool Operating Functions***) or, in case of emergency, by using the CANCEL operating program.

Sample NATMBPvr Execution Jobs

The following examples show sample batch jobs for creating and terminating a global buffer pool.

- Example 1
- Example 2
- Example 3

Note: In the following examples, *v*, *vrs* or *vr* represents the relevant version of the product. For information on product versions, see *Version* in the *Glossary*.

Example 1

```
//MBPSTART JOB
//*
//* Starts a message buffer pool with the name NATvrMBP and
//* a size of 10 MB.
//* The subsystem used is NATv.
//*
//STEP EXEC PGM=NATMBPvr,PARM='BP=NATvrMBP,SI=10'
//SETPLIB DD DISP=SHR,DSN=USER.APF.LINKLIB
```

Example 2

```
//MBPRES JOB
//*
//* Starts a message buffer pool with the name MBP and a default size of
//* 100 MB. The subsystem used is SAGS.
//*
//STEP EXEC PGM=NATMBPvr,PARM='BP=MBP,S=SAGS'
```

Example 3

```
//MBPSTRT2
//* Read commands from SYSIN1:
//*
//* Start 2 message buffer pools (subsystem ID Nvrs) with name
//*   NATMBP1 - size=1000MB
//*   NATMBP2 - size=2000MB
//* If the buffer pools should shut down, send operator command MODIFY with
//* parameter "CF=SYSIN2" to execute the corresponding FSHUTs.
//*
//STEP EXEC PGM=NATMBPvr,PARM='CF=SYSIN1'
//SYSIN1  DD *
CREATE,BP=NATMBP1,S=Nvrs,SI=1000M
CREATE,BP=NATMBP2,S=Nvrs,SI=2000M
SHOWBP S=Nvrs
//SYSIN2 DD *
FSHUT,BP=NATMBP1,S=Nvrs
FSHUT,BP=NATMBP2,S=Nvrs
//*
```

Message Buffer Pool Operating Functions

The following functions are available:

- HELP - Shows an overview of the available syntax
- CREATE - Create a Message Buffer Pool
- FSHUT - Shut Down Message Buffer Pool
- TERMINATE or STOP- Terminate Message Buffer Pool Operating Program
- ZAPS - Display Zaps Applied to Message Buffer Pool

Note: The function names can be abbreviated. It is sufficient to use the first character only, for example `T` for `TERMINATE`.

HELP - Shows an overview of the available syntax

This function prints a list of the available syntax commands and, where applicable, the default values of the function parameters.

CREATE - Create a Message Buffer Pool

This function creates a message buffer pool with the specified **parameters**.

FSHUT - Shut Down Message Buffer Pool

The message buffer pool with the specified **parameters** is deallocated.

TERMINATE or STOP- Terminate Message Buffer Pool Operating Program

The message buffer pool operating program is terminated. Prior to that, all active message buffer pools are deallocated.

ZAPS - Display Zaps Applied to Message Buffer Pool

Displays all Zaps applied to the message buffer pool operating program.

Function Parameters

The functions of the message buffer pool operating program can be controlled with the aid of parameters. These parameters can be specified in any sequence. They can be abbreviated.

The following parameters are available:

BPNAME	**Name of message buffer pool**.
BPLIST	**Name of the preload list (optional)**.
SUBSID	**Natural subsystem ID**.
SIZE	**Size of the message buffer pool**.

Note: The underlined part of the parameter name marks the shortest possible abbreviation.

BPNAME - Name of Message Buffer Pool

BPNAME=*value* specifies the name of the message buffer pool to be created.

Value	Explanation
8 bytes	The name of the message buffer pool. **Note:** If the specified name is shorter than 8 bytes, blanks will be appended to it.
`MTBP`	This is the default value.

BPLIST - Name of Preload List

`BPLIST=value` specifies the name of the optional preload list.

Value	Explanation
8 bytes	The name of the preload list. **Note:** If the specified name is shorter than 8 bytes, blanks will be appended to it. There is no default value.

SUBSID - Natural Subsystem ID

`SUBSID=value` specifies the ID of the Natural subsystem.

Value	Explanation
4 bytes	The ID of the Natural subsystem. **Note:** If the specified name is shorter than 8 bytes, blanks will be appended to it.
`NATv`	This is the default value, where *v* is the first digit of the current Natural version.

SIZE - Size of Message Buffer Pool

`SIZE=value` specifies the size of the message buffer pool.

Value	Explanation
1 - 2000 MB	The size of the message buffer pool.
`1`	This is the default value.

Note: A 1 MB Message Buffer Pool can hold up to 7000 error messages.

Messages

Refer to *Message Buffer Pool Messages* in the *Messages and Codes* documentation.

VII System Spool Access

21 System Spool Access

- Purpose 188
- Prerequisite 188
- Using the Write-to-Spool Feature 188

This document describes the Write-to-Spool feature for Natural.

See also *Installing and Activating the Write-to-Spool Feature* in the section *Installing Entire System Server Interface on z/OS* in the *Installation for z/OS* documentation.

Purpose

The Write-to-Spool feature enables Natural users to write reports to the system spool directly. It can be used in any Natural environment (Com-plete, TSO, CICS, IMS TM, batch, etc.) and uses the Entire System Server view `WRITE-SPOOL`.

Under z/OS, the `SYSOUT` is part of the Entire System Server job stream within the JES spool, and it may be processed by any software which expects output in JES Spool, for example, Entire Output Management. The JES spool may be a JES2 or a JES3 spool.

Prerequisite

To use the Write-to-Spool feature, the Entire System Server needs to be installed.

Using the Write-to-Spool Feature

The Write-to-Spool feature is handled by a so called "access method", which is called `ESS` for Entire System Server. You may define your printer in the **Natural parameter module** or dynamically in your session parameters.

Defining Your Printer

To define your printer

1 Define the printer in the **Natural parameter module**.

Use the `NTPRINT` macro to specify the printer number (*n*) and the access method (`AM`):

```
NTPRINT (n),AM=ESS
```

Example:

```
NTPRINT (1,3),AM=ESS
```

In this example, the printers 1 and 3 are defined for use with access method `ESS` (Entire System Server).

Or:

Define the printer during session startup by specifying the profile parameter `PRINT`, for example:

```
PRINT=((1-6),AM=ESS)
```

In this example, the printers 1 to 6 are defined for use with access method `ESS` (Entire System Server).

2 Link the access-method modules to the Natural nucleus.

See the platform-specific *Installation* documentation.

Or:

Load it dynamically by specifying the following profile parameters `RCA` and `RCALIAS`:

```
RCA=(NATAM11),RCALIAS=(NATAM11,NATPWSAM)
```

where `NATPWSAM` is the delivered write-to-spool module containing the default parameters.

If you have linked a module with adapted parameters, use the name of this module instead.

3 Define the JES destination with the `OUTPUT` option of the `DEFINE PRINTER` statement. You can use one of the following examples depending on whether you want to send the output to a spool file, a local JES printer, or through a remote JES node to a remote user or device.

Example:

```
DEFINE PRINTER (n) OUTPUT 'LOCAL' /* For printing on local JES/POWER printers
```

Or:

```
DEFINE PRINTER (n) OUTPUT 'DAEF' /* For printing to JES spool called DAEF
```

Or:

```
DEFINE PRINTER (n) OUTPUT 'DEST=node-name,REMOTE-USERID=user-id'  /* For printing ↩
to remote JES nodes
```

where:

- *n* is the number in the NTPRINT entry in the **Natural parameter module** described in **Step 1**.
- *node-name* is the name of the remote JES node.
- *user-id* the ID of the user or device who receives the output.

Reports can now be written to the system spool using one of the following statements:

```
DISPLAY (n)
```

or

```
PRINT (n)
```

where *n* is the number in the NTPRINT entry in the **Natural parameter module** in **Step 1**.

Users can set the output format and number of copies using the FORMS and COPIES clauses of the DEFINE PRINTER statement.

Example:

```
DEFINE PRINTER (2) OUTPUT 'DEST'
                   FORMS 'FORM'
```

The defaults for items such as Entire System Server node, forms and output class can be found in the module NATWSPDF.

Examples for z/OS

Example 1

Assume using the factory settings and executing the Natural program:

```
DEFINE PRINTER (2) OUTPUT 'WK1'
WRITE (2) 'THIS IS A SMART RECORD'
CLOSE PRINTER (2)
```

During the execution of this program, you can see the following fields with their values in the **Display Active Tasks** panel:

```
DDNAME     DSID Owner C Dest Rec-Cnt Forms Wtr PageDef FormDef
SYS00001   104  WKK   A WK1     2     STD
```

Browsing this data set, you can see:

```
Page 1
THIS IS A SMART RECORD
```

Example 2

Assume using the default member:

Parameter	Explanation (Possible Values)
WSPDFLT NODE=55526,	Entire System Server target node number (5 characters at maximum)
PROGRAM=HUGO,	JES writer (8 characters at maximum)
CLASS=Y,	SYSOUT class (1 character)
HOLD=YES,	Hold (YES or NO)
CNTL=A,	Carriage control (A or M)
FORM=WOFO,	Form (4 characters at maximum)
RMT=JESWOLF,	JES remote (8 characters at maximum)
FORMDEF=FOWOLF,	Form definition (6 characters at maximum)
PAGEDEF=PAWOLF	Page definition (6 characters at maximum)

Execute the following Natural program:

```
DEFINE PRINTER (2) OUTPUT 'WK1'
WRITE (2) 'THIS IS A SMART RECORD'
CLOSE PRINTER (2)
```

During the execution of this program, you can see the following fields with their values in the **Display Active Tasks** panel:

```
DDNAME   DSID Owner C Dest Rec-Cnt Forms  Wtr   PageDef  FormDef
SYS00002 105   WKK  Y  WK1    2     WOFO  HUGO  PAWOLF   FOWOL
```

Browsing this data set, you can see:

```
Page 1
THIS IS A SMART RECORD
```

Example 3

Assume using the default member:

Parameter	Explanation (Possible Values)
`WSPDFLT NODE=55526,`	Entire System Server (NPR) target node (node number)
`PROGRAM=*OUTPUT,`	JES writer (8 characters at maximum)

The other parameters in the default member are not changed.

Run the following example program:

```
DEFINE PRINTER (2) OUTPUT 'KURT'
PRINT (2) ' here comes KURT'
CLOSE PRINTER (2)
```

After that, Entire System Server fetches the value from the field `OUTPUT` in the `DEFINE PRINTER` statement and inherits it as the JES writer attribute for the specific spool data set.

Looking in TSO/SDSF under the job name of Entire System Server, you can see the following:

```
PREFIX=NPR*  DEST=(ALL)  OWNER=*  SYSNAME=
NP   DDNAME   Time     Forms    FCB  UCS  Wtr      Flash
     SYS00005 10:20:48          **** **** KURT     ****
```

If in JES an associated JES writer program is defined, it gets control and handles this output as defined in the program.

VIII Natural 3GL CALLNAT Interface

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

Natural 3GL CALLNAT Interface - Purpose, Prerequisites, Restrictions

Natural 3GL CALLNAT Interface - Usage, Examples

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

- Purpose of 3GL CALLNAT Interface 196
- Prerequisites 196
- Restrictions 198

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

Purpose of 3GL CALLNAT Interface

With the 3GL CALLNAT interface, Natural enables 3GL programs to invoke and execute Natural subprograms.

The 3GL can be any programming language which supports the standard linkage call interface. In most cases this will be a COBOL program, but the functionality can also be used by, for example, PL/1, FORTRAN, C or Assembler programs.

Availability

The interface is available in batch mode under z/OS and for the following TP-monitor environments:

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

Prerequisites

This section describes the prerequisites to execute a Natural subprogram from a 3GL program, using an internal `CALLNAT` statement. To achieve the desired functionality, a Natural environment must be set up before you execute the `CALLNAT` interface from your 3GL program.

- Space Requirements
- Linking

- Environment Dependencies

Space Requirements

The mechanism of parameter addressing in a Natural program requires that the parameters passed reside in an area allocated by Natural, that is, in any of its sizes. The 3GL program, however, allocates the storage for its variables somewhere in the address space of the task. To make addressing still successful, a "call-by-value" mechanism is used for those variables which do not already reside in a Natural area. This means that, prior to invoking the Natural subprogram, the parameters to be passed are transferred into a Natural area, namely the `DATSIZE` buffer.

In addition to the storage used for the contents of the variables, additional storage will be needed depending on the number of parameters. The total amount of space required is approximately the same as the space that would be needed in the `DATSIZE` buffer if the subprogram-invoking program were coded in Natural.

Linking

To invoke the Natural subprogram, the 3GL program must call the `CALLNAT` interface. Depending on the power and functionality of the call interface of the 3GL programming language, the `CALLNAT` interface can be either placed in an accessible load library for dynamic loading or linked to the 3GL program.

It is recommended, whenever possible, to load the `CALLNAT` interface dynamically from a Natural steplib, as this method makes sure that always the most recent version of that program is used.

The samples `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:

Interface Module	Purpose
NATXCAL	To be used in the following cases: ■ if the 3GL module is either loaded dynamically or linked to Natural and then invoked by a branch and link instruction (batch, Com-plete, IMS TM, TSO, %P=S and %P=LS in CICS). ■ if the 3GL module is called via the INTERFACE4 option of the CALL statement. It provides the INTERFACE4 Natural Callnat Interface as well as the INTERFACE4 Callback Functions. For further information on the INTERFACE4 functionality, see the CALL statement documentation. **Note:** For CALL INTERFACE4 purposes, NATXCAL cannot be loaded dynamically but must be linked to the 3GL program.
NCIXCALL	To be used in a CICS environment if the 3GL module has been invoked using EXEC CICS LINK; NCIXCALL is delivered in source code to be compiled with your CICS macros. See also *Installing the Natural CICS Interface on z/OS* in the *Installation* documentation.
NCIXCPRM	To be used in a CICS environment to build the parameter address list used as COMMAREA for the subsequent EXEC CICS LINK 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**
  IMS TM 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).

23 Natural 3GL CALLNAT Interface - Usage, Examples

- Usage 202
- Sample Environments 206

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

Usage

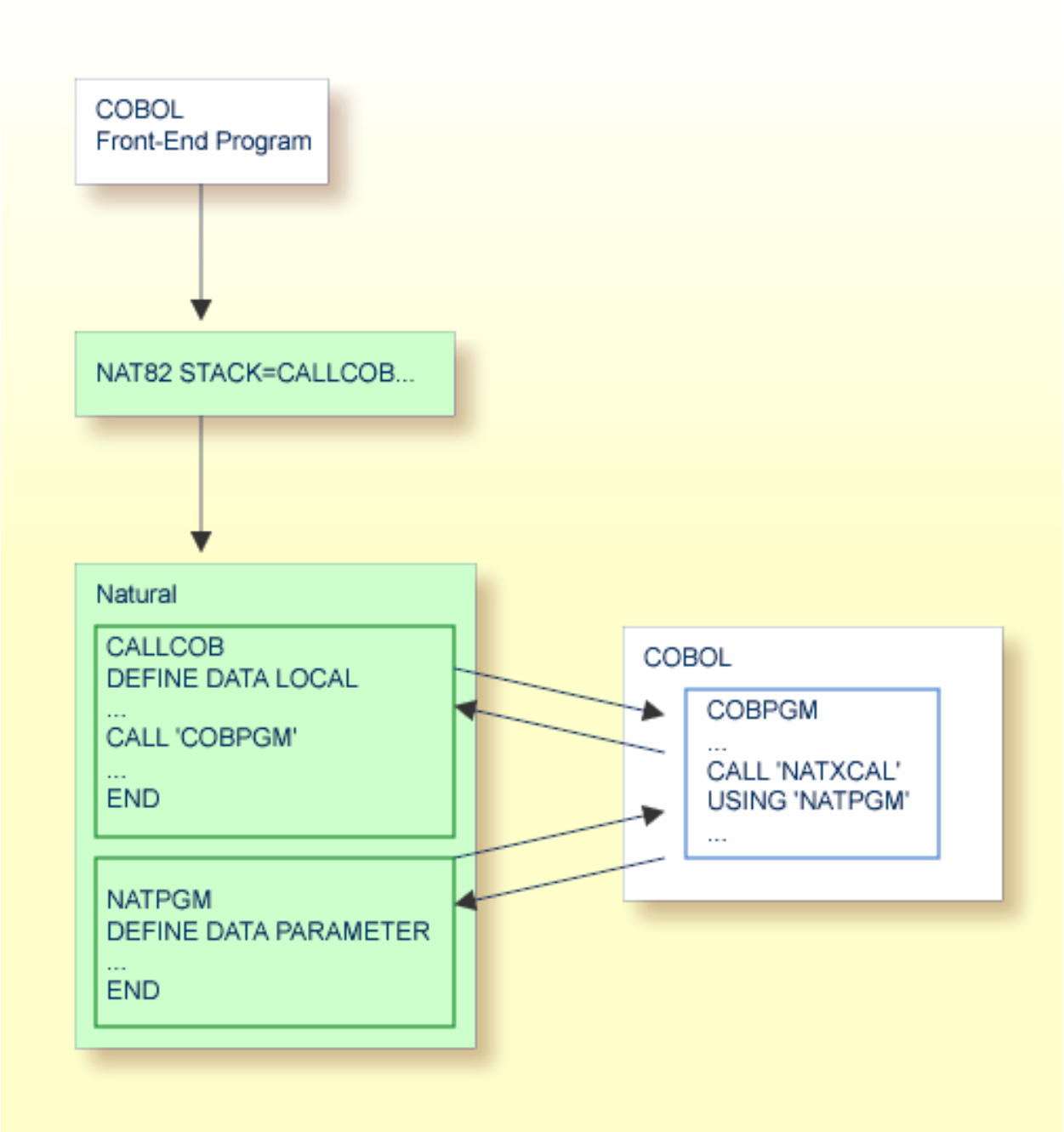
The following topics are covered:

- Overview
- Call Structure
- Parameter Handling

Overview

When you invoke a Natural subprogram from a 3GL program, a Natural session must be active, i.e. the 3GL program itself must be called by Natural.

Therefore you must take special precautions if you do not want the Natural layer to show up. The following figure is intended to give you an overview of how an application using the Natural 3GL CALLNAT interface may be designed in such a case:



The necessary environment is established by first invoking a Natural start-up program. By using the Natural `CALL` statement, this start-up program can then invoke a 3GL program from where you can invoke the `CALLNAT` interface.

Call Structure

The Natural main program is very simple; it only calls, for example, a COBOL program:

```
.....
CALL 'COBPGM'
END
.....
```

The `CALL` statement of the 3GL programming language (for example, COBOL) must have access to the Natural 3GL CALLNAT interface, which then invokes the Natural subprogram:

```
.....
CALL 'interface' USING natpgm p1 ... pn
.....
```

The parameter *interface* is environment-dependent (for example, `NATXCAL`) and linked to the calling program. The parameter *natpgm* must be an alphanumeric variable of 8 bytes that contains the name of the Natural subprogram to be invoked. The parameters *p1 ... pn* are passed to the Natural subprogram.

Example (for all environments except CICS):

The COBOL program `COBPGM` could contain coding similar to the following one:

```
.....
MOVE 'FINDNPGM' TO natpgm
CALL 'interface' USING natpgm number name
IF natpgm NE 'FINDNPGM'
THEN GOTO error_handling_1
.....
```

The invoked Natural subprogram `FINDNPGM` calculates the number of persons in the file `EMPLOYEES` with *name* equal to a value passed from the COBOL program:

```
DEFINE DATA
PARAMETER
1 pnumber (P10)
1 pname (A20)
LOCAL
1 emp VIEW OF employees
END-DEFINE
*
RESET presp
FIND NUMBER emp WITH name=pname
MOVE *NUMBER TO pnumber
ESCAPE ROUTINE
```

If an error occurs while the subprogram is executed, information about this error will be returned in the variable *natpgm* in the form *NAT*nnnn*, where *nnnn* is the corresponding Natural error number.

Example (for CICS only):

Under CICS, the call of a Natural subroutine from, for example, COBOL should be as follows:

```
...
WORKING STORAGE SECTION
...
01 PARM-LIST PIC X(132).
01 NATPGM PIC X(8).
01 NUMBER PIC 9(10) comp-3.
01 NAME PIC X(20).
...
PROCEDURE DIVISION
...
MOVE 'FINDNPGM' TO NATPGM
CALL 'NCIXCPRM' USING PARM-LIST NATPGM NUMBER NAME ...
EXEC CICS LINK PROGRAM('NCIXCALL')
COMMAREA(PARM-LIST) LENGTH(132) END-EXEC.
...
```

The called subroutine NCIXCPRM builds the parameter address list used as COMMAREA in the subsequent EXEC CICS LINK command.

Parameter Handling

There is no format and length checking. It is the caller's responsibility to pass a correct parameter list. The number, format and length of the parameters are defined by the invoked Natural subprogram.

When you are passing parameters, group arrays should not be passed, since they are resolved as individual arrays:

Example of Invalid Syntax:

```
.....
01 GROUP (1:2)
02 F1
02 F2
.....
.....
CALL ..... F1 F2
.....
```

Example of Valid Syntax:

```
.....
01 F1 (1:2)
01 F2 (1:2)
....
.....
CALL ..... F1 F2
.....
```

Arrays with dynamic ranges cannot be used as parameters.

Sample Environments

The objective for the sample 3GL CALLNAT environments below is to demonstrate how a COBOL routine can call a Natural subprogram under specific TP-monitor systems or in batch mode, and to give system-specific instructions to create such environments.

The following topics are covered:

- Sample Environment for CICS
- More Samples
- Sample for Any Other Supported Environment

Sample Environment for CICS

Perform the following steps to create a sample Natural 3GL CALLNAT environment under CICS:

Step 1: Create the Environment Initialization

- Set up the front-end program that initializes the 3GL CALLNAT environment.
- Use the COBOL front-end `XNCIFRCX` in the Natural/CICS source library. It starts Natural, stacks `LOGON YOURLIB` and executes the program `TSTCOB`, which initializes the Natural 3GL CALLNAT environment.
- Locate the string `NCvr` (where *vr* represents the relevant product version) in the source code and replace it with the valid transaction ID for Natural.
- Compile and link-edit the COBOL program and define program to CICS via `CEDA DEFINE PROGRAM`.

Step 2: Install the Sample COBOL Call

Provided in the Natural/CICS source library `NCI.SRCE` is the sample member `XNCI3GC1`, which contains a default call to the Natural subprogram `MYPROG`.

- For test purposes, create the following program in the library `SYSTEM` and stow it as:

```
WRITE 'BEFORE PGM EXECUTION'
CALL 'COBNAT'
WRITE 'AFTER PGM EXECUTION'
END
```

- Look at the XNCI3GC1 source and review the CALL and LINK. Compile and link as COBNAT with the following CICS INCLUDE directives or use Step 2 of the Sample Job NCTI070:

```
INCLUDE CICSLIB(DFHECI)
INCLUDE XNCI3GC1                    <= output from translator and compiler
INCLUDE NCILIB(NCIXCPRM)
ENTRY XNCI3GC1
NAME COBNAT(R)
```

Step 3: Create a Sample Natural Subprogram

By default, the source member XNCI3GC1 is set up to call the Natural subprogram MYPROG in the library YOURLIB. The program TSTCOB, as mentioned above, starts up the process by calling COBNAT that contains the actual call to the Natural subprogram MYPROG.

Create the subprogram MYPROG to demonstrate the executing Natural subprogram.

```
DEFINE DATA PARAMETER
  01 PARM1 (A18)
  01 PARM2 (A18)
  01 PARM3 (A18)
END-DEFINE
*
  MOVE 'PARAM01' TO PARM1
  MOVE 'PARAM02' TO PARM2
  MOVE 'PARAM03' TO PARM3
END
```

Step 4: Verify the CICS Resources

- Use the job NCII005 for a guide to defining the CICS resources (PPT and PCT).
- Define the required CICS resources (PPT and PCT).

Step 5: Test the Environment

Test the environment by using the NCYC default transaction. Use CEDF to monitor the program control and observe the data areas in use.

Important: Since Natural is at the top of the CICS program hierarchy, any COBOL subprogram issuing terminal I/Os must run in conversational mode. Pseudo-conversational programs would need to be modified, and any new development using the Natural 3GL CALLNAT interface should be done in conversational mode.

More Samples

Sample Program	Description
XNCI3GC2	COBOL sample with same functionality as XNCI3GC1, but accepting parameters from the calling Natural program.
XNCI3GP1	PL/I sample with same functionality as COBOL sample XNCI3GC1.
XNCI3GP2	PL/I sample with same functionality as XNCI3GC1, but accepting parameters from the calling Natural program.

More Non-CICS Samples

Sample Program	Description
XNAT3GC2	COBOL sample with same functionality as CICS sample XNCI3GC2.
XNAT3GP2	PL/I sample with same functionality as CICS sample XNCI3GP2.

Sample for Any Other Supported Environment

Perform the following steps to create a sample Natural 3GL CALLNAT:

Step 1: Assemble and Link ASMNAT

The sample Assembler routine XNAT3GA1 contains a basic example to access the CALLNAT interface. The register calling conventions are in the source of this program.

Link NATXCAL with XNAT3GA1 with entry point ASMNAT.

Step 2: Start the Natural Session

Start a Natural session stacking a program that calls the ASMNAT program which in turn calls the Natural subroutine ASMNAT.

IX 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 products.

Editor Work File
Editor Buffer Pool

See also:

- *SYSEDT Utility - Editor Buffer Pool Administration* in the *Utilities* documentation
- *Installing the Software AG Editor* in the *Installation for z/OS* documentation
- *Software AG Editor* in the *Editors* documentation

24 Editor Work File

- Editor Work File Structure 212
- Editor Work File under z/OS 213
- Using the Editor Work File Formatting Utility 214
- Formatting during Initialization 214
- Maintaining the Editor Work File 214
- Editor Work File under Complete/SMARTS 215

This document describes structure, use and maintenance of the editor work file under the various operating systems.

See also:

- *SYSEDT Utility - Editor Buffer Pool Administration* in the *Utilities* documentation
- *Installing the Software AG Editor on z/OS* in the *Installation for z/OS* documentation
- *EDBP - Software AG Editor Buffer Pool Definitions* in the *Parameter Reference* documentation
- *Software AG Editor* in the *Editors* documentation

Editor Work File Structure

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

- Control Record
- Work Records
- Recovery Records

Note: If you use an editor auxiliary buffer pool defined by the profile parameter `EDPSIZE`, no editor work file is required.

Control Record

The control record contains buffer pool control information including the buffer pool parameters.

During the first initialization of the work file or during a buffer pool cold start (triggered by editor buffer pool subparameter `COLD`), the values defined in the editor buffer pool parameter `EDBP` and/or in the corresponding macro `NTEDBP` are saved in the work file control record. Moreover, the current operating system Id (system variable `*HOSTNAME`) and the global buffer pool name or the current job name are saved for subsequent verification.

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

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

Work Records

The work records contain logical file records which have been moved out of the buffer pool due to a lack of free buffer pool blocks.

Logical work file records are lost during a restart of the buffer pool or if a timeout occurs for the logical file.

Recovery Records

The recovery records hold checkpoint information of editor sessions. If the system terminates abnormally, this information can be used by the editor recovery facility to recover logical files. Recovery records are lost during a cold restart of the buffer pool.

The recovery facility is used by Natural ISPF only. If you do not intend to use this product, you can run without the recovery part by defining the editor buffer pool subparameter `PWORK=100`.

Editor Work File under z/OS

One editor work file corresponds to one **Editor Buffer Pool**. If you intend to use a global editor buffer pool, the editor work file must be shared by all users using the same global editor buffer pool. The accessed editor work file can be used only by sessions within the same operating system (system variable `*HOSTNAME`) and with the same global buffer pool or the same job name for local buffer pools. This connection can be dropped by a buffer pool cold start only. Alternatively, the editor work file formatting utility can be used to reset the work file connection.

The editor work file must be large enough to contain the editor sessions of all users. A minimum number of 100 records per editor user is recommended. The record length of the work file must be fixed, can be defined from 504 to 16384 bytes, and must be a multiple of 8.

Note: The record length of data sets or PDS members, which will be edited with Natural ISPF, cannot be larger as the record length of this editor work file.

The size of a work file record is specified either when allocating the editor work file (default size is 4088).

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

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

- offline by using the editor work file formatting utility,
- online during buffer pool initialization.

Using the Editor Work File Formatting Utility

This method is to be preferred, because no online user has to wait until formatting is finished. Optionally, the **Natural parameter module** may be assembled and linked to the editor work file formatting utility to specify editor buffer pool parameters by means of the macro `NTEDBP`. Otherwise, the default parameter values apply.

During reformatting, however, the work file must not be in use, which means that the system(s) using the corresponding buffer pool have been terminated before reformatting.

Formatting during Initialization

When the editor buffer pool is in uninitialized or terminated state, then during the first session which uses the Software AG Editor, a "buffer pool cold start" is performed on one of the following conditions:

1. if the work file has not been formatted yet,
2. if the control record indicates “cold start” (which can also be specified by using the Editor Buffer Pool Administration utility `SYSEDT`),
3. if the buffer pool subparameter `COLD=ON` was specified.

Otherwise, a buffer pool warm start is performed if a valid control record is found during buffer pool initialization. In this case, all buffer pool parameters are taken from the work file control record and no records are formatted.

Maintaining the Editor Work File

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 formatting utility can be used to avoid a buffer pool cold start; that is, the loss of the recovery records.

To copy an existing editor work file, perform the following steps:

1. Modify any buffer pool parameters by using the *SYSEDT Utility*, for example, `PWORK` if you want to change the percentage of work records in the file.
2. Terminate the editor buffer pool by using the *System Administration Facilities* of the *SYSEDT Utility* and ensure that no Natural session is using the editor after the buffer pool termination.
3. Close (if necessary) and deallocate the editor work file.
4. Rename the editor work file by using the VSAM utility `IDCAMS` (`ALTER` command).

5. Define a new editor work file with the original name and possibly a different size, but with the same record length.
6. Perform the following steps:
   - In the `EXEC` JCL card, add `PARM=COPY`.
   - For the renamed editor work file `CMCOPY` to be copied into the new work file `CMEDIT`, add `//CMCOPY DD....`
   - Run the editor work file formatting utility with the new file.
7. Check the job log for potential errors.
8. Reallocate and (if necessary) reopen the editor work file.
9. Use the Editor Buffer Pool Administration utility `SYSEDT` to check that the buffer pool and the work file have been restarted successfully.

Important: All Natural sessions must be restarted if you want them to use the editor after the buffer pool restart.

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.

25 Editor Buffer Pool

- Purpose of the Editor Buffer Pool 218
- Obtaining Free Blocks 219
- Initializing the Editor Buffer Pool 219
- Restarting the Editor Buffer Pool 220
- Editor Buffer Pool Parameters 220
- Buffer Pool Initialization for Multi-User Environments 220

This document describes purpose, use and operation of the Editor Buffer Pool which is an intermediate main storage area used by the Software AG Editor.

Purpose of the Editor Buffer Pool

The editor buffer pool can be seen as an extension of the editor buffer (`SSIZE`). It is an intermediate main storage area used by the Software AG Editor to maintain its logical files.

A logical file consists of one or more logical records and contains the data of a Natural source object or a file (for example, a job, a PDS member or an LMS element) maintained by the editor. As a user can work with more than one object at the same time, several logical files can exist concurrently for each user.

The number of logical files (as well as the percentage of recovery records in the **Editor Work File** is defined in the buffer pool parameter macro.

The editor buffer pool can be defined as a local or a global or an auxiliary (`EDPSIZE`) buffer pool. In multi-user environments (CICS and IMS TM), the editor buffer pool is shared by all editor users of either the same region (local pool) or more than one region (global pool).

The editor buffer pool contains various control tables and a number of data blocks:

Area	Size
Main control block	500 bytes
Logical file table	20 bytes per logical file
Work file table	4 bytes per record
Recovery file table	16 bytes per record
Buffer pool block table	28 bytes per block
Buffer pool blocks	see text below

As the size of a buffer pool block is equal to the size of a work file record, one buffer pool block can contain one logical file record.

The buffer pool is initialized by the first editor user. During warm start buffer pool initialization, all recovery records are checked to build the recovery file table.

Several functions are provided to access the buffer pool (for example, functions to allocate, read, write or delete a record).

Obtaining Free Blocks

If the buffer pool becomes full, buffer pool blocks have to be moved to an external data set, the editor work file, to obtain free blocks.

In such a situation, the editor checks all logical files for their timeout value and deletes any logical file which has not been accessed within the specified time. This means that all its buffer pool blocks and work file records are freed, and the logical file is lost.

If there is still no buffer pool block available, the editor moves the oldest block to the work file, according to the specified timeout parameter values (see the *Generation Parameters* function of the *SYSEDT Utility* in the Natural *Utilities* documentation).

Initializing the Editor Buffer Pool

An uninitialized editor buffer pool is initialized when the Software AG Editor is called for the first time. Then the various control blocks are created. There are two different modes of buffer pool and work file initialization: “cold start” and “warm start”.

Buffer Pool Cold Start

A buffer pool cold start can be triggered by the editor buffer pool subparameter `COLD` or by the Editor Buffer Pool Administration utility `SYSEDT` or automatically (if the editor work file is unformatted).

During a buffer pool cold start, the values of the editor buffer pool parameter `EDBP` or the corresponding macro `NTEDBP` are stored into the work file control record and all work file recovery records are cleared.

Buffer Pool Warm Start

During a buffer pool warm start, the buffer pool parameters are read from the work file control record and all work file recovery records are read to build the recovery file table in the buffer pool.

The accessed editor work file can be used only by sessions within the same operating system (system variable `*HOSTNAME`) and with the same global buffer pool or the same job name for local buffer pools. This connection can be dropped by a buffer pool cold start only. Alternatively, the editor work file formatting utility can be used to reset the work file connection.

Restarting the Editor Buffer Pool

The Editor Buffer Pool Administration utility `SYSEDT` can be used to terminate the editor buffer pool, that is, to set it to the uninitialized state. This avoids the restart of the TP system or of the global buffer pool.

If `SYSEDT` is not available due to buffer-pool problems, the program `BPTERM` can be used to terminate the buffer pool.

Important: All Natural sessions must be restored if you want them to use the editor after buffer-pool restart.

Editor Buffer Pool Parameters

The editor buffer pool parameter `EDBP` or the corresponding macro `NTEDBP` in the **Natural parameter module** is required to define parameters for the operation of the editor buffer pool.

When the editor work file is formatted, these parameters are stored into the work file control record while all other records are cleared. Thus, reformatting a work file that has been previously used, means that all editor checkpoint and recovery information is lost.

Some of these parameters can be modified dynamically during execution of the buffer pool by using the Editor Buffer Pool Administration utility `SYSEDT`.

Buffer Pool Initialization for Multi-User Environments

During the buffer pool initialization, all recovery records are read from the editor work file. Therefore, the first users have to wait for a long time or even receive a timeout message until the editor buffer pool initialization is finished.

For this reason, a special Natural program has been supplied to trigger the buffer pool initialization before the first user becomes active. This program can be activated either during the startup of the TP monitor, or by a batch job if a global buffer pool is used.

The session must then be started with the session parameter:

```
STACK=(LOGON SYSEDT,user,password;BPINIT;FIN)
```

Under CICS: If the session runs asynchronously, `SENDER=CONSOLE` must be specified to obtain any error messages issued during initialization. The source program `FRONTPLT` is supplied as a sample program to show you how to start an asynchronous Natural session during CICS startup via `PLTPI`.

X Natural as a Server

This part describes the use of Natural as a Server under z/OS 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 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.

26 Natural as a Server under z/OS

- Functionality under z/OS 226
- Natural Nucleus Installation in a Server Environment 227
- Print and Work File Handling with External Data Sets in a Server Environment 227

This document applies under z/OS only.

Functionality under z/OS

Besides being a programming language, Natural can also act as a server in a client/server environment. It can provide services, such as the execution of Natural subprograms. Part of the server functionality is the enhanced batch driver. There are a lot of underlying protocols for the client/server communication, such as the execution of stored procedures for Db2 and the execution of remote procedure calls, see the *Natural RPC (Remote Procedure Call)* documentation.

Natural Server Stub

Natural as a server runs in a separate region or within the server subsystem region, for example, for Db2 stored procedures. To run Natural as a server, a service-specific server stub is required. This server stub is supplied as part of the server product. It controls all service requests and is the only interface to the Natural server front-end.

There are different server stubs for Db2, for Natural RPC and for others.

Natural Batch Driver

The Natural batch driver (that is, for example, `NATOS` under z/OS) has been enhanced to act as the environment-specific interface component which maintains the Natural server sessions and supplies environment-specific services to Natural. It can be linked to the server stub module or loaded by the server stub as a separate module.

The batch driver is able to create and to control multiple sessions by using storage threads including functionality for thread storage compression, decompression and rollout to external storage devices.

When the batch driver is called by the server stub for the first time (during server initialization), the storage threads are created in main storage. The number and size of the storage threads is determined by the server stub. Then a static Natural session is initialized. This includes profile parameter evaluation and the allocation of static storage buffers. The resulting pre-initialized storage thread is saved in main storage separately. For every new Natural session, this initial 'session clone' is copied into the thread.

When decided by the server stub, a session can be rolled out to be resumed at a later point of time. The **Natural Roll Server** is used by the driver to save the compressed thread storage of a session. As an alternative, main storage can be used to save the compressed thread storage. In this case, the number of sessions in rolled-out state is limited by the region size.

Natural Nucleus Installation in a Server Environment

The Natural nucleus and its batch driver are designed to support both, server and non-server environments. For the server-specific definitions and requirements, please refer to the specific documentation (for example, to the *Natural RPC (Remote Procedure Call)* documentation or to the *Natural for Db2* documentation).

If the number of sessions is not limited to a small number and if the server type supports session rollout, the **Natural Roll Server** must be installed and be started before the server initializes. To do this, ensure that the `SUBSID` parameter in the **Natural parameter module** is set to the correct value. For the server, the Adabas link interface (`ADALNK`) must be generated so that `ADALNK` is also reentrant, in addition to the server.

You can use a local or a global **Natural buffer pool**. If you define a local buffer pool, it will be shared by all sessions within the server region.

If a logical print or work file number is to be used for processing within any server session, it must be associated with an access method at session start time. This can be done in the **Natural parameter module** with the macros `NTWORK` and `NTPRINT`, as in the following example, if you want to allow the full range of all print and work file numbers possible:

```
NTPRINT (1-31),AM=STD,OPEN=ACC,DEST=*
NTWORK (1-32),AM=STD,OPEN=ACC,DEST=*
```

The subparameter `DEST=*` defines generic DD name generation during the first `DEFINE WORK FILE` or `DEFINE PRINTER` statement, `OUTPUT` clause (see below). Subparameter `OPEN=ACC` avoids pre-opening of the files at program start time. The open is issued upon the first access of the file.

Print and Work File Handling with External Data Sets in a Server Environment

When running many concurrent sessions in one region, there may be resource conflicts with external print and work files. The logical names (DD names) for print and work files are defined by the subparameter `DEST` of macro `NTPRINT`, respectively `NTWORK` or its dynamic equivalents, `PRINT` or `WORK` (defaults `CMPRT`*nn* and `CMWKF`*nn*). For normal Natural batch processing, these files are defined in JCL by a logical (DD) and a physical data set name.

However, DD names are reserved by the operating system for exclusive use by one task, respectively session, that is, if CMWKF01 is opened by one session for processing, no other session could use this file until it is closed again. Other sessions would get an error if they would try to open it.

In a server environment, all print and work file requests are handled by a dedicated I/O subtask. This ensures data set integrity and avoids resource contention. It enables the shared usage of print and work files across Natural session boundaries, that is, multiple sessions can access the same

file concurrently. This is true only for print and work files whose DD-name starts with CM. All other files are considered as exclusive and cannot be shared.

For exclusive usage of print and work files, Natural offers the following two features to support print and work files in a server environment (both require a special implementation within the Natural application programs for the server environment):

- `DEFINE WORK FILE` or `DEFINE PRINTER` statements, `OUTPUT` clause and
- dynamic data set allocation (application programming interface `USR2021N`, see *SYSEXT - Natural Application Programming Interfaces*).

The `DEFINE WORK FILE` and the `DEFINE PRINTER` statement `OUTPUT` clause can be used

- to define the logical DD name for a work or print file, or
- to define the physical data set name, or
- to define an output spool class.

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

If a physical data set name or a spool file class is specified, the access method itself allocates the data set dynamically during the execution of the `DEFINE ...` statement. To ensure that a unique DD name is used, `DEST=*` should be predefined in the **Natural parameter module**. This avoids any DD name conflicts.

If the application is using the application programming interface `USR2021N`, it may specify an asterisk value for the DD name variable to get back a unique DD name from the access method. This DD name can be used for a subsequent `DEFINE ...` statement.

By default, the access properties of the server job are used for print and work files. Some server types, for example, Natural Development Server and Natural RPC, support impersonation, that is, the access properties of the individual client account is used for exclusive print and work files. For more information, refer to the corresponding section in your server documentation.

27 Natural as a Server under CICS

- Functionality under CICS 230
- Natural CICS Interface Installation in a Server Environment 230
- Restrictions 231

This document applies under CICS only.

See also:

- *Natural under CICS* in the *TP Monitor Interfaces* documentation
- *Natural RPC (Remote Procedure Call)* documentation

Functionality under CICS

Natural as a Server

Besides being a programming language, Natural can also act as a server in a client/server environment. It can provide services, such as the execution of Natural subprograms. There are a lot of underlying protocols for the client/server communication, such as the execution of stored procedures for Db2 (see *Natural for Db2*) and the execution of remote procedure calls (see *Natural RPC (Remote Procedure Call)*).

Natural Server Stub

Natural as a server runs in a separate region or within the server subsystem region, for example for Db2 stored procedures. To run Natural as a server, a service-specific server stub is required. This server stub is supplied as part of the server product. It controls all service requests and is the only interface to the Natural server front-end.

There are different server stubs for Db2, for RPC and for others.

Natural CICS Interface Installation in a Server Environment

There is nothing specific to define when installing the Natural CICS Interface in order to serve as a Natural server environment. There are no requirements on thread type or type of rolling (CICS roll facilities or roll server).

Actually, Natural server sessions may share a Natural under CICS environment with “normal”, for example, terminal bound Natural sessions. The difference is that, in case of a Natural server session, the Natural CICS Interface does not deal with a principal facility, such as a terminal or printer, but with a server stub. In terms of CICS, a Natural server session is a series of asynchronous CICS tasks, and the session context (session restart data) is maintained by the server stub using a unique 8-byte session ID.

Restrictions

The following restrictions apply when Natural is used as a server under CICS:

1. Natural server sessions under CICS can only run in pseudo-conversational mode. A Natural server session cannot run in conversational mode, as the Natural CICS Interface always has to pass control back to the server stub; therefore `PSEUDO=ON` is forced for Natural server sessions under CICS. Because of the same reason `RELO=ON` is forced for Natural server sessions using `TYPE=GETM` threads.
2. 3GL programs called by Natural should be aware of the fact that Natural server sessions are running asynchronously in CICS, that is, no CICS terminal (TCTTE) is available.
3. The profile parameter `ADAMODE` should be set to 1 or 2, otherwise Adabas may build a different UQE ID for each dialog step of the Natural server session.
4. The profile parameter `PROGRAM` or equivalent back-end program settings by Natural are not honored, as the logic flow at session termination from the Natural CICS Interface to the server stub must not be interrupted and/or falsified by a potential back-end program.
5. Care should be taken when using the parameter `TERMVAR` (`&TID`) in the macro `NTCICSP` 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.

XI Natural Execution - Miscellaneous Topics

This part provides general information on Natural execution.

Natural 31-Bit Mode Support
Support and Use of Natural and Non-Natural Objects
Input/Output Devices
Double-Byte Character Sets
Asynchronous Processing

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

28 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.

29 Support and Use of Natural and Non-Natural Objects

- Support for Natural Objects from Previous Natural Versions 238
- Back-End Program Calling Conventions 238
- LE Subprograms 240
- External Sort Programs 243

Support for Natural Objects from Previous Natural Versions

Natural objects created in an earlier version of Natural can be executed in the current Natural version without any adjustments to the objects or any conversion or migration procedure being required. This also applies to objects that have been cataloged with the Natural Optimizer Compiler.

For details about supported Natural versions, see *Product Versions Supported by Natural* in the current Natural for z/OS *Release Notes*.

Back-End Program Calling Conventions

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

Notes:

1. Except under z/OS in batch mode, a specified back-end program is *not* invoked if the Natural session is executing on a Natural Development Server.

This section covers the following topics:

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

Back-End Program Calling Conventions (Batch Mode)

If the profile parameter `PROGRAM` is specified (or set dynamically during a Natural session by calling the subprogram `CMPGMSET` in the library `SYSEXTP`), a back-end program is invoked, regardless of whether the session terminated normally or abnormally. The back-end program is called using standard OS linkage conventions and must return the control to its caller.

If a back-end program is available, Natural does not issue any session termination messages. Non-zero user return codes, specified via *`operand1`* of the Natural `TERMINATE` statement, are indicated by the Natural error message NAT9987.

A parameter area containing the following information is passed to the back-end program:

- a fullword that holds the Natural system or user return code,
- a Natural termination message of 72 characters,
- a fullword that holds the length of the Natural termination data (or zero),

- the termination data passed by *operand2* of the TERMINATE statement (if any).

The back-end program parameter area is at least 80 bytes long. The macro NAMBCKP, which contains a DSECT layout of the back-end program parameter area, is supplied in the Natural source library and can be used by Assembler back-end programs.

Special Considerations under CICS

Under CICS, the back-end program parameter data is passed in the COMMAREA and in the TWA. In the TWA, only 80 bytes are passed, containing return code and message, while the length field contains an address that points to the full back-end program parameter area. The same TWA is also provided if Natural has been invoked via EXEC CICS LINK; see also *Natural under CICS*, *Front-End Invoked via LINK* in the Natural *TP Monitor Interfaces* documentation.

If the parameter BACKRPL=ALL is set in the NTCICSP macro (depending on the Natural CICS Interface version installed), only the termination data is passed in the COMMAREA.

Special Considerations under IMS TM

Under IMS TM, the calling conventions for a back-end program are different in a dialog-oriented environment. There, the back-end program is called by a program-to-program switch and the name of the back-end program is used as an IMS TM transaction code. In this case, the Natural environment is terminated before the program-to-program switch takes place; see *Natural under IMS TM*, *Support of Natural Profile Parameter PROGRAM* in the Natural *TP Monitor Interfaces* documentation.

Sample Back-End Programs

The following table contains a number of sample programs:

Sample Back-end Program for Batch and TSO Environments in COBOL:

```
LINKAGE  SECTION
  01     BACKEND-PARM-AREA.
  02     TERMINATION-RETURN-CODE            PIC S9(8) COMP.
  02     TERMINATION-MESSAGE                PIC X(72).
  02     TERMINATION-DATA-LENGTH            PIC S9(8) COMP.
  02     TERMINATION-DATA                   PIC X(100)
...
PROCEDURE DIVISION USING BACKEND-PARM-AREA
```

Sample Back-end Program for Batch and TSO Environments in Assembler:

```
BACKPROG CSECT
         SAVE   (14,12)
         LR     11,15
         USING  BACKPROG,11
         L      2,0(1)
         USING  BCKPARM,2
...
         RETURN (14,12)
BCKPARM  NAMBCKP
         END
```

Sample Back-end Program for CICS in Assembler:

```
L        2,DFHEICAP
USING    BCKPARM,2
...
BCKPARM  NAMBCKP
         END
```

Sample Back-end Program XNATBACK for Batch Mode:

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` and the operator console; potential termination data is printed on `SYSPRINT` in dump format.

LE Subprograms

This section applies to z/OS batch mode, CICS, Com-plete, IMS TM and TSO. It provides information on how Natural supports IBM Language Environment (LE) subprograms.

This section covers the following topics:

- Support of IBM LE Subprograms
- Enabling Natural Support of LE Subprograms
- Passing LE Runtime Options

- LE Abend Handling

Support of IBM LE Subprograms

To support IBM Language Environment (LE) subprograms, Natural must be prepared for the `CALL` statement to be able to call LE subprograms. LE subprograms can be static (profile parameters `CSTATIC` and `RCA`) or dynamic subprograms of Natural.

Dynamic LE subprograms of Natural are loaded via the `CEEFETCH` LE service and deleted by the `CEERELES` service as per the `DELETE` profile parameter.

Enabling Natural Support of LE Subprograms

The following is required to be able to call IBM Language Environment (LE) subprograms from Natural:

1. When installing Natural CICS Interface, the environment-dependent nucleus must be generated as described in the appropriate installation steps in *Installing Natural CICS Interface on z/OS* in the *Installation* documentation.

   For LE enablement of Natural under CICS, see also the appropriate installation steps and the section *Natural CICS Interface and IBM Language Environment (LE)* in the *TP Monitor Interfaces* documentation.
2. For LE enablement of Natural under Com-plete, the `LE370` keyword subparameter of the `NTCOMP` macro must be set to `ON` (see the *Parameter Reference* documentation). See also the chapter *IBM Language Environment Considerations* in your *Com-plete* documentation.
3. 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.
4. Under z/OS batch, IMS TM and TSO, Natural can also call LE main programs, but only as dynamic subprograms. If an LE main program is to be called dynamically, this has to be indicated by specifying `SET CONTROL 'P=L'` before the `CALL` statement. Otherwise, the LE environment created by Natural will be terminated by the LE main program.

Passing LE Runtime Options

Under z/OS Batch and TSO:

You have three options:

1. You can pass LE run-time options by using the `PARM=` parameter in your JCL. The following applies:
   - The run-time options that are passed to the main routine must be followed by a slash (/) to separate them from the Natural parameters.

- If you want to use a slash within your Natural parameters, then your Natural parameters must begin with a slash.

  Example:

  ```
  PARM='/ID=/,...'
  ```

2. You can pass LE run-time options by using the 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)
   /*
   ```

3. You can define LE run-time options by modifying and re-assembling the supplied source module NATLEOPT. For example, if you have any subprograms still running in 24 bit mode, set SYSPARM(RMODE24) as parameter for the assembler rather than changing NATLEOPT.

   If you have other specific requirements for your LE subprograms, you can add the desired LE options for the CEEXOPT macro in source module NATLEOPT.

Under IMS TM:

You can pass LE run-time options by providing the region-specific run-time options load module CEEROPT in your STEPLIB concatenation. In addition, the LE library routine retention initialization routine CEELRRIN must be present on the PREINIT list of your region JCL.

The following is a sample definition of a CEEROPT load module that allows the execution of AMODE(24) subprograms:

```
CEEROPT  CSECT
CEEROPT  AMODE ANY
CEEROPT  RMODE ANY
         CEEXOPT ALL31=((OFF),OVR),                                    X
               STACK=((128K,128K,BELOW,KEEP,512K,128K),OVR)
         END   CEEROPT
```

LE Abend Handling

Natural supports the LE-specific user error handling, that is, if an LE subprogram has defined a user error handler, this handler gets control when an abend, a program check or any other LE error condition occurs in the subprogram. If no LE user error handler has been defined, Natural reacts according to the setting of the `DU` profile parameter.

In this case, a special error message (NAT0950 if `DU=OFF` or NAT9967 if `DU=ON`) is issued which indicates the LE error number. In addition, the corresponding LE error message is issued on `CEEMSG` and an LE snap dump is written to `CEEDUMP` according to LE run-time option `TERMTHDACT`.

Note: In case of `DU=FORCE`, the abend handling of Natural is disabled and the LE error handling takes place even if no LE subprogram is active at the time of the abend. In this case, it is strongly recommended to specify the LE run-time option `TERMTHDACT(UAIMM)` to get all required diagnostic information.

External Sort Programs

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

The following topics are covered:

- Support of External Sort Programs
- Special Considerations

Support of External Sort Programs

The Natural `SORT` statement may optionally invoke an external sort program that carries out the actual sorting. An external sort program is used if the keyword subparameter `EXT` of the macro `NTSORT` is set to `ON` in the Natural parameter module.

Natural supports all external sort programs that comply with the sort interface documented in the manuals for the relevant operating system.

The requirements (for example, space and data sets) are identical to those for the execution of a 3GL (for example, COBOL, PL/I) application program that invokes the operating system sort program and can vary according to the external sort program in use.

The communication with the external sort program is via the E15 and E35 user-exit routines. As a consequence, Natural does not require the data sets `SORTIN` and `SORTOUT`.

Special Considerations

All external sort programs supporting the extended parameter list can be used.

30 Input/Output Devices

- Terminal Support 246
- Light Pen Support 246
- Printer Support 247

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

Terminal Support

Natural supports a wide variety of terminal types for the use with z/OS 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:

- ***NTDVCE - Terminal-Device Specification Table***
- *Terminal Communication - Profile Parameters Grouped by Function* (*Parameter Reference* documentation)
- ***NATCONFG Module*** (various I/O translation topics)
- *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 '....'
...
```

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 ....
```

`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
40 F0 C6 C9 D9 E2 E3 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 E2 C5 C3 E4 D5 C4 C4 40 40 40 40 40 40 40 (hex)
+ 1                       S E C O N D
4E F2 40 40 40 40 40 40 40 40 40 40 40 40 40 E3 C8 C9 D9 D4 40 40 (hex)
+ 2                                            T H I R D
4E F3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 C5 (hex)
+ 3                                                              F
```

Sample JCL for Laser Printer Usage

```
....
//xxxx JOB xxxxx,....
.
//xxxxx EXEC PGM= XXXXXX;......
.
// PARM='INTENS=4,XXXX,.......
.
.
//OUT1 OUTPUT PAGEDEF=XXXX,FORMDEF=XXXX,TRC=ON
.                      I             I
.                      I             I => 3800 form definition
.                      I
.                      I => 3800 page definition .
//CMPRT01 DD SYSOUT=Y
//        DCB=(RECFM=FBA,LRECL=133),OUTPUT=*,OUT1
//        CHARS=(WWWW,XXXX,YYYY, ZZZZ)
                 I
                 I => IBM font names
...
```

31 Double-Byte Character Sets

- Natural Profile Parameter SOSI 252
- Output Format Specification 252
- Parameter Definitions for DBCS Support 252
- Editor Profile Options 253
- Input Data Check 253
- Output Data Adjustment 254
- Natural Stack Data 254
- Application Programming Interfaces for DBCS Handling 254
- Alternate Text Module NATTXT2U 255

This document is only relevant for Asian countries which use double-byte character sets. It describes all features implemented in Natural to support DBCS terminals and printers.

Natural Profile Parameter SOSI

In alphanumeric fields with SBCS and DBCS characters mixed, the DBCS character strings are separated from the SBCS strings by shift codes called SO (shift-out) and SI (shift-in). The Natural profile parameter SOSI is used to pass the values of the shift-in and shift-out codes used in the current environment to Natural.

It is strongly recommended to use the IBM characters `X'0E'` and `X'0F'` internally. With this technique, all applications and data can be handled in a compatible manner, which means that a network supporting different mainframe types can still use the same Natural applications and process the same data.

For detailed information on this parameter, see `SOSI`.

Output Format Specification

The Natural session parameter `PM=D` is used to define DBCS-only fields. A DBCS-only field must contain only valid DBCS characters; shift-out/shift-in characters (SO/SI) are not allowed within such a field. To display a field with the session parameter `PM=D` specified, the screen attribute `X'43F8'` is added for IBM terminals.

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 `NATCONFG` module.
`SOSI=(0E,0E,0F,0F,1)`	Defines the DBCS shift-out and shift-in values for IBM hardware.
`LC=ON`	Does not translate all input data to uppercase, which again would destroy possible DBCS input data.

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

Editor Profile Options

If you want to enter DBCS or half-width Katakana characters in one of the Natural editors, the following editor general default options should be set in the editor profile to avoid that character constants or field names containing DBCS or half-width Katakana characters are unintentionally converted to upper case:

Option	Value	Explanation
Editing in Lower Case	Y	Lower-case characters in the source code are not automatically converted to upper case. This option is required if you are using DBCS or half-width Katakana characters.
Dynamic Conversion of Lower Case	N	Any source code remains as you enter it. This option is required if you are using half-width Katakana characters.

For detailed information on the editor general default options, see *General Defaults*. For detailed information on the editor profile, see *Editor Profile* in the *Editors* documentation. To avoid the need to change these options for every user, you can modify the default profile for your installation by means of the user exit routine USR0070P, which also supports DBCS; see ***USR0070P - User Exit for Editor Profiles*** in the section *Configuring Natural*.

Input Data Check

If the session parameter PM=D is set for a field, it is verified that the input data

- contains an even number of bytes,
- contains only valid DBCS characters,
- does not contain shift-out/shift-in characters (SO/SI).

Because the detection of non-DBCS characters requires ICU, this check will not be performed if ICU is not available (that is, if the profile parameter CFICU=OFF has been set).

Output Data Adjustment

If a window is to be displayed for user interaction, the window might overlay DBCS characters that are already displayed, or the window might itself contain DBCS characters which are truncated because of the window size. An overlay may also occur if the `NO ERASE` option is used with an `INPUT` statement. In order to prevent screen corruption in case of such an overlay, the following actions are performed to adjust the output data, if necessary:

- if the session parameter `PM=D` is set for a field, an orphan byte (that is, a single byte left at the beginning or end of the data to be displayed as a result of a partial overlay of a DBCS character) is replaced by an attribute; this operation assures that only valid DBCS characters are displayed;
- if the profile parameter `SOSI` has been set, the field contents of an alphanumeric field for which `PM=D` is not specified is examined for shift-out/shift-in characters (SO/SI); if a shift-out character (SO) is found for which the correlating shift-in character (SI) is missing, either the last character of the output data is replaced by a shift-in character (SI) or the last two characters are replaced by a shift-in character (SI) followed by a blank; if a shift-in character (SI) is found for which the correlating shift-out character (SO) is missing, either the first character of the output data is replaced by a shift-out character (SO) or the leading two characters are replaced by a blank followed by a shift-out character (SO); this operation assures that DBCS characters are enclosed properly by shift-out/shift-in characters (SO/SI).

Natural Stack Data

To avoid unintentional interpretation of DBCS characters as delimiter or control characters, the `FORMATTED` option of the `STACK` statement should be used if the data to be placed on the Natural stack contains DBCS characters.

See the *Statements* documentation for further information on the `STACK` statement.

See the *Programming Guide* for further information on the Natural Stack.

Application Programming Interfaces for DBCS Handling

The following user application programming interfaces (API) are available to support DBCS handling:

- USR4211N - Get DBCS Characters

- USR4213N - String Handling for DBCS Support

These APIs are contained as subprograms in the Natural library `SYSEXT`. Detailed information on how to use an API is included in the corresponding text object (`USR`*xxxx*`T`). See also *SYSEXT Utility - Natural Application Programming Interfaces* in the *Utilities* documentation.

USR4211N - Get DBCS Characters

The application programming interface `USR4211N` can be used to obtain information on the availability of DBCS support and the defined SOSI characters.

USR4213N - String Handling for DBCS Support

The application programming interface `USR4213N` can be used to perform the following functions:

- Convert a normal Latin character string into the corresponding DBCS character string.
- Convert a DBCS character string that contains Latin data only into a single-byte character string.
- Add the current shift codes at the beginning and at the end of a character string.
- Remove leading and trailing shift codes from a character string.

The last two functions can be used to either produce native DBCS strings or generate mixed-mode data out of native DBCS strings.

Alternate Text Module NATTXT2U

The alternate text module `NATTXT2U` contains certain keywords for English language in all upper case which are contained in mixed case in text module `NATTXT2`. `NATTXT2U` should be linked to the Natural nucleus instead of `NATTXT2` in environments where lower case code points `H'81'` to `H'A9'` are used to display national characters.

32 Asynchronous Processing

- Identifying Asynchronous Natural Sessions 258
- Handling Output of an Asynchronous Natural Session 258
- Handling Unexpected or Unwanted Input 259
- Other Profile Parameter Considerations 259

This document describes asynchronous Natural processing, a method which is available under all TP monitors supported by Natural.

An asynchronous Natural session is a session which is not associated with any terminal and therefore cannot interact with a terminal user. It can be used to execute a time-consuming task "in the background" without the user having to wait for the task to finish.

Related Topics:

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

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.
`ENDMSG`	The error message NAT9995 (normal termination message) can be suppressed by specifying `ENDMSG=OFF`.
`IMSG`	Natural initialization error messages and warnings can be suppressed by specifying `IMSG=OFF`.
`MENU`	Asynchronous sessions only have the Natural stack for command inputs; therefore, it is recommended to specify `MENU=OFF` and to navigate through Natural by using direct commands.
`PLOG`	Dynamic parameter logging is executed by sending all parameters line by line to the `SENDER` destination.
`PROGRAM`	If a standard back-end program/transaction is defined in your installation, it should be checked if this program can run asynchronously or if it is desired to deal with terminal-bound sessions only. Specifying `PROGRAM=0` bypasses the back-end logic.