# Natural

## Operations

Version 9.3.3

October 2025

ADABAS & NATURAL

This document applies to Natural Version 9.3.3 and all subsequent releases.

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



**Document ID: NATUX-NNATOPERATIONS-933-20251030**

# Table of Contents

# Preface

This documentation contains information for operating Natural in a Linux and Cloud environment. It is organized under the following headings:

| | |
|---|---|
| **Profile Parameter Usage** | Information on the parameter hierarchy. How to assign profile parameter values statically, dynamically and at runtime. |
| **System Files** | How system files and Natural objects are stored in the file system. Information on the system files `FNAT`, `FUSER` and `FDDM`. |
| **Work Files** | How to define work files. Information on the different work file formats. |
| **Natural Buffer Pool** | How the buffer pool is used by Natural and how it is started. |
| **Using the Buffer Pool Monitor (`NATBPMON`)** | How to invoke the `NATBPMON` utility. Information on the commands that are available with this utility. |
| **Natural in Batch Mode** | How to run Natural in batch mode. Information on the required input and output channels. How to use batch mode simulation. |
| **Support of Different Character Sets with *NATCONV.INI*** | How to define different character sets in the file *NATCONV.INI*. |
| **Natural Exit Codes** | Information on the Natural exit codes, including startup errors. |
| **Setting Up the Entire System Server Interface** | How to activate the Entire System Server Interface for the product Entire System Server. |
| **Tuning SQL Database Access** | How to configure the handling of the SQL driver's statement table. |
| **User Exit for Computation of Sort Keys - `NATUSK`*nn*** | How to sort characters of other languages in the correct alphabetical order. |
| **Abnormal End (Abend) Handling** | Information on the signal handlers. |

The Natural utilities which can be used to execute numerous administrative functions are described separately; see the *Tools and Utilities* documentation for detailed information.

Security is also described separately; see the *Natural Security* documentation for detailed information.

# 1 About this Documentation

## 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:<br><br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br><br>Text you must type in.<br>Messages displayed by the system.<br>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://techcommunity.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.

# 2 Profile Parameter Usage

Natural profile parameters affect the appearance and the response of your working environment.

The parameters are described in detail in the *Parameter Reference*.

## Parameter Hierarchy

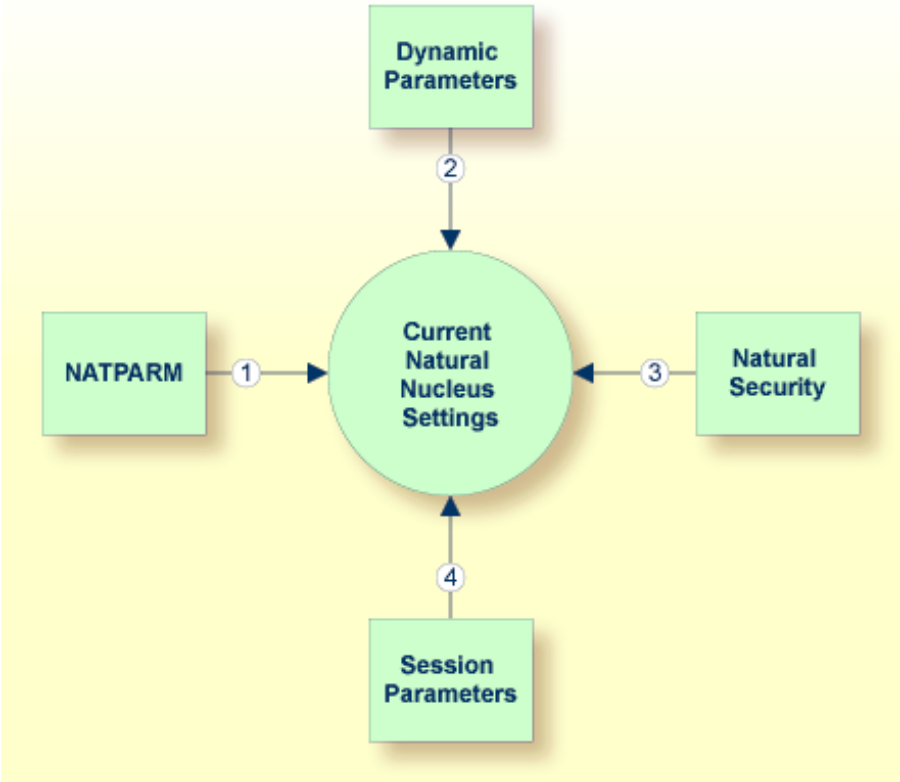
The values for the Natural parameters are taken from different sources. The priority of the parameters is as follows:

1. **Static Assignments**
   Lowest priority. Static assignments are made by parameters specified in the Natural parameter file `NATPARM`.
2. **Dynamic Assignments**
   Dynamic assignments are made by specifying an alternative parameter file and/or individual parameters when starting Natural.
3. **Runtime Assignments**
   Highest priority. Runtime assignments are made during the session by specifying session parameters.

See the remainder of this section for further information on the different types of assignments.

**Note:** When Natural Security is active, the use of specific parameters may be restricted.

The following graphic illustrates the parameter hierarchy:




## Static Assignment of Parameter Values

By default, the parameter specifications in the parameter file `NATPARM` are used to determine the characteristics of your Natural environment. Initially, this file contains default values. It can be changed using the Configuration Utility.

**Tip:** It is recommended that you do not modify the default parameter file `NATPARM`. If you want to use Natural with parameter values other than the default values, create your own parameter file (see also the following section).

## Dynamic Assignment of Parameter Values

Using the dynamic parameters, you can set up your own environment when starting Natural. When the session is started, the operating system passes the values for the dynamic parameters to Natural.

The dynamic parameters are valid for the current Natural session. They override the static assignments specified in the default parameter file `NATPARM`.

Using the Configuration Utility can also create your own parameter files. To use one of your own parameter files, you have to specify its name when starting Natural.

**To start Natural with dynamic parameter values**

- Add the dynamic parameters and their values to the command that is used to start Natural.

  Example: The profile parameter `PARM` is used to invoke Natural with the alternative parameter file `MYPARM`. The values for the profile parameters `SM` and `DTFORM` are to be used instead of those defined in `MYPARM`:

  ```
  natural PARM=MYPARM SM=ON DTFORM=I
  ```

### Special Characters

Special characters like brackets and asterisks are interpreted by the operating system. Therefore, it is necessary to put the parameters which use these special characters in double quotation marks. Example:

```
natural "FNAT=(99,30) FUSER=(99,32)"
```

As an exception to this rule, the parameters `FNAT`, `FDIC`, `FSEC`, `FDDM` and `FUSER` can also be specified without brackets to avoid using quotation marks. Example:

```
natural FNAT=99,30 FUSER=99,32
```

For each opening bracket that you specify, you also have to specify the corresponding closing bracket. Escape sequences are not supported with dynamic parameters.

## Runtime Assignment of Parameter Values

The runtime assignments are made during the session by setting session parameters. The values of the session parameters override static and dynamic assignments.

Session parameters are set with the system command `GLOBALS`. Example:

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

Session parameters can also be set with the `SET GLOBALS` statement in a program. Example:

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

**Note:** In addition to setting the session parameters at session level (as described above), you can also set them at program, statement or field level. For further information, see *Introduction to Session Parameters* in the *Parameter Reference*.

# 3 System Files

Natural for Linux stores objects in files accessible by operating system functions. Unlike Natural for z/OS where the objects are stored in Adabas system files, Natural for Linux stores the objects in specific directories on the disk. Thus, a database such as Adabas is not required to run Natural for Linux.

## System File Structure

By default, the Natural libraries are created as subdirectories below the Natural root directory of a specific Natural version. The subdirectories have the same names as the libraries.

The Natural objects are stored as files in the subdirectories. The file name for a Natural object has the following form:

```
file-name.NKT
```

| | |
|---|---|
| *file-name* | This the name of the object. See also *Object Naming Conventions* in *Using Natural*. |
| N | The first character of the extension is always "N". It stands for “Natural”. |
| *K* | The second character of the extension can be one of the following:<br>S – for source files<br>G – for generated programs<br>R – for resources |
| *T* | The third character of the extension stands of the type of the object. For valid values, see the list below. |

For example, the source program `TESTPROG` is stored as file *TESTPROG.NSP*, while the generated code for the map `TESTMAP` is stored as file *TESTMAP.NGM*.

**Note:** The file name is not always identical to the object name. Both the current object name and the corresponding internal object name are documented in the file ***FILEDIR.SAG***.

The following object types and the respective letters and numbers are used for the extensions available:

| Letter or Number | Object Type |
|---|---|
| A | Parameter data area (PDA) |
| C | Copycode |
| D | DDM |
| G | Global data area (GDA) |
| H | Helproutine |
| L | Local data area (LDA) |

| Letter or Number | Object Type |
|---|---|
| M | Map |
| N | Subprogram |
| P | Program |
| S | Subroutine |
| T | Text |
| 4 | Class |
| 5 | Command processor |
| 7 | Function |
| 8 | Adapter |

## Access Rights

By default, objects allocated by Natural get the access rights "rw-rw-rw-". This makes sure that users not belonging to the group of the owner (that is, other users) can recatalog Natural programs. If this is not desirable, you have to run the Linux utility `umask` and set the appropriate mask.

Users developing Natural applications must have read and write access to all objects belonging to the application. In a plain production environment, the write access rights may be restricted to the maintenance team. If an object cannot be accessed due to too low access rights, Natural behaves as if an object was not found.

## System Files FNAT and FUSER

The Natural system files `FNAT` (for system programs) and `FUSER` (for user-written programs) are located in different subdirectories.

`FNAT` assumes the following directory structure:

```
FNAT
 └LIBDIR.SAG
 └SYSTEM
      └FILEDIR.SAG
      └SRC
      └GP
      └ERR
      └RES
 └SYS*
      └FILEDIR.SAG
```

```
      SRC
      GP
      ERR
      RES
```

The file *LIBDIR.SAG*, which is only available for FNAT, contains information on all further installed products using Natural. This information can be displayed by using the system command SYSPROD.

FUSER assumes the following directory structure:

```
FUSER
 SYSTEM
      FILEDIR.SAG
      SRC
      GP
      ERR
      RES
 user-library1
      FILEDIR.SAG
      SRC
      GP
      ERR
      RES
```

The name of a user library must not start with "SYS".

The directory structure is generated during the installation of Natural. The directories representing the system and user libraries contain the following:

- ***FILEDIR.SAG***
  This file contains internal library information used by Natural. For further information, see *The File FILEDIR.SAG* below.
- ***SRC***
  This subdirectory contains the Natural source objects stored in the library.
- ***GP***
  This subdirectory contains the generated Natural programs stored in the library.
- ***ERR***
  This subdirectory contains the error messages stored in the library.
- ***RES***
  This subdirectory contains the private and shared resources stored in the library.

DDMs can be stored in local libraries. If DDMs are used by a program, Natural first searches the current library, then the steplibs, and then the library SYSTEM. If the DDMs are not found, the

program does not compile and displays an error message. However, if **FDDM mode** has been activated, Natural searches for the DDMs only in the system file FDDM.

The paths to the system files FNAT, FUSER and FDDM are defined in the Configuration Utility. System files are version-dependent. Therefore, Natural can only access system files of the current Natural version. It is recommended that you only have one FNAT system file. It is possible, however, to define several FUSER system files (for example, when you have different development areas for different purposes).

## System File FDDM

The system file FDDM is a container in which all DDMs can be stored.

FDDM assumes the following directory structure:

```
FDDM
└SYSTEM
      └FILEDIR.SAG
      └SRC
      └GP
```

By default, the system file FDDM is not active. If you want to use it, you have to activate FDDM mode as described below.

- Activating FDDM Mode
- Migrating DDMs to the System File FDDM
- Checking whether the System File FDDM is Used

### Activating FDDM Mode

If FDDM mode is activated (both database ID and file number do not equal 0 in the global configuration file), all DDMs are stored and read in the system file FDDM. DDMs stored in libraries will no longer be accessible from Natural. This is similar to z/OS, where all DDMs are stored in the system file FDIC.

If the FDDM system file is undefined in the global configuration file, the DDMs are stored in the Natural libraries FUSER and FNAT.

**To activate FDDM mode**

1 Create an empty directory in which the DDMs are to be stored in FDDM mode. The directory can have any name which corresponds to the Natural naming conventions.

2 Invoke the Configuration Utility.

3 In the global configuration file (category **System Files**), assign a database ID and file number for the system file FDDM and define the path to the directory that you have created in the first step.

4 Open the required parameter file.

5 Locate the parameter FDDM.

**Tip:** Locate this parameter by searching for "FDDM". See *Finding a Parameter* in the *Configuration Utility* documentation for further information.

6 For the parameter FDDM, specify the same database ID and file number that you have defined in the global configuration file.

7 Save your changes.

8 Migrate all required DDMs to the system file FDDM as described below.

### Migrating DDMs to the System File FDDM

All DDMs that are to be available in FDDM mode must be contained in the system file FDDM. Especially the example DDMs delivered with Natural in library SYSEXDDM must be available in the system file FDDM.

For migration of DDMs to the FDDM system file, you can choose between different alternatives:

- You can use the Object Handler which supports the FDDM system file and offers the possibility to migrate the DDMs into the FDDM system file. The DDMs can be unloaded from the Natural libraries and can be stored into the FDDM system file in the active Natural session.

  **Important:** To migrate a complete development environment, it is recommended to use the Object Handler.

- It is also possible to migrate the DDMs with the copy or move function of the SYSMAIN utility. In this case, it is required that the FDDM parameter is first deactivated so that your old environment is used again.

These alternatives are described below in detail.

**Note:** The INPL utility loads DDMs either to Natural libraries if FDDM mode is not active or to the system file FDDM if FDDM mode is active. This may have some impact if the loaded INPL files are intended to work in both modes. It may be necessary that the DDMs are available in the Natural libraries as well as in the FDDM system file.

#### To migrate DDMs to the system file FDDM using the Object Handler

1 Activate FDDM mode as described above.

2 Start Natural using the modified parameter file (that is, the parameter file in which path for the parameter FDDM has been defined).

3 Issue the direct command SYSOBJH to invoke the Object Handler.

The following steps assume that you use the Object Handler wizards.

4 In the main menu, mark the **Unload** function and press ENTER.

5 In the resulting screen, mark the option **Unload objects into Natural work file(s)** and press ENTER.

6 In the resulting screen, mark the option **Set additional options** and press ENTER.

7 In the resulting screen, deactivate the option **Use FDDM file for processing DDMs** and press ENTER to return to the previous screen.

This activates your old environment (which contains the DDM to migrated). If you do not deactivate this option, you cannot access the DDMs that are to be migrated.

8 Press ENTER repeatedly until the screen is shown in which the object type for the unload has to be selected.

The option **Natural library objects only** is selected by default. This option is required for the next steps.

9 Press ENTER.

10 In the resulting screen, enter an asterisk (*) in the fields **Library** and **Object name**. In addition, mark the field **More detailed specification of objects**. Press ENTER.

11 In the resulting screen, deactivate the options **Error messages** and **Shared resources**. In the **Natural types** field, enter "V" and press ENTER.

12 Press ENTER to display the command that is to be processed.

13 Press ENTER to start the unload function.

14 When the objects have been unloaded, return to the main menu.

15 In the main menu, mark the **Load** function and press ENTER.

16 In the resulting screen, mark the option **Load objects from Natural work file(s)** and press ENTER.

17 In the resulting screen, mark the option **Set additional options** and press ENTER.

18 In the resulting screen, activate the option **Use FDDM file for processing DDMs**.

This activates your new environment containing the FDDM system file.

**Note:** In different libraries, DDMs can exist with identical names. To prevent overwriting DDMs in the FDDM system file and to detect DDMs with identical names, it is recommended to load the DDMs with the **Do not replace** option. This option is located on the same page as the option **Use FDDM file for processing DDMs**.

19 Press ENTER to return to the previous screen.

20 Press ENTER repeatedly until the screen is shown in which the object type for the load has to be selected.

The option **Load all option from the work file** is selected by default. This option is required for the next steps.

21 Press ENTER.

The command that is to be processed is now shown.

22 Press ENTER to load the objects.

### Checking whether the System File FDDM is Used

When you have migrated all DDMs to the system file FDDM, you can check whether FDDM is used.

**To check whether FDDM is used**

1 Start Natural.

2 Issue the system command SYSPROF.

3 If the FDDM file is displayed, Natural will access only DDMs stored in this system file.

If the FDDM file is not displayed or if the expected files are not displayed, revise the parameter file used for your session.

## Important Information and Warnings

A Natural developer must have read, write and delete rights for all objects.

An end-user must only have read rights for the generated programs (and in some special cases also read rights for the sources).

Do not access Natural files with operating system utilities. These utilities might modify and destroy the Natural directory information.

The use of an external editor is not recommended as code page conflicts may arise. These conflicts can - but not necessarily must - deteriorate your source code.

Do not store private data files in the directories FNAT, FUSER and FDDM, since Natural may delete or modify them in an unexpected way.

Do not use one of the directories FNAT, FUSER and FDDM as working directories for your Linux applications, since this can cause problems when issuing Natural system commands.

The file name (i.e path including file name in 8.3 format) of any object accessed by Natural must not exceed 255 bytes.

## The File FILEDIR.SAG

The file *FILEDIR.SAG* supports up to 60000 objects. It contains internal library information used by Natural including the programming mode of an object (structured or reporting) and internally converted object names. These internal object names are automatically created when storing Natural objects to disk with:

- names longer than 8 characters (which can be the case with DDMs);
- names containing any special character supported by Natural but not by the operating system.

Internal object names are unique and consist of an abbreviation of the current object name and an arbitrary number. Both the current object name and the corresponding internal object name are documented in *FILEDIR.SAG*.

Even if an object is located in the correct directory, it can only be used by Natural after this library information is included in *FILEDIR.SAG*. For objects created within Natural, the library information is included automatically. For all other objects, the **Import** function of the `SYSMAIN` utility should be used.

The utility `FTOUCH` can be used to update *FILEDIR.SAG* without entering Natural.

## Portable Natural System Files

The directory file *FILEDIR.SAG* in a Natural library as well as the Natural error message files are created in a portable platform-independent format. This offers, for example, the possibility of exchanging `FUSER` libraries between different Windows and Linux platforms simply by copying the libraries via operating system commands.

The `FNAT` system file belongs to a Natural installation and is both version-specific and platform-specific. Therefore, it is not recommended to share `FNAT` system files among different platforms. Especially the `FNAT` system file on a Windows platform contains a completely different set of utilities as the `FNAT` system file on the Linux platform.

Although it is now possible to share an `FUSER` system file among different platforms, this possibility should by handled with care because Natural's locking mechanism does not cross machine boundaries and hence it would be possible for two Natural sessions on different platforms to modify the same object at the same time with unpredictable results.

The following topics are covered below:

## Language-dependent Objects

When the application to be ported uses the system variable *LANGUAGE, you have to take notice of the following information.

Almost all Natural objects are stored in the system file with a name which contains only upper-case characters. An exception are the language-dependent objects (that is: the objects which have been created for a specific language). Language-dependent objects may contain lower-case characters in their names. Since Windows is a case-preserving operating system (whereas Linux is a case-sensitive operating system), it may happen that names which have been created under Linux cause a conflict in Windows, or that an application which has been developed under Linux yields unexpected results in Windows.

### Example

The command SAVE PGM& creates an object where the object name contains the language identifier. The resulting object name depends on the setting of *LANGUAGE:

| Setting of *LANGUAGE | An object with the following name is created |
|---|---|
| 33 | PGMX (with an upper-case X) |
| 59 | PGMx (with a lower-case x) |

The separate objects which have been created under Linux (*PGMX.NGP* and *PGMx.NGP*) get entries in the file *FILEDIR.SAG* with the names PGMX and PGMx. These two objects will be treated differently, depending on the environment in which Natural is being executed:

- When you execute PGMX with Natural for Linux, the file *PGMX.NGP* is loaded into the buffer pool and executed.
- When you execute PGMX with Natural for Windows, either the file *PGMX.NGP* or *PGMx.NGP* is loaded into the buffer pool and executed. This is because Windows does not distinguish between these two objects and treats them as one and the same object. Thus it may be possible that applications which share an FUSER, or a copy of such an FUSER, behave in a different manner.

### Migrating Non-Portable Message Files to 64-Bit Platforms

Message files in the old, non-portable format which have not been created on a 64-bit platform are not readable.

If you want to migrate your applications from a 32-bit platform to a 64-bit platform, you must first convert your old message files to the portable format. You do this by using the export and import functions of the SYSERR utility. First, you export the message file to a text file, and then you generate a new message file by importing the text file into Natural. This creates a portable message file which is readable on Windows and Linux. For detailed information on the export and import functions, see *Generating Message and Text Files* in the *Tools and Utilities* documentation.

## Synchronizing Access to the System Files Using Semaphores

Semaphores are used to synchronize access to the Natural system files. Since this requires additional operating-system resources, you should consider incrementing the kernel parameters `SEMMNI` and `SEMMNS` by the number of system files to be accessed.

With the usage of semaphores, several users have permission to address the system files `FNAT` and `FUSER`. The semaphore ID is saved together with a lock file (*.*LCK*). If a further Natural session is started, the buffer pool looks for the semaphore ID and the corresponding lock file for synchronization.

If the lock file is not present, a new semaphore ID and lock file will be generated. This means that no synchronization will be possible.

**Caution:** It is not allowed to delete only one of the resources. You must always delete the semaphore ID and the corresponding lock file.

## Using NFS to Store Natural Libraries

When you use NFS (Network File System) to store Natural libraries, you can run into problems when the directories in which the Natural libraries are stored are mounted via NFS from a file server in your network.

The reason for this is the need to lock the *FILEDIR.SAG* file stored in each library during update operations of Natural objects.

If your NFS locking is incompatible or not properly set up between the involved platforms, Natural can hang in an uninterruptible state while waiting for NFS locking requests to be processed. These requests are generally logged on the consoles of the involved systems or in some other system-dependent log file.

The work-around to solve this problem is to store Natural libraries only on local disks if problems with a hanging and uninterruptible nucleus occur.

# 4 Work Files

Work files are files to which data can be written and from which data can be read by Natural programs. They are used for intermediate storage of data and for data exchange between programs. Data can be transferred from or to a work file by using the Natural statements `READ WORK FILE` and `WRITE WORK FILE`, or `UPLOAD PC FILE` and `DOWNLOAD PC FILE`.

## Defining Work Files

Using the Configuration Utility or the `DEFINE WORK FILE` statement, you can assign names (including the path) for up to 32 work files.

The maximum number of work files that can be used depends on the setting of the parameter `WORK`.

If you run a program which uses a work file for which a name and path has not been assigned, Natural automatically creates the file name and writes the work file into the temporary directory specified in the local configuration file. The name of such a file consists of the specified work file number and an arbitrary number assigned by the operating system. The generation of the work file name is based on an algorithm which tries to generate a unique name. Depending on the Natural parameter `TMPSORTUNIQ`, the naming convention may vary. If work file names are referenced from outside Natural, it is recommended that you specify the names explicitly to avoid problems identifying the files.

The following topics are covered below:



### Defining Work File Names with the Configuration Utility

In the Configuration Utility, the work file names are assigned in the category **Work Files** of a parameter file. The above mentioned parameters `WORK` and `TMPSORTUNIQ` can also be found in this category. See *Work File Assignments* in the *Configuration Utility* documentation for further information.

**Tip:** Locate the work file assignments by searching for "Work Files". See *Finding a Parameter* in the *Configuration Utility* documentation for further information.

## Defining Work File Names with Environment Variables

The following topics are covered below:



### About Defining Work File Names with Environment Variables

Work files can also be defined by using Linux environment variables. Once you have defined the work file names in the parameter file, the work file names can be set without further change to the parameter file. For example, when you specify the following name for a work file in the parameter file (or in a `DEFINE WORK FILE` statement):

```
$NATURAL/$myfile
```

and assume the following settings in your operating system:

```
set NATURAL=/usr/natural
set myfile=sub/test
```

this will expand into the following file name:

```
usr/natural/sub/test
```

**Note:** Since the different shells interpret the tilde character (~) in different ways, this character is not interpreted by Natural.

### Delimiters of Environment Variables

Names of environment variables are delimited by special characters. A left-hand delimiter is to the left of a variable, a right-hand delimiter is to the right.

For example, the string `$TEMP` identifies an environment variable named `TEMP`; `$` is used as both the left-hand and right-hand delimiter.

Valid delimiters are:

| Type of Delimiter | Valid Delimiters |
|---|---|
| Left-hand delimiter | $ |
| Right-hand delimiter | /<br>. |

### Defining Work File Names with an Application Programming Interface

You can also define work files with the application programming interface `USR1050N` in library `SYSEXT`.

## Work File Formats

The format of a work file depends on the work file type that has been defined. Different work file formats are available. Natural recognizes the format by checking the file name and its extension:

```
file-name.extension
```

where *file-name* can have a maximum of 8 characters and *extension* can have a maximum of 3 characters.

The work file formats are:

- Binary Format
- ASCII Format
- Entire Connection Format
- Portable Format
- Unformatted Format
- CSV Format

See also *Work Files and Print Files* in the *Unicode and Code Page Support* documentation.

### Binary Format

Possible type: SAG.

This format is the preferred format since it can be used with all data types. However, it is not portable across platforms with different endian modes.

Each record that is written is preceded by two bytes which contain the length of the record. The length itself is written in a platform-specific form.

To define binary format for a work file, use a file name with a period and the extension *SAG* (for example, *<file-name>.SAG*).

### ASCII Format

Possible types: ASCII and ASCII compressed.

Since each written record is terminated with a line feed (LF), ASCII format is only recommended for alphanumeric data.

To define ASCII format for a work file, enter either a file name with a period and any extension except *SAG* and *NCD* (for example, *<file-name>.<ext>*), or a file name with a period and without an extension (for example, *<file-name>*).

### Entire Connection Format

Possible types: Entire Connection and Transfer.

Work files can be accessed in two different ways:

- Locally on Linux. The work file type Entire Connection is used for this purpose.
- Via a data transfer with Entire Connection. The work file type Transfer is used for this purpose. The data are sent to Entire Connection which writes the data to the PC.

The product Entire Connection uses two files: a data file which contains the actual data and a format file which contains formatting information about the data in the data file.

Natural automatically generates the corresponding format file for the type Entire Connection. The format file has the same name as the data file, however the extension is *NCF*. For detailed information on the content of a format file with the extension *NCF*, see the Entire Connection documentation.

When using the type Transfer, the format file is generated by the product Entire Connection (provided that the option **Create format file** has not been deactivated in the user properties; see the Entire Connection documentation for further information).

To define Entire Connection format for a work file, enter a file name with a period and the extension *NCD* (for example, *<file-name>.NCD*).

You can read/write work files in Entire Connection format directly from/to your local disk.

See also ***Special Considerations for Work Files with Extension NCD***.

**Notes:**

1. The `RECORD` option of the `READ WORK FILE` statement is not available for reading work files of format Entire Connection.
2. The operand format U (Unicode) is not supported for the work file types Entire Connection and Transfer. If U is used with these work file types, a runtime error message is displayed.

### Portable Format

Possible type: Portable.

The type Portable performs an automatic endian conversion of a work file when the work file is transferred to a different machine. For example, a work file written on a PC (little endian) can be read correctly on an RS6000 or HP machine (big endian). The endian conversion applies only to field formats I2, I4, F4, F8 and U. The floating point format is assumed to be IEEE. There are, however, slight differences in IEEE floating point representation by different hardware systems. As a rule, these differences apply only to infinity and NaN representations, which are normally not written into work files. Check the hardware descriptions if you are uncertain.

The files are always written in the machine-specific representation, so that a conversion is performed only if the file is read by a machine with different representation. This keeps performance as fast as possible.

There are no other conversions for this format apart from the conversions mentioned above.

When a `READ WORK FILE` statement is used for a dynamic variable, the variable is resized to the length of the current record.

### Unformatted Format

Possible type: Unformatted.

The type Unformatted reads or writes a complete file with just one dynamic variable and just one record (for example, to store a video which was read from a database). No formatting information is inserted; everything is written and read just as it is.

### CSV Format

Possible type: CSV (comma-separated values).

**Note:** If you want to use the work file type CSV, you have to recatalog your sources using the `CATALOG` or `STOW` command. It is not possible to use the work file type CSV with generated programs of Natural Version 4.

The Natural fields are stored in a CSV work file as described below.

1. In the first step, the internal field data is converted into a readable format:
   - The field data of the internal Natural data formats B (binary), O (object handle), G (GUI handle) and C (attribute control) is copied to the record without field conversion. The data is taken as it is.
   - The field data of the internal Natural data format A (alphanumeric) is converted into the specified work file code page (see *Work Files* in the *Configuration Utility* documentation). If

no work file code page is specified in the Configuration Utility, the default code page which is defined with the parameter CP is used and no conversion is done.

The field data of the internal Natural data format U (Unicode), is converted into the specified work file code page (see *Work Files* in the *Configuration Utility* documentation) or, if no work file code page is specified, into the default code page which is defined with the parameter CP.

- The values of the internal Natural formats D (date) and T (time) are converted into an alphanumeric output format. The DTFORM parameter is evaluated so that the user-specified date and time format is used.
- The internal field values of the numeric types are converted into an alphanumeric output format.

2. In the second step, the field data in readable format is copied to the CSV work file record. The fields in the work file are separated by the specified separator character. If a field contains special characters, the field is delimited by double quotes. Each written record is terminated with a carriage return and line feed (CR/LF).

If you have defined that a header with the Natural field names is to be written to the work file (see *Work File Assignments* in the *Configuration Utility* documentation), the following applies:

- With the WRITE WORK FILE statement, a header line containing the field names of the first written record is stored in the first line of the work file. If subsequent CSV records contain a different number of fields, it may be possible that the header line does not correspond to these subsequent CSV records.
- With the READ WORK FILE statement, it is assumed that the first line of the CSV work file is the header line. Therefore, the first line is skipped (that is: the record data in the first line is not returned).

## Special Considerations for Work Files with the Extension NCD

If files with the extension *NCD* are created by Entire Connection and are then read into Natural via the READ WORK FILE statement, it is required that the Entire Connection option **Keep trailing blanks** is activated in the session properties. See your Entire Connection documentation for further information.

**Note:** When you create an NCD file using Entire Connection and load this file using the Object Handler, you may receive an error indicating that the source control record is missing. To avoid this, make sure that the option **Keep trailing blanks** is active when you create the NCD file.

The following considerations apply for work files in Entire Connection format:

- If an NCD file is read with a `READ WORK FILE` statement and the corresponding NCF format file is not available or contains invalid information, the NCD file is assumed to be an ASCII work file.
- When the `APPEND` attribute is used to append data to an NCD file, the record layouts (that is: the field format and length information which is written to the NCF format file) of the old and new data must match. If the record layouts are different, an error occurs during runtime.
- The maximum work-file record size for `WRITE WORK FILE VARIABLE` that can be handled by Entire Connection is 32767 bytes.
- If you have "old" work files with the extension *NCD*, the extensions must be changed.
- Each of the following profile parameters must be set to the same value for both read and write operations:

  `DC` (decimal character)
  `IA` (input assign character)
  `ID` (input delimiter character)
- Remember that the range of possible values for floating point variables on z/OS is different from that on other platforms. The possible value range for F4 and F8 variables on z/OS is:

  $\pm 5.4 * 10^{-79}$ to $\pm 7.2 * 10^{75}$

  The possible value range on most other platforms for F4 variables is:

  $\pm 1.17 * 10^{-38}$ to $\pm 3.40 * 10^{38}$

  The possible value range on most other platforms for F8 variables is:

  $\pm 2.22 * 10^{-308}$ to $\pm 1.79 * 10^{308}$
- A Natural error message is returned if DBMS calls are issued during an Entire Connection data transfer and their number exceeds the limit for DBMS calls permitted between screen I/Os (specified with the profile parameter `MADIO`). To circumvent this error, the application programming interface `USR1068N` in library `SYSEXT` is provided. `USR1068` resets the database call counter to zero (0). It must be invoked each time a DBMS call is issued during data transfer.

## Using the Work File Type Transfer

With local access (that is, without any data transfer being involved), you can read/write work files in Entire Connection format directly from/to your local disk. However, work files in Entire Connection format can also be accessed by using a data transfer. Both methods can be used simultaneously, but with different work file numbers only.

Work files to be accessed by using a data transfer (type Transfer) must be in Entire Connection format (NCD).

With data transfer, the Natural statements `READ WORK FILE` and `WRITE WORK FILE` do not read from and/or write to your local disk, but transfer the data to a PC that runs Entire Connection. The read/write operations are then done by Entire Connection from/to the disk of the PC.

For the work file number to be used, you have to set the profile parameter `ECPMOD` to `ON` in the Configuration Utility. It is not required that you assign a work file name in this case, because Entire Connection prompts you to enter a file name.

# 5 Natural Buffer Pool

# About the Natural Buffer Pool

The Natural buffer pool is used to share Natural objects between several Natural processes that access objects on the same computer. It is a storage area into which compiled 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.

Since Natural generates reentrant Natural object code, it is possible that a single copy of a Natural program can be executed by more than one user at the same time. For this purpose, each object is loaded only once from the system file into the Natural buffer pool, instead of being loaded by every caller of the object.

The following topics are covered below:



## Objects in the Buffer Pool

Objects in the buffer pool can be any executable objects such as programs and maps. The following executable objects are only placed in the buffer pool for compilation purposes: local data areas, parameter data areas and copycodes.

When a Natural object is loaded into the buffer pool, a control block called a directory entry is allocated for that object. This control block contains information such as the name of the object, to which library or application the object belongs, from which database ID and Natural system file number the object was retrieved, and certain statistical information (for example, the number of users who are concurrently executing a program).

### User Access for the Buffer Pool under Linux

Resource sharing requires that access to the buffer pool be coordinated among all users. Several system resources are necessary to accomplish this. For example, shared memory on the Linux operating system is used to store the objects and their administrative information. To synchronize access to these objects, a set of semaphores is used. The amount of available shared memory and the number of semaphores is configured statically in the operating system, and as a result, it may be necessary to change system parameters and to recreate the operating system kernel for your installation. Further information about these topics is system-dependent and is described in the installation documentation for your Linux computer.

### Multiple Buffer Pools

Depending on the individual requirements, it is possible to run different buffer pools of the same Natural version simultaneously on the same computer.

### Storing Objects in the Buffer Pool

When a user executes a program, a call is made to the buffer pool manager. The directory entries are searched to determine whether the program has already been loaded into the buffer pool. If it does not yet exist in the buffer pool, a copy is retrieved from the appropriate library and loaded into the buffer pool.

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

For this purpose, the buffer pool maintains a record of which user is currently using which object, and it detects situations in which a user exits Natural without releasing all its objects. It dynamically deletes unused or out-of-date objects to accommodate new objects belonging to other applications.

### Fast Locate

When a Natural object is executed, the Natural runtime system remembers the object name, the library (name, database ID and file number) and the address of the corresponding buffer pool directory entry. This data is referred to as “fast locate information”.

When a Natural object is executed again, the Natural runtime system passes the fast locate information to the buffer pool manager and performs a time-saving fast locate call. A fast locate call bypasses the normal locate procedure including the steplib search and the search in the buffer pool. It is therefore the most efficient way to locate an object. It provides significantly better performance of subsequent program loads especially when steplib libraries are involved in multi-user environments.

The address of an object saved as fast locate information is no longer valid once the object is removed from the buffer pool, overwritten by another object or reloaded to another buffer pool

location. If the fast locate call does not find the object at the given address, the object is searched in the buffer pool. If not found in the buffer pool, the object is reloaded from the system file.

This section covers the following topics:



### Fast Locate at Object Resume

Fast locate calls are issued when an object is accessed or resumed. An object resume operation is performed, for example, when an object continues to execute after a `CALLNAT` statement. For object resume operations, the Natural runtime system keeps fast locate information of the calling object for each program level on the internal stack.

### Fast Locate Table

The Natural runtime system keeps fast locate information about each accessed object in the internal fast locate table. The fast locate table also contains information about all libraries in which an object was searched. For a subsequent call, a fast locate is issued if the current library and associated steplibs are still the same.

The fast locate table is a hash table. The entries can be directly accessed without searching for an object name. The hash value is calculated from the object name. It determines the slot index number for the object. If another object has the same hash value (hash collision), a normal locate call is performed and the entry in the fast locate table is overwritten.

If an object for the library given in the fast locate table is neither found in the buffer pool nor in the system file (which means that the object has been deleted or moved to another library), a normal locate call with the full steplib search is scheduled automatically.

The **Locate Statistics** of the buffer pool monitor shows how many locate attempts were made and how many of these attempts were fast locate calls (see ***Statistical Information About the Buffer Pool***). These values can be used to review the efficiency of the fast locate table. If the fast locate table is activated for an application that calls the same objects many times, and if these objects are contained in a steplib library, the following applies:

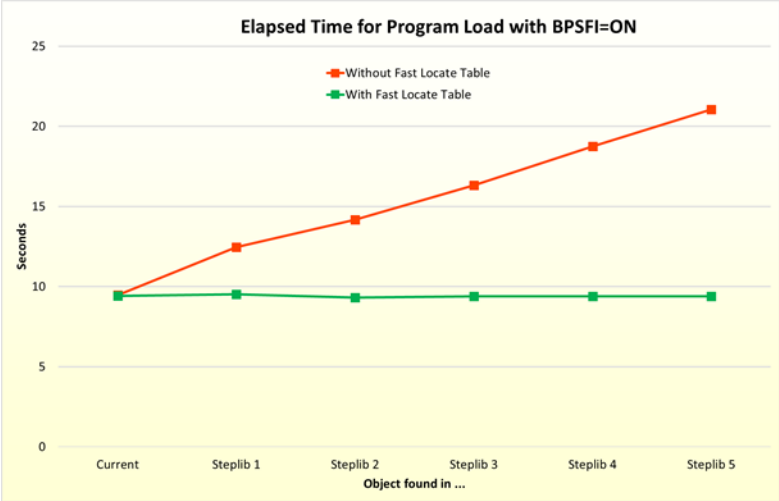
- The number of locate attempts should decrease significantly (compared with a deactivated fast locate table).
- The number of fast locate attempts should be close to the number of locate attempts.

### Fast Locate Table with BPSFI=ON

If the BPSFI (Object Search First in Buffer Pool) profile parameter is set to ON, the fast locate table is activated by default. It is initialized at the start of the Natural session and it is not cleared implicitly during the running session. It can be deactivated or cleared with the application programming interface USR3004N as described in the section *Maintaining the Fast Locate Table*.

### Performance with BPSFI=ON

In the following example, a subprogram is called 3,000,000 times. In the first test, the subprogram is found in the current library, then in Steplib 1, Steplib 2, and so on. The red line shows the elapsed time needed for the program load with a deactivated fast locate table, the green line with an activated fast locate table.




The diagram above shows that there is no performance improvement if the object is found in the current library. The more steplibs there are involved in object search operations, the higher is the performance improvement. For five steplibs, the program loads require less than half the time.

### Fast Locate Table with BPSFI=OFF

If the BPSFI profile parameter is set to OFF, the fast locate table is deactivated by default. It can be activated or cleared with the application programming interface USR3004N as described in the section *Maintaining the Fast Locate Table*. It is initialized at the start of the Natural session and it is implicitly cleared when the application is back on Program Level 0 (NEXT prompt).

Activation of the fast locate table for BPSFI=OFF can lead to unexpected results in the following scenario:

- The list of steplibs contains the libraries S1 and S2 whereby S1 is searched before S2.
- An object from S2 is accessed during the current Natural session.

- Another Natural session copies a new version of this object into S1.

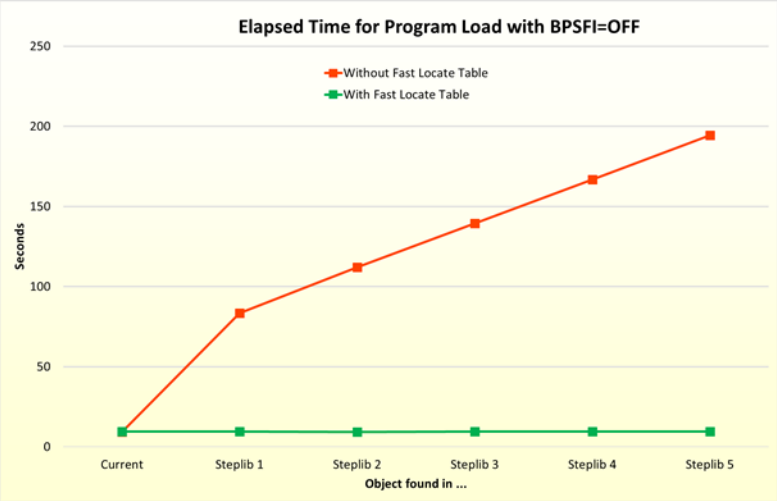
If the application is still running (not back on Program Level 0 in between) and the object is accessed again, the new version of the object will not be used.

If you want to activate the fast locate table when `BPSFI=OFF` is set, make sure that the scenario described above cannot occur.

If `BPSFI=ON` is set, object names should always be unique across all libraries involved in object search operations. This also guarantees that such scenarios do not occur.

### Performance with BPSFI=OFF

In the following example, a subprogram is called 3,000,000 times. In the first test, the subprogram is found in the current library, then in Steplib 1, Steplib 2, and so on. The red line shows the elapsed time needed for the program load with a deactivated fast locate table, the green line with an activated fast locate table.



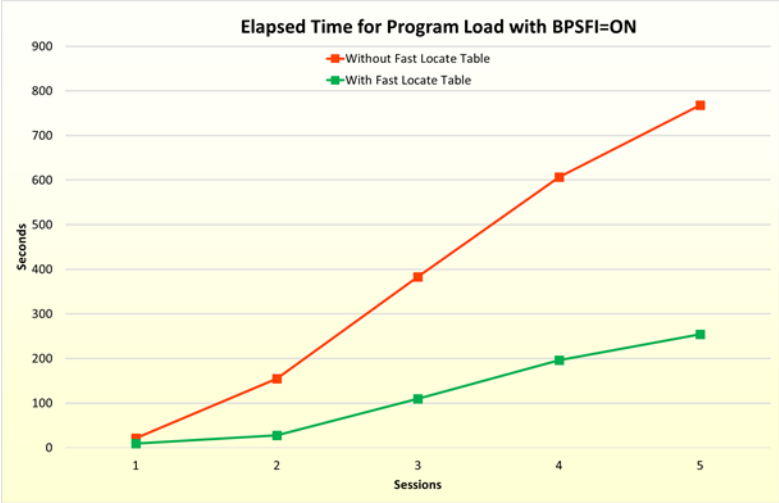

The diagram above shows that there is no performance improvement if the object is found in the current library. The more steplibs there are involved in object search operations, the higher is the performance improvement. Since the search operation on the system file is considerably slower than the search in the buffer pool, the improvement is much higher than the corresponding improvement when `BPSFI=ON` set. For five steplibs, the program load is about 20 times faster. If the fast locate table is activated, in general, the time needed for subsequent program loads for `BPSFI=OFF` is about the same as for `BPSFI=ON`, and it is always about the time needed to search for an object in the current library only.

### Performance in a Multi-User Environment

If an object is searched in a (read/write) buffer pool or on the system file, lock operations are issued to ensure that no other session performs changes concurrently. The lock operations serialize the access to the buffer pool, one session is processed after the other.

The fast locate table reduces the number of locate calls if steplibs are involved. Therefore, less lock operations are required, and overall performance of the buffer pool is improved.

In the following example, a subprogram is called 3,000,000 times, and the subprogram is always found in Steplib 5. In the first test, only one session is active. In the second test, two sessions execute the same application simultaneously, then three sessions, and so on. The red line shows the elapsed time needed for the program load with a deactivated fast locate table, the green line with an activated fast locate table.




As indicated in *Performance with BPSFI=ON*, the program load with a single session is more than 2 times faster if the object is found in Steplib 5 with `BPSFI=ON` set. If multiple sessions access the buffer pool simultaneously, the tests show that the performance can be 3 to 5 times faster.

### Maintaining the Fast Locate Table

Usage of the fast locate table can be activated and deactivated by calling the application programming interface (API) USR3004N. The API can also be used to get the current state of the fast locate table, to clear the fast locate table and to receive statistical data. The API is delivered in the SYSEXT library. For more information on using APIs, see the section *SYSEXT Utility - Natural Application Programming Interfaces* in the *Utilities* documentation.

#### To use API USR3004N

- Copy the USR3004N subprogram to the SYSTEM library, to the appropriate steplib library, or to the required library.

The function to be performed by USR3004N requires that the respective parameter value (`ON`, `OFF`, `STATE`, `CLEAR` or `COUNT`) is specified first in the `CALLNAT` statement. The parameter values can be specified in uppercase or lowercase. On return, `P-RETURN` contains the return code, whereby Return Code 0 indicates that the function performed successfully. All parameters are optional for compatibility with previous versions of the API on z/OS.

### To activate the fast locate table

- Call USR3004N with the following `CALLNAT` statement:

```
CALLNAT 'USR3004N' 'ON' P-STATE 2X P-RETURN-CODE
```

### To deactivate the fast locate table

- Call USR3004N with the following `CALLNAT` statement:

```
CALLNAT 'USR3004N' 'OFF' P-STATE 2X P-RETURN-CODE
```

### To retrieve the current state of fast locate table usage

- Call USR3004N with the following `CALLNAT` statement:

```
CALLNAT 'USR3004N' 'STATE' P-STATE 2X P-RETURN-CODE
```

If the `P-STATE` state field is `TRUE`, the fast locate table is used. The state field is returned for each API function.

### To clear the fast locate table

- Call USR3004N with the following `CALLNAT` statement:

```
CALLNAT 'USR3004N' 'CLEAR' P-STATE 2X P-RETURN-CODE
```

As described in *Fast Locate Table with BPSFI=OFF*, unexpected results can be encountered if the fast locate table is used with `BPSFI=OFF`. For `BPSFI=OFF`, the fast locate table is cleared when the application is back on Program Level 0 (`NEXT` prompt). A restart of the application therefore ensures that the latest version of the object is found.

Since a server in a client/server environment never reaches Program Level 0, you can clear the fast locate table by using the `CLEAR` function of USR3004N to ensure that the latest version of the object is found.

**To receive slot counts of the fast locate table**

- Call USR3004N with the following `CALLNAT` statement:

```
CALLNAT 'USR3004N' 'COUNT' P-STATE P-SLOTS-USED P-SLOTS-TOTAL
  P-RETURN-CODE
```

The counters indicate how well the hash function operates. The hash function is used to calculate the slot index number in the fast locate table.

| Field | Description |
|---|---|
| `P-SLOTS-USED` | Shows the number of slots in the fast locate table that are currently occupied.<br><br>The hash function operates well if this number increases with the number of objects accessed until close to the total number of slots. |
| `P-SLOTS-TOTAL` | Shows the total number of slots available in the fast locate table.<br><br>The used hash function requires that the total number is a prime number. There are 593 slots available in the fast locate table. |

## Read-Only Buffer Pool

A read-only buffer pool is a special buffer pool that only allows read access. If an object is not found in the read-only buffer pool, Natural issues error 82 (object not found). As no attempt is made to retrieve the missing object in the system files, all lock operations on the system file as well as on the buffer pool are skipped. Account data are gathered.

A read-only buffer pool is defined in the Configuration Utility (see also ***Setting up a Buffer Pool*** below).

The utility `NATBPSRV` expects a preload list in a file named *<bufferpool-name>.PRL* at the location of the Natural parameter files, which is defined in the local configuration file (installation assignments). For example, if the name of the read-only buffer pool is "ROBP", the file name must be *ROBP.PRL*.

A preload list can be generated using the Natural utility `CRTPRL`. This utility extracts the contents of a buffer pool and merges it with the existing preload data of a buffer pool.

The preload list in the *PRL* file contains records with comma-separated data in the following form:

```
database-ID,file-number,library,object-name,kind,type
```

The keywords in the file have the same meaning as the keywords shown by the DIR command of the NATBPMON utility.

With the exception of directory-describing records (the kind of object is D, which means the object is part of *FILEDIR.SAG*), a value must be assigned to all keywords. Examples:

| Keywords | NATBPSRV loads the following into the buffer pool |
|---|---|
| 222,111,MY_LIB,PGM1,G,P | Object code of program PGM1 from library MY_LIB which is located on database 222 and file number 111. |
| 222,113,*,*,D | *LIBDIR.SAG* which is located on FNAT=222,113. |
| 222,111,MY_LIB,*,D | *FILEDIR.SAG* from library MY_LIB which is located on FUSER=222,111 . |

Using a read-only buffer pool has the disadvantage that the application must be known in detail (as missing objects cannot be loaded). This means that all objects needed by an application must be specified in the preload list. In seldom cases, the complete set of objects needed by an application can be determined in advance.

#### Secondary Read/Write Buffer Pool

Natural can run with a read-only buffer pool as the primary buffer pool. Such a buffer pool is not modifiable. Objects missing in the read-only buffer pool cannot be loaded. If an object is not found in the read-only buffer pool, Natural issues error 82 (object not found). To avoid this, Natural can attach during execution to a secondary standard buffer pool (which allows read/write access) and activate the missing objects there. If a call to locate an object in the primary buffer pool fails, the secondary buffer pool operates as a backup buffer pool. The dynamic parameter BPID2 identifies the secondary buffer pool.

Other than for the read-only buffer pool, object locking through semaphores takes place each time the secondary buffer pool is accessed.

The preload list of the read-only buffer pool can be updated/enhanced by merging the contents of the secondary read/write buffer pool with the preload list of a read-only buffer pool using the utility CRTPRL.

#### Alternate Read-Only Buffer Pool

For a read-only buffer pool, it is possible to define the name of an alternate buffer pool in the Configuration Utility (see also *Setting up a Buffer Pool* below).

Using the SWAP command of the NATBPMON utility, which is only available for a read-only buffer pool, you can tag a read-only buffer pool as "obsolete". All Natural sessions attached to an obsolete buffer pool will detach from this buffer pool and will attach to the alternate buffer pool - but only if the alternate buffer pool is also a read-only buffer pool. The swap from one buffer pool to the other occurs either when Natural tries to load a new object (for example, when executing a CALLNAT or RETURN statement) or when Natural tries to interpret a command which has been put on the stack. The IPC resources (that is, the shared memory segment) of a buffer pool tagged as obsolete can be removed after issuing the SWAP command of the NATBPMON utility. This feature allows exchanging a buffer and its contents by another read-only buffer pool with updated contents without stopping Natural sessions.

Known issues: The IPCRM command of the NATBPMON utility will report an error trying to delete the semaphores associated to a read-only buffer pool.

#### Creating a Preload List Using the CRTPRL Utility

The Natural utility CRTPRL, which is located in the library SYSBPM, is used to create a preload list for a read-only buffer pool.

The utility uses the content of a source buffer pool as the basis for the preload list and checks whether the preload list already exists for a read-only (target) buffer pool:

- If the preload list exists, the existing data in the preload list is merged with the data from the source buffer pool, and the preload list is saved with the new content.
- If the preload list does not yet exist, it is created using the content from the source buffer pool.

The content of the resulting preload list determines the content of the read-only buffer pool. The preload list is read by the utility NATBPSRV which loads the corresponding objects into the read-only buffer pool.

### Buffer Pool with Enhanced Performance

A buffer pool with enhanced performance is a read/write buffer pool that is optimized for performance and scalability. The enhanced performance features are enabled automatically when you start a read/write buffer pool, unless only a runtime license is installed.

The buffer pool with enhanced performance combines the advantages of a read-only and read/write buffer pool.

Like the read-only buffer pool, the buffer pool with enhanced performance does not use locks if only read operations are performed, and read operations do not use locks and read operations

from multiple user sessions can be executed concurrently. Therefore, single-session performance matches that of a read-only buffer pool and exceeds that of the traditional read/write buffer pool, due to the reduced number of system calls.

The buffer pool with enhanced performance scales better than the read-only buffer pool and traditional read/write buffer pool. The performance advantage of the buffer pool with enhanced performance increases monotonically with increasing number of concurrent sessions actively using it.

The buffer pool with enhanced performance supports read and update operations. Thus, although a preload list can be used to seed the buffer pool on startup, its usage is optional, and even if specified, any objects missing from the list can be loaded later on demand without having to set up and maintain a secondary buffer pool for this purpose. Likewise, objects in the buffer pool with enhanced performance can be directly replaced, implying that application updates can be applied without having to switch to an alternate buffer pool with the `SWAP` command, as is the case with the read-only buffer pool.

The buffer pool with enhanced performance performs fewer update operations on internal data structures due to deferred structural updates that are combined and applied in a later consolidation operation. Furthermore, it only uses locks in conjunction with update operations. This enhances robustness by reducing the risk of a process dying in the middle of updating its data structures and/or while owning the lock.

The buffer pool with enhanced performance supports exclusive access for operations that need it unlike the read-only buffer pool. The operations that are not performance-critical, such as attaching to and detaching from the buffer pool, can be performed on their own, without interference from other users, thus further enhancing robustness. This function of the buffer pool with enhanced performance depends on the availability of the exclusive access, which is why it performs and scales better than the read-only buffer pool. Exclusive access also allows a true snapshot of the buffer pool status to be obtained via the `STATUS` command of the `NATBPMON` utility. The displayed statistical information is not modified as they it is gathered. In addition, the exclusive access is necessary for the `VERIFY` command, which is not available for the read-only buffer pool.

The extent to which the performance improvements shown in synthetic benchmarks apply to real-world applications depends on the number of concurrent sessions actively using the buffer pool and the frequency of buffer pool operations. For example, the longer the time spent in a called object (for example a subprogram, external subroutine, or function), the lower the impact of the buffer pool operations on overall performance.

When a buffer pool with enhanced performance is started, the `NATBPSRV` utility outputs the information `Enhanced performance cache created`, for example:

```
NATURAL/C Buffer Pool 9.3(932) of 07/18/2024 started (internal version 2).
Existing shared memory will be deleted.
Creation of shared memory completed.
Enhanced performance cache created.
Existing semaphores will be deleted.
Creation of semaphores completed.
Permanent IPC resources created.
Buffer pool is ready to run.
The server process completed successfully.
```

To check whether an existing buffer pool is a buffer pool with enhanced performance, attach to it with the NATBPMON utility and issue the STATUS command. If the output shows information about the operation type (Read operations, Sync read operations, Update operations, and Consolidations), the buffer pool is a buffer pool with enhanced performance. Otherwise, it is a buffer pool that was started with NATBPSRV utility version 9.3.1 or lower, a read-only buffer pool, or a standard Natural license could not be found. A standard Natural license allows installation of the extended environment with additional functionality, while a runtime license only enables a runtime environment.

### Restrictions of the Natural Buffer Pool

When using the Natural buffer pool, only minimum restrictions must be considered:

- When a Natural session hangs up, do not initially terminate it by using the Linux command kill -KILL (also kill -9), the terminal command break or the interrupt key.

  If this session is currently performing changes to the buffer pool internal data structures, an interruption may occur at a stage where the update is not fully completed. If the buffer pool internal data structures are inconsistent, this could have negative effects.

  Instead, use the Linux command kill -TERM (also kill -15) to terminate the hung-up session.

  **Note:** This can only happen when the Natural nucleus is executing buffer pool routines.

- All resources must be shared among all users of one Natural buffer pool. Group membership of a process is used to give access rights for the buffer pool. This means that the shared memory can be changed by all group members, but not by anyone else. The same applies to the semaphores.

  **Note:** All users of the same Natural buffer pool must belong to the same user group on the Linux operating system.

## Setting up a Buffer Pool

The buffer pool assignments are stored in the local configuration file. To set up a buffer pool, you have to specify specific values in the local configuration file using the Configuration Utility. For a list of these values, see *Buffer Pool Assignments* in the *Configuration Utility* documentation.

## Using the Utility NATBPSRV for Creating the Buffer Pool

The buffer pool is created using the utility `NATBPSRV`.

**Note:** The utility `NATBPSRV` should not be accessible to all Natural users, because it can cause damage to the work of other buffer pool users.

`NATBPSRV` allocates the resources required by the buffer pool and creates the permanent communication facilities (that is, shared memory and semaphores) used for the buffer pool. The necessary specifications for the resources and facilities are made with the Configuration Utility (see ***Setting up a Buffer Pool***).

The `NATBPSRV` utility should only be used during system startup, from within the startup procedure *natstart.bsh*.

By default, the buffer pool `NATBP` is started. If another buffer pool is to be started, you specify its name with the following `NATBPSRV` command line option:

```
NATBPSRV BP = buffer-pool-name
```

If `NATBPSRV` discovers in the process of creating a buffer pool that a buffer pool of the same name is already active, it deletes the already active buffer pool. If the deletion fails, `NATBPSRV` terminates with an appropriate error message.

## NATBPSRV Error Messages

`NATBPSRV` can issue the following error messages if the buffer pool that is to be created is meant to be a read-only buffer pool:

**Unable to attach to buffer pool. Return code ... received from bp_init.**

**Explanation** To load the objects described in the preload list, `NATBPSRV` attaches to the previously created buffer pool as a user. The attach process failed.

**Action** Contact Support.

**Unable to get parameter path.**

**Explanation** The path defined in the local configuration file identifying Natural's parameter files could not be established.

**Action** Contact Support.

**File ... is not accessible.**

**Explanation** The preload list is not accessible or not present.

**Action** Revise access rights or create a preload list.

**Unable to open file ...**

**Explanation** The preload list cannot be read.

**Action** Re-create preload list.

**Skipped erroneous record: '...'. Buffer pool may not operate correctly.**

**Explanation** An invalid record was found in the preload list. The record is skipped and the load process is continued. There may arise errors in your application due to missing objects.

**Action** Correct the record if it has been created manually, or contact Support.

**Unable to retrieve LIBDIR.SAG in FNAT(...,...). Application will not run.**

**Explanation** *LIBDIR.SAG* was not found. An application depending on `FNAT(...,...)` will not run.

**Action** Correct the record if it has been created manually, or contact Support.

**Buffer pool manager returned with error code ... . Buffer pool is not operational.**

**Explanation** *FILEDIR.SAG* could not be loaded into the buffer pool. The buffer pool is either too small to hold *FILEDIR.SAG* or *FILEDIR.SAG* is damaged. The previously listed message tells which *FILEDIR.SAG* is causing the trouble.

**Action** Correct the record if it has been created manually, or contact Support.

**Buffer pool manager returned with error code ... . Error ... occurred.**

| | |
|---|---|
| **Explanation** | An error occurred loading an object into the buffer pool. |
| **Action** | Normally, the size of the buffer pool is too small. Increase its size and repeat the operation. If the problem remains, contact Support. |

**Object ... in library ... on system file (...,...) not found. Application may not run.**

| | |
|---|---|
| **Explanation** | The preload record processed pointed to an object that was not found. This normally happens if an application is modified and the corresponding preload list is not updated. |
| **Action** | Remove/revise preload record in question |

**Preload executed. Buffer pool is ready to run.**

| | |
|---|---|
| **Explanation** | All preload records were processed. The buffer pool is unlocked and Natural can access that buffer pool. |

## Monitoring the Buffer Pool

The Buffer Pool Monitor is used to oversee the buffer pool's activity during its operation. The Buffer Pool Monitor can also be used to shut down the buffer pool when Natural must be stopped on a computer.

The Buffer Pool Monitor collects information on the current state of your Natural buffer pool.

If multiple buffer pools are active on the same computer and an object that is loaded to more than one buffer pool is modified by a Natural process, the object will only be removed from the buffer pool to which the modifying Natural process is attached.

For detailed information for how to use the Buffer Pool Monitor, see ***Using the Buffer Pool Monitor (NATBPMON)***.

## Trouble Shooting

This section describes problems that may occur when using the Natural buffer pool and how to solve them.

It is assumed that you are familiar with the Linux commands `ipcs` and `adb`.

The following are typical command output examples, with an explanation of what went wrong during execution.

### Problem 1

Either Natural or the Natural Buffer Pool Monitor (NATBPMON **utility**) cannot be started.

#### Examples

The following examples describe the most typical problems you are likely to encounter as a Natural administrator or user. These problems occur when you start Natural or the Natural Buffer Pool Monitor, and the buffer pool is not active.

- You try to start Natural with the following command:

```
natural bp = sag
```

  The following message appears:

```
Natural Startup Error:    16
Unable to open Buffer Pool,
Buffer Pool error:   "unexpected system call error occurred "  (20)
Global shared memory could not be attached.:   shmkey = 11111111

Operating System Error 2 - No such file or directory
```

- You try to start the Natural Buffer Pool Monitor with the following command:

```
natbpmon bp = sag
```

  When you enter the WHO command at the NATBPMON prompt, the following message appears:

```
Buffer Pool error:   unexpected system call error occurred  (20)
Global shared memory could not be attached.:   shmkey = 11111111
Operating System Error 2 - No such file or directory
```

#### Solution

1. Start the buffer pool service as described in ***Using the Utility NATBPSRV for Creating the Buffer Pool***.
2. Use the Linux command ipcs to verify the existence of the necessary semaphores and the shared memory:

```
ipcs -m -s
```

   This results in the following output:

```
IPC status from /dev/kmem as of Mon 23-MAY-2005 12:03:24.30
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m    807 0x4e425031 --rw-rw----      sag  natural
Semaphores:
s     85 0x4e425031 --ra-ra----      sag  natural
```

**Note:** The above output was edited to exclude memory segments and semaphores that do not belong to the Natural buffer pool.

If you cannot find a shared memory segment or a set of semaphores with the key you assigned them, the buffer pool was not started.

### Problem 2

The Natural buffer pool and a Natural utility are not of the same Natural version.

#### Examples

If a utility tries to use the buffer pool, the utility and buffer pool versions are checked for equality. If they differ, the access is denied and an error message is output.

- You try to start Natural and the following message appears:

```
Natural Startup Error 16: Unable to open buffer pool.
Buffer pool error: "Buffer pool does not correspond with your version of ↩
Natural"(25).
Internal version of buffer pool is 0 but requested internal version is 1. ↩
```

- You try to start the Natural Buffer Pool Monitor and the following message appears:

```
Buffer pool error: Buffer pool does not correspond with your version of Natural ↩
(25).
Internal version of buffer pool is 0 but requested internal version is 1.
```

#### Solution

Verify that your Natural version corresponds to your buffer pool version number and that the internal buffer pool version (BP version) is also correct. Restart the buffer pool with the same version as Natural but make sure that no other users are active.

**Important:** The **internal buffer pool version number (BP version)** can vary in between service pack releases (third digit of the product version number). For example, a buffer pool that has been initiated using Natural Version *vrs* cannot be used with Natural Version *vr(s+1)* and vice versa.

## Shutting Down and Restarting the Buffer Pool

Usually it should not be necessary to shut down and restart the buffer pool. This may only be necessary if the buffer pool should become unusable due to serious internal errors in the buffer pool, which is extremely unlikely to occur, or because the parameters defining the buffer pool structure became obsolete.

If the `NATBPMON` **utility** is still able to access the buffer pool, proceed as follows:

1. Shut down the buffer pool with the `SHUTDOWN` command of the `NATBPMON` utility.

   Once the `SHUTDOWN` command is executed, new users are denied access to the buffer pool.

   **Tip:** Active buffer pool users can be monitored by issuing the `WHO` and `STATUS` commands of the `NATBPMON` utility.

2. After the last user has stopped accessing the buffer pool, buffer pool resources can be deleted by issuing the `IPCRM` command of the `NATBPMON` utility.
3. To restart the buffer pool, call the file *natstart.bsh* from a sufficiently privileged account.

If you have super user rights, you can use the `FORCE` option of the `SHUTDOWN` command:

1. Shut down the buffer pool with the `SHUTDOWN FORCE` *`grace-period`* command of the `NATBPMON` utility.

   This command - like the `SHUTDOWN` command without options - denies new users access the buffer pool. However, the terminate signal `SIGTERM` is sent to all active Natural sessions, forcing them to log off from the buffer pool.

   If the optional parameter *`grace-period`* is omitted, this command waits until all active sessions have performed their shutdown processing and then executes the `IPCRM` command of the `NATBPMON` utility .

   If the optional parameter *`grace-period`* has been specified, `NATBPMON` waits the specified number of seconds before it executes its `IPCRM` command - regardless of the closedown status of the sessions logged on to the buffer pool. Therefore, the value defined for the grace period should be sufficiently large to allow the sessions to terminate in time.

   **Note:** `SHUTDOWN FORCE 0` is the same as `SHUTDOWN FORCE` (without the parameter *`grace-period`*).

2. To restart the buffer pool after successful execution of the `SHUTDOWN FORCE` command, call the file *natstart.bsh* from a sufficiently privileged account.

If the NATBPMON utility is not able to perform a clean shutdown of the buffer pool, the buffer pool must be deleted by using operating system commands:

1. Use the Linux command ipcs to find out the status of the buffer pool's shared memory and semaphores:

   ```
   ipcs -a -m
   ```

   In the column **NATTCH** of the output of an ipcs -m -a command, you can see the number of processes currently attached to a shared memory segment. For example:

   ```
   IPC status from /dev/kmem as of Mon May 23 12:15:38.39 2002
   T         ID     KEY        ...  OWNER  GROUP  ... NATTCH    SEGSZ
   Shared Memory:
   m        707 0x4e425031  ... sag    natural ...      7   153600
   ```

2. It is highly probable that the number of processes attached to shared memory incorporates a Natural nucleus or the NATBPMON utility currently running. Inform the users who run these processes and ask them to terminate their sessions or terminate them yourself by using the Linux command kill once you have found out their process IDs using the ps command.
3. Once you are sure that no one is using the buffer pool for important work, its resources can be deleted by using the Linux command ipcrm. For example:

   ```
   ipcrm -M 0x4e425031 -S 0x4e425031
   ```

   The values specified for the -M and -S options must be those that were specified inside the parameter file used to start the buffer pool.

   Be careful when you delete shared memory and semaphores using the Linux command ipcrm. If you accidentally delete the wrong resource, this might have a serious impact on other software products running on your computer.
4. The result of deletion can be verified by using the Linux command ipcs again.

   If there are still some memory segments or message queues displayed, they could belong to other software, or they are marked for deletion because some other process is still attached to them.

If the buffer pool cannot be started after removing the shared memory and semaphores, you should consider either rebooting your computer or contacting Support.

# 6 Using the Buffer Pool Monitor (NATBPMON)

See also *Natural Buffer Pool* which provides general information on the buffer pool and explains how to start the buffer pool.

**Caution:** This utility should not be generally accessible to all users of Natural, because its use can cause damage to the work of other users of the buffer pool.

## Invoking the NATBPMON Utility

You can invoke the NATBPMON utility either for the default buffer pool NATBP or for another existing buffer pool.

The NATBPMON administrator can always invoke the utility. If the maximum buffer pool user limit is reached, the administrator accesses the utility as an emergency user. Only one additional buffer pool administrator can use NATBPMON at a time. You can define a Natural buffer pool administrator in the local configuration file in the Configuration utility.

### To invoke the NATBPMON utility

1 If the default buffer pool NATBP is to be used, enter the following command in the command line:

```
NATBPMON
```

Or:

If another buffer pool is to be used, enter the following command in the command line:

```
NATBPMON BP=buffer-pool-name
```

The following prompt appears:

```
NATBPMON>
```

2 If you want the NATBPMON utility to terminate with an appropriate error message, add GIVERC to your command in the command line.

## NATBPMON Commands

The following commands can be entered at the NATBPMON prompt:

| Command | Description |
|---|---|
| CLEAR | This is the same as the ZERO command. |
| CORPSES | Displays the list of corpses. A corpse is an object that has been deleted, but was still being used in the buffer pool when its deletion took place. Once this object is no longer used, it will automatically disappear from the list of corpses.<br><br>**Note:** The column **cusr** which is shown with the DIR command indicates if an object is being used. |
| DELETE {*pattern*\|[*]} | Deletes an object from the buffer pool. All objects can be deleted from the buffer pool by using an asterisk (*). A pattern is used to specify a collection of objects, similar to current operating systems which allow the specification of a class of files with wildcards. For further information, see ***Specifying a Pattern***. |
| DIR {*pattern*\|[*]} | Displays a directory containing all objects in the buffer pool. For further information, see the sections ***Specifying a Pattern*** and ***Displaying the Objects in the Buffer Pool***. |
| DUMP | Used for error analysis.<br><br>**Important:** Do not use this command unless you are requested to do so by Support. |
| EXIT | Exits the NATBPMON utility. |
| FIN | Exits the NATBPMON utility. This is the same as the EXIT command. |
| HELP | Displays a list of all available commands of the NATBPMON utility. |
| IPCRM | Frees the resources allocated to the buffer pool. This command should only be used following a SHUTDOWN command when there are no active users. |
| KILL *n* | Kills the specified buffer pool user. *n* is the number of the user to be "killed". This number corresponds to the index number as displayed by the WHO command. |
| PARAM | Displays the buffer pool settings. For further information, see ***Displaying the Buffer Pool Settings***. |
| QUIT | Exits the NATBPMON utility. This is the same as the EXIT command. |
| SHUTDOWN [FORCE [*grace-period*]] | Without the option FORCE: Shuts down the buffer pool. No new processes will be able to use the buffer pool once this command has been issued. The NATBPMON utility is able to run with a buffer pool which has the shutdown status "pending"; all commands of the NATBPMON utility are available in this case. As soon as all users have stopped using the buffer pool, the buffer pool's resources can be deleted with the IPCRM command.<br><br>The option FORCE requires NATBPMON to be executed with super user rights. After SUDO or SU has validated the password and given control to NATBPMON, any new sessions will be inhibited to log in and the terminate signal SIGTERM will be sent to all active Natural sessions. NATBPMON will then wait the number of seconds defined with the parameter *grace-period* before the IPC resources used by the buffer pool are removed |

| Command | Description |
|---|---|
| | from the system. If the optional parameter *grace-period* is omitted (or set to 0), NATBPMON will wait until all processes performed their cleanup processing. This process can be considered as an emergency stop. If it is executed without super user rights, no action takes place and a message reporting the incapability to execute the command is sent. See also ***Shutting Down and Restarting the Buffer Pool***.<br><br>**Note:** To start the buffer pool after shutdown, you can use the utility NATBPSRV. |
| STATUS | Displays statistical information about the buffer pool. For further information, see ***Statistical Information About the Buffer Pool***. |
| SWAP | Only available for a **read-only** buffer pool. Tags a read-only buffer pool as "obsolete". All Natural sessions attached to such a buffer pool will detach from that buffer pool and attach to the alternate buffer pool. |
| VERIFY [NOW\|ON\|OFF] | This command cannot be used on a read-only buffer pool, because the required exclusive access is not available for it.<br><br>Performs, enables, or disables buffer pool consistency checks. If the command is entered on its own or with the NOW option, any existing consistency error is reported, otherwise consistency checks are performed and the result is returned. If the option ON is provided, consistency checks are performed after each consolidation (see Consolidations in the output of the STATUS command) until a consistency error is detected or until disabled again with the OFF option. By default, no automatic consistency checks are performed. |
| WHO | Displays a list of all users who are using the buffer pool. The following statistics are displayed: a number that the NATBPMON utility automatically assigns to each buffer pool user (index) and the user ID, terminal ID and process ID of the process using the buffer pool (tid). |
| WRITE | Writes a buffer pool object onto the disk. You are prompted to specify an index and a file name.<br><br>**Note:** The column "indx" which is shown with the DIR command shows the index numbers. |
| ZERO | Resets to 0 all counters that are displayed by the STATUS command. |

## Displaying the Objects in the Buffer Pool

The DIR command displays a list of objects. This list contains the following information:

| Column | Explanation | |
|---|---|---|
| **indx** | A number that the NATBPMON utility automatically assigns to an object when it is loaded into the buffer pool. | |
| **cusr** | The current number of users that are using an object in the buffer pool. | |
| **pusr** | The peak number of concurrent activations of an object in the buffer pool. | |
| **nusg** | The number of times an object has been activated in the buffer pool. | |
| **g** | Specifies whether an object is being loaded into the buffer pool from the system file. Has one of the following values: | |
| | 0 | The object is not being loaded. |
| | 1 | The object is being loaded. |
| **size** | Specifies the size (in bytes) of an object in the buffer pool. | |
| **gpv** | The version number of the generated program. | |
| **key** | Specifies the following information about an object: | |
| | D | Database ID. |
| | F | File number. |
| | L | The library in which the object is located. |
| | N | The name of the object. Numbers and the at sign (@) indicate chunks of *FILEDIR.SAG* for the currently loaded library. |
| | K | The kind of object (G=generated object module; S=source; D=part of *FILEDIR.SAG*). |
| | T | The object type (which is blank when D is shown in the K field). |

When the DIR command is issued, all objects in the pool will be displayed in a notation similar to the following:

```
indx:    index of the element
 cusr:    current number of concurrent users
 pusr:    peak number of concurrent users
 nusg:    number of usages
 g   :    set if object is generating
 gpv :    version of generated program

indx | cusr | pusr |  nusg  |  g  |  size  |  gpv  |  key
-----+------+------+--------+-----+--------+-------+-------------------------------------------------
 1   |  0   |  1   |    4   |  0  |   920  |       |  (D=99 F=101 L="DEMO" N="SEL-MAP" K='G' T='M')
 2   |  1   |  7   |    2   |  0  |  3096  |       |  (D=99 F=101 L="DEMO" N="EMWND" K='G' T='P')
 3   |  4   |  9   |    4   |  0  |   604  |       |  (D=99 F=101 L="DEMO" N="HDR" K='G' T='P')
 4   |  2   |  3   |    7   |  0  |   412  |       |  (D=99 F=101 L="RPA" N="MMUPROG1" K='G' T='P')
 5   |  0   |  1   |    5   |  0  |   372  |       |  (D=99 F=101 L="RPA" N="MMUPROG2" K='G' T='P')
 6   |  0   |  5   |    4   |  0  |   372  |       |  (D=99 F=101 L="RPA" N="MMUPROG3" K='G' T='P')
```

## Specifying a Pattern

A pattern can be specified with the commands `DIR` and `DELETE`. The examples in this section apply to the `DIR` command.

To select some objects, it is possible to restrict the values of certain key fields by specifying a matching pattern expression.

To restrict the allowed field values of a given field, the following pattern notation must be used:

```
name=expression
```

You can specify multiple patterns by separating them with a comma.

The specified patterns must all match their corresponding fields in order to accept the entire key.

The expression accepts the specification of the wildcard characters "*" and "?".

The character "*" matches any or no occurrences of a sequence of characters, and the wildcard character "?" matches exactly one specific character.

**Examples**

To select all objects of type `P` in the sample above, the following command would be used:

```
DIR T=P
```

To select all programs in the demo library, the following command would be used:

```
DIR T=P, L=DEMO
```

To select all objects containing an `M` in their name, the following command would be used:

```
DIR N=*M*
```

## Displaying the Buffer Pool Settings

The following settings are displayed with the `PARAM` command:

```
SHM active since .......: 13-AUG-2024 19:39:21, Version 9.3(932) BP version 2
Last time cleared ......: 13-AUG-2024 19:39:21

Bpid ...................: TESTRW
Readonly ...............: no
Shmkey .................: 0x39211291
Semkey .................: 0x39211291
Memsize ................:      10485748
Maxusers ...............:           200
```

| | |
|---|---|
| **SHM active since** | Date and time when the buffer pool was started, the version number of the program that started it and created the shared memory, and the internal buffer pool format version that is incremented on every structural change. |
| **Last time cleared** | Date and time when the buffer pool statistical information was most recently cleared (either implicitly on buffer pool creation, or explicitly via the `CLEAR` or `ZERO` command). |
| **Bpid** | Buffer pool ID. |
| **Readonly** | Indicates whether this is a special buffer pool which only allows read access. |
| **Shmkey** | Unique name used to create a buffer pool or to connect to a buffer pool. |
| **Semkey** | Unique name used to synchronize accesses to the buffer pool memory. |
| **Memsize** | Size of the available shared memory. |
| **Maxusers** | Maximum number of users that can have simultaneous access to the buffer pool. |

See *Buffer Pool Assignments* in the *Configuration Utility* documentation.

## Statistical Information About the Buffer Pool

The following statistics are displayed with the `STATUS` command:

```
Buffer Pool Monitor version 9.3(932F00) of 07/18/2024
NATBPMON>st
SHM active since .......: 13-AUG-2024 19:39:21, Version 9.3(932F00)
Last time cleared ......: 13-AUG-2024 19:39:21
Bpid ...................: TESTRW
Allocated memory (b) ...:        255280 Max users ............:           200
Smallest allocation ....:            48 Current users ........:             2
Largest allocation .....:         97624 Peak users ...........:             2
Free memory (b) ........:      10230480 Dead users purged ....:             0
Smallest free ..........:           368
Largest free ...........:      10228240 Peak parallel usages .:             1

Dormant objects ........:            12 Smallest object (b) ..:           108
Active objects .........:             0 Largest object (b) ...:         31228
Generating objects .....:             0 Total object sizes ...:        120638
Obsolete objects .......:             0
Dormant objects purged .:             0 Object reusage factor :          6.29

Attempted locates ......:           183 Stored objects .......:             0
Attempted fast locates .:            79 Loaded objects ........:            24
Successful fast locates :            62 Activated objects ....:           151
Percent ................:         78.48 Aborted loads .........:             0

Read operations ........:           334 Update operations ....:            81
Sync read operations ...:             0 Consolidations .......:             5
```

| General Information | |
|---|---|
| **Buffer Pool Monitor version** | Version of the buffer pool including its fix level (enclosed in brackets as *vrs*F*nn*, where *vrs* is the buffer pool version and *nn* is its fix level). |
| **SHM active since** | Date and time when the buffer pool was started, and the version number and fix level of the program that started it and created the shared memory. |
| **Last time cleared** | Date and time when the buffer pool statistical information was most recently cleared (either implicitly on buffer pool creation, or explicitly via the `CLEAR` or `ZERO` command). |
| **Bpid** | Buffer pool ID. If applicable, the read-only and swap status is shown enclosed in brackets. |
| **Memory Allocation** | |
| **Allocated memory (bytes)** | Total of all allocated memory. |
| **Smallest allocation** | Smallest amount of allocated memory. |
| **Largest allocation** | Largest amount of allocated memory. |
| **Free memory (bytes)** | Total of all free memory. |
| **Smallest free** | Smallest amount of contiguous free memory. |
| **Largest free** | Largest amount of contiguous free memory. |
| **User Statistics** | |
| **Max users** | Maximum number of users that can have simultaneous access to the buffer pool. See *Buffer Pool Assignments* in the *Configuration Utility* documentation. |

| | |
|---|---|
| **Current users** | Number of users currently using the buffer pool. |
| **Peak users** | Peak number of users that have been using the buffer pool. |
| **Dead users purged** | Number of inactive users that have been deleted from the buffer pool. This number should be close to 0 (zero). An increment of this number indicates that entries for buffer pool users (i.e. Natural sessions) were canceled or killed unconditionally. Each time an entry for such a user is identified by the buffer pool manager, this number is incremented and cleanup is performed to remove residuals which have been left in the buffer pool by a canceled session. |
| **Peak parallel usages** | The maximum number of users that have been concurrently using one of the objects in the buffer pool. |
| **Object Use Statistics** | |
| **Dormant objects** | Number of available, but inactive objects. These objects are in the buffer pool, but are not being used. They are available for later use and will become active objects as soon as a buffer pool user requests their availability. |
| **Active objects** | Number of active objects. These objects are currently in use by one or more buffer pool users. |
| **Generating objects** | Number of objects that are currently being loaded into the buffer pool. These objects will become available as soon as the load operation completes. |
| **Obsolete objects** | Number of objects that are to be deleted from the buffer pool, but are still being used. These objects can be displayed by using the `CORPSES` command. An obsolete object is removed from the buffer pool as soon as all users who activated this object have released this object. In a production environment, this number should be 0 (zero). A value other than zero indicates that objects were deleted either using the `DELETE` command of `NATBPMON` or became obsolete because new objects were created (for example, due to a `CATALOG` command). |
| **Dormant objects purged** | The number of unused objects deleted from the buffer pool to make room for newly loaded ones. |
| **Object reusage factor** | Average number of times an object was reactivated. This number is the ratio of the number of object activations to the number of objects loaded into the buffer pool. |
| **Object Size Statistics** | |
| **Smallest object (bytes)** | Size of smallest object in the buffer pool. |
| **Largest object (bytes)** | Size of largest object in the buffer pool. |
| **Total object sizes** | Total size of all objects in the buffer pool. |
| **Locate Statistics** | |
| **Attempted locates** | Number of successful and failed object locates. This number tells you how many times the buffer pool manager was asked to locate an object in the buffer pool. |
| **Attempted fast locates** | Number of attempted activations with known slot. This is the number of object activations when the former location of an object was known. It is highly probable that an object can be found in the same place in the buffer pool when it is reactivated. |

| | |
|---|---|
| **Successful fast locates** | Number of successful fast locates. |
| **Percent** | Percentage of successful fast locates. |
| **Object Loading Statistics** | |
| **Stored objects** | The number of objects stored in the buffer pool. This is the number of objects that were stored into the buffer pool and which were not loaded from the system file. |
| **Loaded objects** | The number of objects loaded from the system file. Each time an object is not found in the buffer pool, it is loaded from the system file. This number is increased each time an object is successfully loaded into the buffer pool. |
| **Activated objects** | The number of activated objects. Activation is the process of marking an object which is found in the buffer pool as "in use" by a buffer pool user. |
| **Aborted loads** | The number of load operations that were aborted due to memory shortages within the buffer pool, or due to an error when loading an object into the buffer pool. This number should not vary in a noticeable way. |
| **Operation Type Statistics** | |
| **Read operations** | Not available for read-only buffer pools or if only a Natural Runtime license is found.<br><br>The total number of read operations, both unsynchronized and synchronized. Unsynchronized read operations are fast. Furthermore, multiple operations of this type can be executed in parallel. |
| **Sync read operations** | Not available for read-only buffer pools or if only a Natural Runtime license is found.<br><br>The number of read operations that needed to be synchronized with update operations. Operations of this type cannot be executed in parallel with any other operations. |
| **Update operations** | Not available for read-only buffer pools or if only a Natural Runtime license is found.<br><br>The number of update operations. These are operations that explicitly request exclusive access to the buffer pool (for example to modify it or to obtain a data snapshot). Operations of this type cannot be executed in parallel with any other operations. |
| **Consolidations** | Not available for read-only buffer pools or if only a Natural Runtime license is found.<br><br>For performance reasons and to allow multiple read operations to be executed in parallel, some internal administrative tasks associated with read and update operations are not performed immediately. Instead, they are combined and performed asynchronously in a later consolidation operation. The value shown here is the number of such operations. Operations of this type cannot be executed in parallel with any other operations. |

# 7 Natural in Batch Mode

This chapter contains special considerations that apply when running Natural in batch mode.

## About Batch Mode

Natural distinguishes between two processing modes:

- interactive mode (via the Natural Main Menu)
- batch mode

The main difference between these two modes is that in interactive mode, the commands and data are input by the user by means of the keyboard and the output is displayed on a screen. In batch mode, input is read from a file and output is written to a file - without user interaction.

When Natural is run as a batch job, no interaction between Natural and the person who submitted the batch job is necessary. The batch job consists of programs that are executed sequentially and that receive sequential input data.

Batch mode is useful for mass data processing on a regular basis.

## Starting a Natural Session in Batch Mode

Batch mode is activated with the parameter `BATCHMODE`.

### To start a Natural session in batch mode

1 Start Natural with the **dynamic** parameter `BATCHMODE` as shown below:

```
natural BATCHMODE
```

The above call (where only the `BATCHMODE` parameter is specified) assumes that the required input and output channels have already been defined in the Configuration Utility. For information on the input and output channels, see ***Using Natural in Batch Mode*** later in this section). For information on the batch-mode-relevant profile parameters in the parameter file, see *Batch Mode* in the *Configuration Utility* documentation.

It is also possible to add the required input and output channels as dynamic parameters to the above call. This is illustrated in ***Sample Session for Batch Mode*** later in this section. Any input and output channels that are specified as dynamic parameters with the above call override the channel definitions in the parameter file.

2 Check the file which has been defined as the output channel. At its end, this file should contain the message that your session has terminated normally.

## Terminating a Natural Session in Batch Mode

A Natural session in batch mode is terminated when one of the following is encountered during the session:

- the system command FIN in the **batch input file**, or
- a TERMINATE statement in a Natural program which is being executed.

**Note:** When an end-of-input condition occurs in the batch input file, the batch session is also terminated. In this case, the file which has been defined as the output channel contains a message which indicates an unexpected end.

## Using Natural in Batch Mode

To **start** a Natural session in batch mode you have to specify the dynamic parameter BATCHMODE. In addition, input and output channels have to be defined as described below.

**Important:** The input channels CMSYNIN and/or CMOBJIN and the output channel CMPRINT are always required for batch mode.

The following topics are covered below:



### Input and Output Channels

The following parameters are available for batch mode:

| Parameter | Description |
|---|---|
| CMSYNIN | Defines the batch input file which contains the Natural commands and (optionally) data to be read by INPUT statements during execution of Natural programs. |
| CMOBJIN | Defines the batch input file which contains the data to be read by INPUT statements. This data can alternatively be placed in the file defined with the parameter CMSYNIN, immediately following the relevant RUN or EXECUTE command. |
| CMPRINT | Defines the batch output file for the output resulting from DISPLAY, PRINT and WRITE statements in a Natural program. |
| CMPRT*nn* | Defines an output file for additional reports referenced by any Natural program executed during the session. *nn* is a two-digit decimal number in the range from 01 to 31 which corresponds to the report number used in a DISPLAY, PRINT or WRITE statement. |

| Parameter | Description |
|---|---|
| CMWRK*nn* | Defines a work file referenced by any Natural program executed during the session. *nn* is a two-digit decimal number in the range from 01 to 32 which corresponds to the number used in a READ WORK FILE or WRITE WORK FILE statement. |
| NATLOG | Used to log messages that could not be written to the batch output file defined with the parameter CMPRINT. It is recommended to enable NATLOG in batch mode. |

### Code Pages for the Input and Output Files

The following parameters are used to specify the code pages in which the input files are encoded and in which the output file shall be encoded.

| Parameter | Description |
|---|---|
| CPSYNIN | Specifies the code page in which the batch input file for commands is encoded. This file is defined with the parameter CMSYNIN. |
| CPOBJIN | Specifies the code page in which the batch input file for data is encoded. This file is defined with the parameter CMOBJIN. |
| CPPRINT | Specifies the code page in which the batch output file shall be encoded. This file is defined with the parameter CMPRINT. |

Encoding for CMSYNIN and CMOBJIN:

- If a code page is specified for one of the input files CMSYNIN or CMOBJIN, it is assumed that the data in the input file is encoded using this code page.
- If no code page is specified for one of the input files CMSYNIN or CMOBJIN, it is assumed that the data in the input file is encoded using the default code page specified in the Natural parameter CP.
- If no code page is specified in the Natural parameter CP, it is assumed that the data in the input file is encoded using the current system code page.

Encoding for CMPRINT:

- If a code page is specified for the output file CMPRINT, the output data will be encoded using this code page.
- If no code page is specified for the output file CMPRINT, the output data will be encoded using the default code page specified in the Natural parameter CP.
- If no code page is specified in the Natural parameter CP, the output data will be encoded using the current system code page.

If the encoding/decoding fails (for instance if a character is written to CMPRINT that is not contained in the code page used to encode the file), the batch job terminates with a startup error 42 (batch mode driver error) that specifies the file on which the encoding/decoding error occurred.

Note that it is possible in particular to specify UTF-8 as code page in each of these parameters. This allows for reading and writing Unicode data encoded in UTF-8.

## Sample Session for Batch Mode

This example demonstrates how to start Natural in batch mode. A simple Natural program is executed and data items are taken from the batch input file. After the items are processed with the `INPUT` statement, a `DISPLAY` statement follows, which writes the data to the batch output file. Then, Natural terminates.

This example uses the program `RECCONT` which is stored in the library `SYSEXBAT`.

**Note:** See the text `A-README` in the library `SYSEXBAT` for information on the objects that are stored in this library.

The sample session is invoked with the following call:

```
natural BATCHMODE CMSYNIN=cmd.txt CMOBJIN=data.txt CMPRINT=out.txt NATLOG=ALL
```

**Note:** This call assumes that all files can be found in the current directory and that the output is written to this directory. If the files are located in different directories or if the output is to be written to a different directory, you have to specify the path.

The parameters in the above call are described below:

`BATCHMODE`
: The parameter `BATCHMODE` enables batch mode and sets the value of the system variable `*DEVICE` to `BATCH`.

`CMSYNIN=cmd.txt`
: The batch input file *cmd.txt* is a text file which is stored in your file system. The content of this file is shown below. It contains Natural system commands for logging on to the library `SYSEXBAT`, executing the Natural program `RECCONT`, and terminating the Natural session.

```
LOGON SYSEXBAT
EXECUTE RECCONT
FIN
```

The Natural program `RECCONT` has the following content:

```
DEFINE DATA
LOCAL
  1 #firstname   (A10)
  1 #lastname    (A10)
END-DEFINE
INPUT (IP=OFF AD=M) #firstname #lastname
DISPLAY #firstname #lastname
END
```

`CMOBJIN=data.txt`

The `INPUT` statement in the program `RECCONT` uses the data which is defined in the batch input file *data.txt*. This is a text file which is stored in your file system. The content of this file is shown below.

```
Ben %
Smith
```

**Note:** The percent character (%) indicates that the information continues in the next line.

`CMPRINT=out.txt`

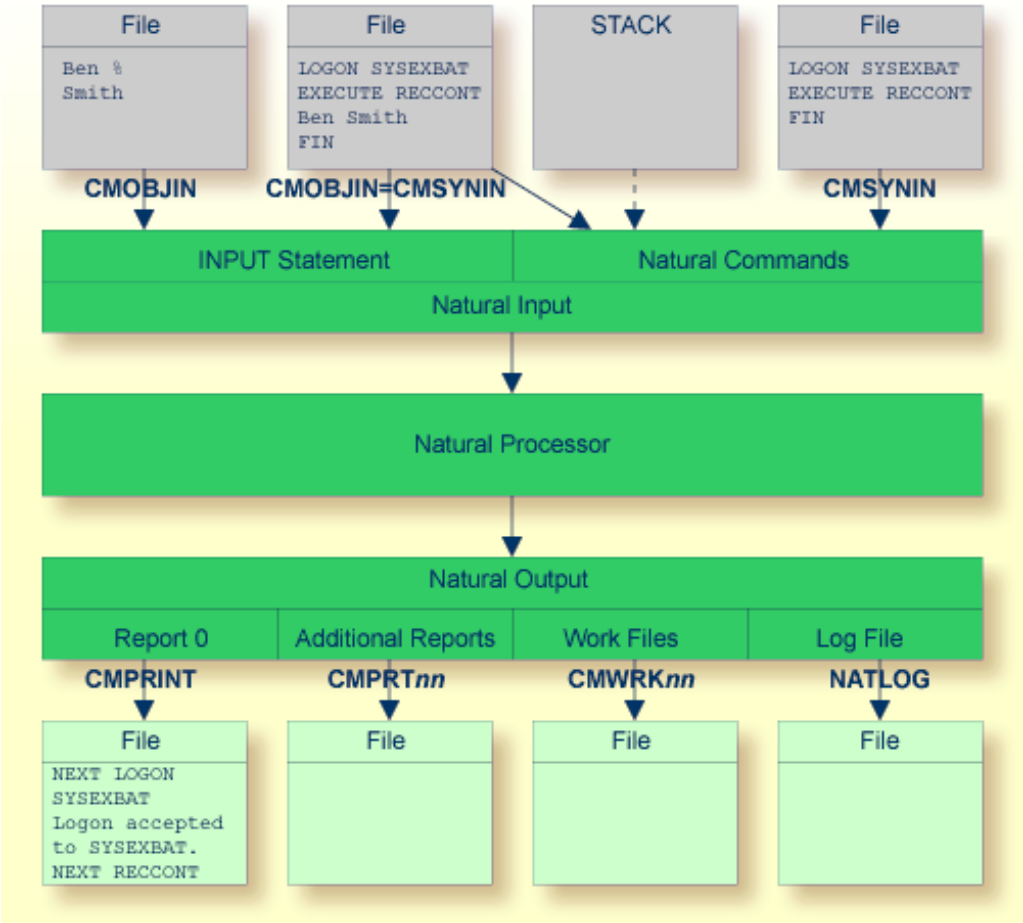
The `DISPLAY` statement in the program `RECCONT` writes the data to the batch output file *out.txt* which is created in your file system. The content of this file is shown below:

```
NEXT LOGON SYSEXBAT
Logon accepted to library SYSEXBAT.
NEXT EXECUTE RECCONT


DATA Ben %
DATA Smith
Page     1                                                  25.04.05  13:39:09

#FIRSTNAME #LASTNAME
---------- ----------

Ben        Smith
NEXT FIN
NAT9995 Natural session terminated normally.
```

`NATLOG=ALL`

When you invoke the sample session with the above call, a log file is created with contains all types of messages (which also includes the names of the batch input and outfile files). The log file is normally created in Natural's temporary directory which is defined in the local configuration file. See also the description of the `NATLOG` parameter.

The image below illustrates the different ways in which Natural reads input and writes output in batch mode.




As shown in the above graphic, you can proceed in one of the following ways:

- `CMOBJIN` **and** `CMSYNIN`
  Different files are used for batch input. One file contains the Natural commands and the other file contains the data:

```
natural BATCHMODE CMSYNIN=cmd.txt CMOBJIN=data.txt CMPRINT=out.txt
```

- `CMSYNIN`
  One file is used for batch input. It contains both the Natural commands and data:

```
natural BATCHMODE CMSYNIN=data.txt CMOBJIN=data.txt CMPRINT=out.txt
```

**Note:** Even though only one batch input file is used, both parameters `CMSYNIN` and `CMOBJIN` have to be specified. Both parameters must refer to the same file.

- CMOBJIN **and** STACK
  One file is used for batch input. It contains the data. The Natural commands are specified with the profile parameter STACK:

```
natural BATCHMODE CMOBJIN=data.txt STACK="(LOGON SYSEXBAT; RECCONT;FIN)"
```

## Batch Mode Detection

The system variable *DEVICE indicates whether Natural is running in batch mode or interactive mode.

| Mode | Description |
|---|---|
| Batch mode | *DEVICE contains the value BATCH. This value is set by the parameter BATCHMODE. |
| Interactive mode | *DEVICE contains a value other than BATCH. In most cases, it contains the value VIDEO. |

Example:

```
IF *DEVICE = "BATCH" THEN
   WRITE 'This is the background task'
ELSE
   WRITE 'This is the interactive session'
END-IF
```

## Batch Mode Restrictions

When Natural is running in batch mode, some features are not available or are disabled:

- Interactive input or output is not possible.
- Only data for an INPUT statement can be processed.
- The terminal database SAGtermcap is not supported. Therefore, the terminal capability TCS which is used for a different character set is not supported. To use a different character set, use environment variable NATTCHARSET instead.
- No colors and video attributes are written to the batch output file defined by CMPRINT.
- Filler characters are not displayed within an INPUT statement.
- Certain Natural system commands are not executable in batch mode, and are ignored. In the *System Commands* documentation, a corresponding note is provided for each system command to which this restriction applies.

## Batch Mode Simulation

In addition to the batch mode as described above, you can also simulate batch mode. However, it is recommended to use batch mode instead of batch mode simulation. Batch mode has the following advantages over batch mode simulation:

- Easy data input with support of keyword delimiter mode.
- Configurable and formatted output processing.
- Extended error handling.
- Faster startup and shutdown.
- Faster program execution.

If the input channel is redirected to a file, Natural does not read the input commands and data from the keyboard but from this file. You have to specify the data in exactly the same way as you would do on the terminal. For example, for two input fields you have to fill up the first field with trailing blanks to position to the second field. No keyword delimiter mode is supported. To use keyword delimiter mode, use batch mode instead of batch mode simulation.

If the output channel is redirected to a file, Natural writes any output that would appear on the screen to this file. Control sequences are also written to the file, which makes the file unreadable. To get a formatted output, use batch mode instead of batch mode simulation.

Use the dynamic parameter `BATCH` when starting Natural, to set the system variable `*DEVICE` to the value `BATCH`. This value can be checked within a Natural program.

**Example: Redirecting the Input Channel**

```
natural BATCH < input-file-name
```

Natural then receives all input operations from this input file (an example of this input file is provided below).

**Example: Redirecting the Input and Output Channel**

```
natural BATCH < input-file-name > output-file-name
```

If you want to keep Natural reports only and hide all other output (write output to the null device), set the profile parameter `MAINPR` to a valid printer number and assign an executable file to the corresponding logical printer (device) in the parameter file, then specify:

```
natural BATCH < input-file-name > /dev/null
```

Any Natural reports are written to the executable file, whereas any screen output is suppressed. An input file must be specified even if Natural does not expect any input at all. In this case, also the null device may be used.

### Sample Input File

```
dlist program *^M
fin^M
```

The input file for batch mode simulation must contain the same keystrokes that you would make in an interactive session.

The following keystrokes are used in the above sample input file:

| | |
|---|---|
| d | Opens the **Direct Command** window. |
| list program * | Executes the Natural system command which is used to list all programs. |
| ^M | Stands for the key combination CTRL+M (carriage return). Simulates the ENTER key. |
| fin | Executes the Natural system command which is used to terminate the Natural session. |
| ^M | Stands for the key combination CTRL+M (carriage return). Simulates the ENTER key. |

# 8 Support of Different Character Sets with NATCONV.INI

The settings in the configuration file *NATCONV.INI* apply to the A format. For the U format, the ICU library is used.

This chapter describes how Natural supports different character sets.

## Why Support Different Character Sets

The support of multiple languages with different character sets represents Natural's approach towards internationalization. It can help you when using:

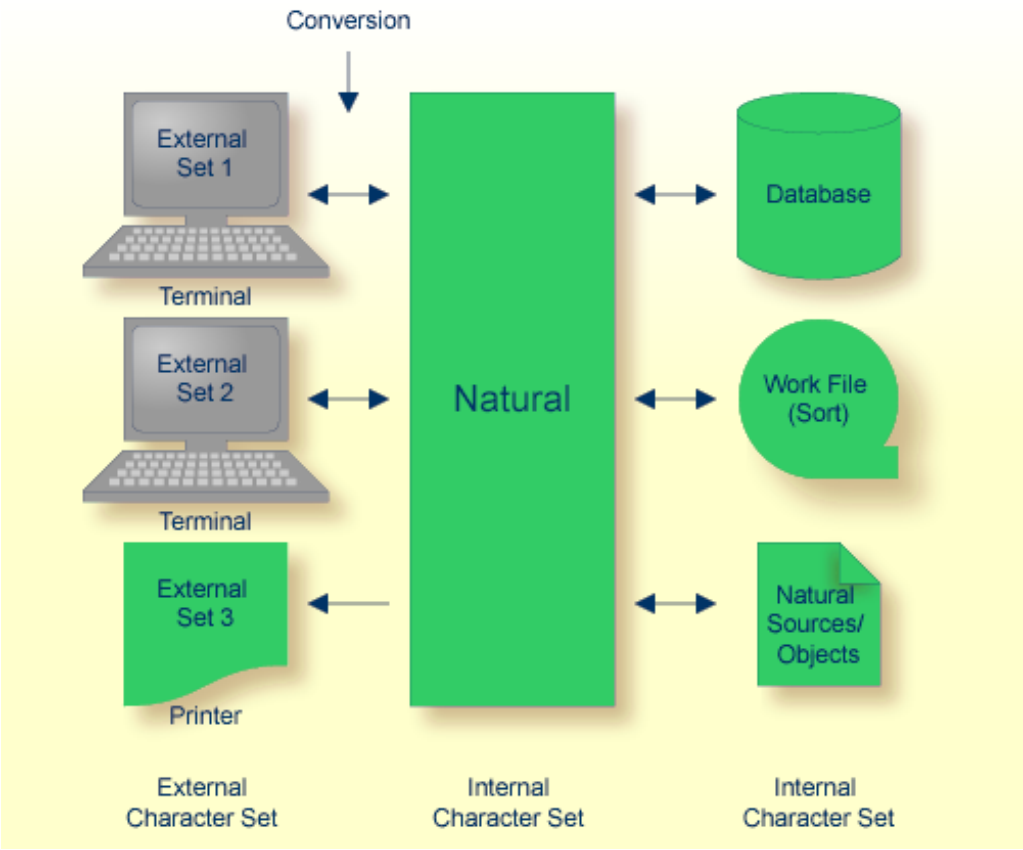
- terminals and printers with different character sets, all communicating with the same Natural environment;
- several Natural environments sharing one database and located on different platforms;
- upper-/lower-case translation of language-specific characters;
- language-specific characters in Natural identifiers, object names and library names;
- language-specific characters in an operand compared with a mask definition (see *MASK Option* in the *Programming Guide*).

## Character Sets that are Supported

Natural supports any single-byte character set that conforms to the ASCII character set in the lowest seven bits.

Natural distinguishes between an internal character and several external character sets; the internal character set is used by Natural itself.

As illustrated below, conversion between the internal and an external character set is performed after the input from a terminal and before the output to a terminal or printer. There is no conversion to an external character set available for work file I/Os, database I/Os and reading/writing of Natural objects.




### Internal Character Set

By default, Natural uses the internal character set ISO8859_1. If the default character set does not meet your requirements, you can choose either one of the predefined character sets provided by Natural or any other standard character set.

**Note:** Problems may occur if you run computers with different internal character sets sharing the same database, or if you try to exchange data or Natural objects between such computers.

### External Character Sets

You can define an external character set for any terminal and printer.

For a terminal, the name of its character set is defined by the TCS entry in the terminal database, for example:

```
:TCS = usascii:
```

You can also use the Linux environment variable `$NATTCHARSET` which overrides all TCS settings.

If neither a TCS entry nor the logical `NATTCHARSET` (which is set with the environment variable `$NATTCHARSET`) is defined, no conversion is performed during terminal I/O.

For a printer, the name of an external character set name can be defined in the printer profile. This is part of the global configuration file. See *Printer Profiles* in the *Overview of Configuration File Parameters* of the *Configuration Utility* documentation.

## How to Use Different Character Sets

All check, translation and classification tables used by Natural to support language-specific characters reside in the configuration file *NATCONV.INI*. By default, this file is located in Natural's *etc* directory.

You can modify *NATCONV.INI* to support local or application-specific character sets.

In a standard application, *NATCONV.INI* need not and should not be modified, because this could lead to serious inconsistencies, in particular if Natural objects and database data are already present.

Modifications are necessary if you want to do any of the following:

- use an internal character set other than the default one,
- use a terminal or printer whose character set is not supported by *NATCONV.INI*,
- allow or disallow the use of certain characters in identifiers,
- support local characters when evaluating the `MASK` option.

Any modifications of *NATCONV.INI* should be well considered and carefully performed, otherwise problems might occur that are difficult to locate.

*NATCONV.INI* is subdivided in sections and subsections. The following sections are defined:

| Section | Description |
|---|---|
| `CHARACTERSET-DEFINITION` | This section defines the name of the internal character set. The default is ISO8859_1.<br><br>If you choose a different character set, subsections for this character set must be contained in the sections described below. |

| Section | Description |
|---|---|
| CHARACTERSET-TRANSLATION | This section contains the tables required for the conversion between the internal character set and external character sets.<br><br>If you use, for example, a terminal with an entry in SAGtermcap of :TCS = ASCII_GERMAN: and if ISO8859_1 is used as internal character set, the following two subsections must be contained in this section:<br><br>■ [ISO8859_1->ASCII_GERMAN]<br>■ [ASCII_GERMAN->ISO8859_1] |
| CASE-TRANSLATION | This section contains the tables required for the conversion from upper-case to lower-case which is performed when one of the following is specified:<br><br>■ the terminal command %U,<br>■ the field attribute AD=T,<br>■ the statement EXAMINE TRANSLATE.<br><br>This conversion is done within the internal character set. If the internal character set is, for example, ISO8859_5, the following two subsections must be contained in this section:<br><br>■ [ISO8859_5->UPPER]<br>■ [ISO8859_5->LOWER] |
| IDENTIFIER-VALIDATION | This section contains the tables required for the validation of identifiers (that is, user-defined variables in source programs), object names and library names. It contains a subsection for each defined internal character set.<br><br>The special characters "#" (for non-database variables), "+" (for application-independent variables), "@" (for SQL and Adabas null or length indicators) and "&" (for dynamic source generation) can be redefined in this section. In addition, the set of valid first and subsequent characters for identifiers, object names and library names can be modified.<br><br>**Note:** When extending the set of valid characters for object names with values greater than x7f (decimal 127), the sorting sequence of the objects (for example, during a LIST * command) may not be in the numerical order. |
| CHARACTER-CLASSIFICATION | This section contains the tables required for the classification of characters, which, for example, are used when evaluating the MASK option. It contains a subsection for each defined internal character set. |

The section CHARACTERSET-DEFINITION and each subsection contain lines which describe how characters are to be converted and which characters are related with which attributes. These lines are represented as follows:

```
line          ::=  key = value
key           ::=  name_key | range_key
name_key      ::=  keyword{ CHARS }
keyword       ::=  INTERNAL-CHARACTERSET | NON-DB-VARI | DYNAMIC-SOURCE |
                     GLOBAL-VARI | FIRST-CHAR | SUBSEQUENT-CHAR |
                     LIB-FIRST-CHAR | LIB-SUBSEQUENT-CHAR | ALTERNATE-CARET
                     ISASCII | ISALPHA | ISALNUM | ISDIGIT | ISXDIGIT |
                     ISLOWER | ISUPPER | ISCNTRL | ISPRINT | ISPUNCT |
                     ISGRAPH | ISSPACE
range_key     ::=  hexnum | hexnum-hexnum
value         ::=  val {, val }
val           ::=  hexnum | hexnum-hexnum
hexnum        ::=  xhexdigithexdigit | xhexdigithexdigit
```

**Notes:**

1. If the `range_key` variable is specified on the left-hand side, the number of values specified on the right-hand side must correspond to the number of values specified in the key range, unless only one value is specified on the right-hand side, which is then assigned to each element of the key range.
2. When the `name_key` variable is specified on the left-hand side and the corresponding list of character codes does not fit in one line, it can be continued on the next line by specifying `name_key` = again. You must not start the lines with leading blanks or tabulators.

**Examples of Valid Lines**

| | |
|---|---|
| x00-x1f = x00 | All characters between `x00` and `x1f` are converted to `x00`. |
| x00-x7f = x00-x7f | All characters between `x00` and `x7f` are not converted. |
| x00-x08 = x00,x01-x07,x00 | The characters `x00` and `x08` are converted to `x00` and characters between `x01` and `x07` are not converted. |
| ISALPHA = x41-x5a,x61-x7a,xc0-xd6,xd8<br>ISALPHA = xd9-xf6,xf8-xff | The attribute `ISALPHA` is assigned to all characters specified in these two lines. |

**Examples of Invalid Lines**

| | |
|---|---|
| x41 = 'A' | All characters must be specified in hexadecimal format. |
| 0x00-0x1f = 0x00 | Hexadecimal values have to be specified in either of the following ways:<br><br>x*digitdigit*<br>X*digitdigit* |
| x00-x0f = x00,x01 | The number of specified values does not correspond to the number of elements in the key range. |

# 9 Natural Exit Codes

There are two types of Natural exit codes:

- **Startup errors**, where exit codes 0 and 1 indicate success and all other exit codes indicate errors.
- Errors generated by the TERMINATE statement, where exit codes 0 to 255 are possible.

## Natural Startup Errors

The following exit codes may occur when starting Natural.

| | |
|---|---|
| 2 | Terminal Control String (TCS) capability specified in SAGtermcap or Environment Variable NATTCHARSET. |
| 3 | Failed to initialize character conversion table. |
| 4 | Error in character conversion file *NATCONV.INI*. |
| 5 | Unable to read database assignments from global configuration file *NATCONF.CFG*. |
| 6 | Unable to find FNAT(*dbid*,*fnr*) or FUSER(*dbid*,*fnr*). Check your configuration files. |
| 7 | Cannot initialize LFILE table. |
| 10 | Obsolete. |
| 11 | Obsolete. |
| 12 | Unable to read specified parameter file. Please verify the parameter file. |
| 13 | Unable to read parameter file NATPARM. |
| 14 | Storage manager initialization failed. |
| 15 | End of input file (EOF) encountered while reading from STDIN. |
| 16 | Unable to open buffer pool; contact the Natural system administrator. |
| 17 | Unable to read buffer pool assignments from *NATURAL.INI* file. |
| 18 | Invalid FDIC assignment. |
| 19 | Invalid FNAT assignment. |
| 20 | Invalid FSEC assignment. |
| 21 | Invalid FUSER assignment. |
| 22 | Unable to load Natural login module. |
| 23 | Unable to allocate memory for local data. Reduce USIZE and/or SSIZE parameter. |
| 24 | Unable to load Natural display module. |
| 25,26 | Error loading shareable image or DLL. |
| 28 | Security violation during start of Natural. Natural terminates. |
| 31 | NAT0866 Your Natural nucleus is not a Natural Security nucleus. |
| 33 | Lock manager cannot create/initialize semaphores. |
| 34 | No library is accessible or present in specified FNAT/FUSER. Check system file assignments and file attributes of FNAT and FUSER (directories and files). |

| | |
|---|---|
| 35 | Internal wfc i/o terminal driver error. |
| 36 | Internal XVT error. |
| 37 | NNI Startup error. |
| 38 | Creation of runtime context failed. |
| 39 | Unable to find NATDIR and/or NATVERS environment variable. If you have set the NATDIR environment variable, please check that it does not contain invalid or whitespace characters! NATVERS should only contain the Natural version. The path must contain a valid drive ID. |
| 40 | Natural zmodem error. |
| 41 | Creation of TF table failed because there are entries with different database types from older parameter module. Check parameter module with Natural Configuration Utility. |
| 42 | Batch mode driver error. |
| 43 | Screen window size is too small. |
| 44 | Exit from SQL signal handler. |
| 46 | Unable to access FNAT library SYSLIB. Insufficient privilege or file protection violation. |
| 47 | Unable to read PARM_PATH entry from *NATURAL.INI* file or directory is not accessible. |
| 48 | Unable to read CONFIG_NAME entry from *NATURAL.INI* file or file is not accessible. |
| 49 | Unable to read NATTCAP entry from *NATURAL.INI* file or file is not accessible. |
| 50 | Unable to read NATCONV entry from *NATURAL.INI* file or file is not accessible. |
| 51 | Unable to process TMP_PATH entry from *NATURAL.INI* file. Path '*path*' not accessible. |
| 52 | Unable to read PROFILE_PATH entry from *NATURAL.INI* file or directory is not accessible. |
| 53 | Unable to open local configuration file *NATURAL.INI*. |
| 54 | Unable to read *NATCONF.CFG* for NATOSDEP. |
| 55 | Unable to read *NATURAL.INI* for NATEXTLIB. |
| 59 | Unrecognized option '*option*' specified. |
| 60 | Not enough memory to initalize internal tables. |
| 61 | Execution or compilation error occured. |
| 63 | Natural session with active repository already running. |
| 64 | Failed to open FNAT's *LIBDIR.SAG*. Check presence and access protection. |
| 65 | The FNAT assigned to this Natural session is out of date. |
| 72 | This is an evaluation copy of Natural ... It is valid until... |
| 73 | The test period of this evaluation copy of Natural ... has expired. It was valid until... |
| 77 | Invalid FDDM assignment. |
| 78 | The specified server session ... is not accessible. |
| 80 | Invalid combination of options encountered. |
| 81 | NDV server could not be terminated. Reason: . . . . |
| 82 | Error accessing file 'NDVSERVER.PRU'. |
| 83 | Error accessing file 'NDVSERVER.PRU'. |
| 85 | Natural runtime startup error during context initialization. |

| | |
|---|---|
| 86 | Invalid code page [ . . . ] specified. |
| 87 | Failure initializing signal handlers. |
| 88 | Conflicting buffer pool usage. |
| 90 | Invalid Client type [ . . . ] encountered. Please use ONE, NAT or ANY. |
| 91 | No connection to Natural Web I/O Interface. |
| 93 | SSL/TLS could not be activated. Reason: . . . . |
| 94 | Pre-loading of OpenSSL libraries failed. |
| 95 | RDC environment not found. |
| 96 | Failed to create RDC trace buffer. |
| 97 | Invalid RDC trace buffer length. |
| 98 | Failed to create RDC resource file. |
| 99 | Failure on writing RDC resource file. |
| 100 | Generic RDC error. |
| 101 | Failed to create RDC consolidation buffer. |
| 102 | Invalid RDC consolidation buffer length. |
| 103 | Cannot create RES subdirectory for storage of RDC data. |
| 104 | RDC resource full name (including path) too long. |
| 105 | Value of dynamic parameter ITERM must be ON or OFF. |
| 106 | Terminate on error during initialization. |
| 107 | Profiling with sampling not allowed if code coverage also active. |
| 108 | Profiling with event trace not allowed if code coverage also active. |
| 110 | Invalid ICU version (custom BS2000 code page support missing). |
| 120 | Failed to attach to RDC trace buffer. |
| 121 | Failed to write to RDC trace buffer. |
| 122 | License check failed. |
| 123 | WEBIO=ON is allowed for server sessions only. |

# 10 Setting Up the Entire System Server Interface

The Entire System Server Interface is required if the product Entire System Server is to be used. The Entire System Server Interface is part of Natural and no extra installation is needed.

Additionally, Natural provides the libraries SYSNPE and SYSNPR.

SYSNPE is the Entire System Server online tutorial which is provided as a starting help for Entire System Server users. For more information about Entire System Server, see the Entire System Server documentation.

The library SYSNPR contains the program CHANGEDB which is used to change the database ID of the Entire System Server DDMs.

## Prerequisites

The Entire System Server Interface provides access to Entire System Server on z/OS via Entire Net-Work. For full support of the Entire System Server Interface, Entire Net-Work Version 5.8.1 or above is required on z/OS platforms.

## Activation

The Entire System Server Interface is not active if you use the standard Natural configuration settings. The value of the Entire System Server Interface database (Natural profile parameter ESXDB) is set to 0 by default. To use the Entire System Server Interface you have to set the value of the parameter ESXDB to 148 using the Configuration Utility.

In the Configuration Utility, the parameter ESXDB is assigned in the parameter group **Product Configuration** of a parameter file.

**Tip:** Locate this parameter by searching for "ESXDB". See *Finding a Parameter* in the *Configuration Utility* documentation for further information.

ESXDB specifies the database ID used for the DDMs of Entire System Server. This DBID does not specify the target DBID of Entire System Server requests but tells Natural which DBID is used for the cataloged Entire System Server DDMs. The effective Entire System Server target DBID will be specified with the NODE field which is part of all Entire System Server DDMs.

**Important:** Change the value of ESXDB to 148 to run Natural with Entire System Server Interface support. All Entire System Server DDMs are cataloged with DBID 148.

After starting Natural again, you may access Entire System Server nodes running on z/OS via Entire Net-Work.

The customization of Entire System Server Interface supports the modification of the Entire System Server DDMs only.

## Changing the Database ID for the Entire System Server DDMs

The library SYSNPR contains the program CHANGEDB which is used to modify the database ID of all Entire System Server DDMs. You will find all Entire System Server DDMs in the library SYSNPE. The database ID entered as a new DBID value in the program CHANGEDB must also be specified as the value of the Entire System Server Interface database parameter (ESXDB) in the Configuration Utility.

# 11 Tuning SQL Database Access

By default, the Natural SQL driver manages a table with the 16 most recently used Natural statements. All statements in this table are marked as prepared, which indicates that the statement can be executed immediately without being compiled by the database system.

To ensure maximum performance, the dynamic parameters SQLRELCMD and SQLMAXSTMT are provided. These parameters configure the handling of the SQL driver's statement table. Note that these parameters are not profile parameters.

## SQLRELCMD

This parameter determines when commands are to be released from the SQL statement table.

Possible values:

- ENDGP (default): if a generated program terminates, all statements from this program that are in the statement table are removed from the table.
- NEVER: No statement will be deleted from the table.

## SQLMAXSTMT

This parameter determines the size of the statement table.

Possible values are 1 to 64 (default: 16).

If you set the SQLMAXSTMT parameter, keep the following in mind:

- Resource consumption may be higher if you are keeping more prepared statements in the table.
- If the size of the statement table exceeds the limit of dynamic SQL statements in the target database, the application will receive SQL errors.
- It depends on the database whether there is a real benefit from the SQLMAXSTMT optimization.
- In general, performance in batch-type applications will be improved if the number of PREPARE statements is minimized, while performance in online applications will probably be worse because of the increased resource consumption of the target database.

## Example

To set the above parameters dynamically, enter them when starting Natural:

```
natural sqlrelcmd=never sqlmaxstmt=40
```

Natural will then start with a statement table size of 40 and the statement table will only be cleared when Natural is terminated.

# 12 User Exit for Computation of Sort Keys - NATUSKnn

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 the C programming language using the 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.

**Note:** A conversion table is not supplied.

NATUSK*nn* is called using the CALL interface. The parameters of the C function have the following values:

| Parameter | Contents |
|---|---|
| 1 | The number of arguments. |
| 2 | The array of pointers to the operands. |
| 3 | The array of field information for each operand. |

If you use the Natural system function #OP1=SORTKEY(#OP2), the source operand is in the arrays at index 0 and the target operand (#OP1) is in the arrays at index 1.

A sample user exit, *natusk01.c*, is provided in source form: it applies to English and converts all English lower-case letters in the character string to upper-case letters. The sample is to be found in <*install-dir*>/*natural/samples/sysexuex*, where you can also find the other user exits.

The source code of the example contains all comments which are needed to write a specific user exit for SORTKEY.

For linkage and loading conventions, refer to the CALL statement.

# 13 Abnormal End (Abend) Handling

The signal SIGTERM is caught. The signal handler for SIGTERM releases all currently used resources and terminates Natural smoothly.

Natural's default signal handlers for SIGBUS, SIGSEGV and SIGILL are only installed if the command gcore is available and if the user running Natural has execution rights for this command. If one of these signal handlers gets control, the handlers that were valid at startup time are restored, gcore is executed and an attempt is made to behave as if the signal SIGTERM was caught.