Natural

Operations

Version 9.3.4

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.3.4 and all subsequent releases.

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

Copyright © 1992-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

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

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

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

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

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

Document ID: NATWIN-NNATOPERATIONS-934-20261013

Table of Contents

Preface vii
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
I 5
2 About this Documentation 7
3 Using the Windows Firewall with Natural 9
4 Profile Parameter Usage 11
Parameter Hierarchy 12
Static Assignment of Parameter Values 13
Dynamic Assignment of Parameter Values 13
Runtime Assignment of Parameter Values 14
5 System Files 17
System File Structure 18
System Files FNAT and FUSER 19
System File FDDM 21
Important Information and Warnings 24
The File FILEDIR.SAG 24
Portable Natural System Files 25
Using NFS to Store Natural Libraries 26
6 Work Files 27
Defining Work Files 28
Work File Formats 31
Special Considerations for Work Files with the Extension NCD 34
7 Natural Buffer Pool 37
About the Natural Buffer Pool 38
Setting up a Buffer Pool 50
Using the Natural Buffer Pool Service 50
Using the Utility NATBPSRV for Creating the Buffer Pool 53
Monitoring the Buffer Pool 53
Trouble Shooting 54
8 Using the GUI Version of the Buffer Pool Monitor 57
Starting and Terminating the Buffer Pool Monitor 58
Elements of the Natural Buffer Pool Monitor Window 59
Disconnecting and Connecting a Buffer Pool 62
Shutting Down a Buffer Pool Server 63
Starting a Buffer Pool Server 63
Changing the Properties of the Buffer Pool Monitor 64
Global Information 65
Buffer Pool Content 67
Graphic Analyzer 70
Reports 75

9 Using the Command Line Version of the Buffer Pool Monitor (NATBPMON) ... 81
Invoking the NATBPMON Utility ... 82
NATBPMON Commands ... 83
Displaying the Objects in the Buffer Pool ... 85
Specifying a Pattern ... 86
Displaying the Buffer Pool Settings ... 87
Statistical Information About the Buffer Pool ... 88
10 Natural in Batch Mode ... 93
About Batch Mode ... 94
Starting a Natural Session in Batch Mode ... 94
Terminating a Natural Session in Batch Mode ... 95
Using Natural in Batch Mode ... 95
Sample Session for Batch Mode ... 97
Batch Mode Detection ... 100
Batch Mode Restrictions ... 100
Hints for Using Natural Maps and Dialogs in Batch Mode ... 101
11 Output Window ... 103
About the Output Window ... 104
Working in the Output Window ... 104
Changing the Output Window Profile ... 105
Using Your Own Icon for the Output Window ... 106
12 Natural Runtime ... 109
Unsupported Functions by Natural Runtime ... 110
Porting Procedure Overview ... 111
Step 1 - Packaging the Application on the Development Workstation ... 111
Step 2 - Installing Natural Runtime ... 115
Step 3 - Installing the Application on the Runtime Workstation ... 116
Step 4 - Starting the Application on the Runtime Workstation ... 118
Using the Natural Runtime Startup Service ... 119
13 Support of Different Character Sets with NATCONV.INI ... 123
Why Support Different Character Sets ... 124
How to Use Different Character Sets ... 124
14 Natural Exit Codes ... 127
Natural Startup Errors ... 128
15 Setting Up the Entire System Server Interface ... 131
Prerequisites ... 132
Activation ... 132
Changing the Database ID for the Entire System Server DDMs ... 133
II Administrating NaturalX Applications ... 135
16 NaturalX Servers ... 137
COM Classes and Servers ... 138
NaturalX Classes and Servers ... 138
NaturalX Servers and Natural Sessions under Windows ... 138
The Role of the Server ID ... 139

Organizing Server IDs ... 140
17 Activation Policies ... 141
Activation Policies on Windows Platforms ... 142
Setting Activation Policies ... 142
When to Use Which Activation Policy ... 143
18 Registration ... 147
Registration with Natural ... 148
Automatic Registration ... 148
Manual Registration ... 149
Registration Files and Type Library ... 151
Client Registration ... 152
Registration Hints ... 153
19 Type Information ... 155
About Type Information ... 156
NaturalX and Type Information ... 156
Using Type Information ... 156
20 Configuration Overview ... 161
Server Configuration - General Settings ... 162
Server Configuration - Application-Specific Settings ... 163
Client Configuration - General Settings ... 163
Client Configuration - Application-Specific Settings ... 164
21 Security with NaturalX ... 165
About NaturalX Security ... 166
Activation Security ... 166
Call Security ... 167
22 DCOM Configuration on Windows ... 169
Configuring NaturalX Servers ... 170
Configuring NaturalX Clients ... 180
23 NaturalX System Registry Entries ... 185
Registry Entries for Servers ... 186
Registry Entries for Clients ... 187
24 Using Statements and Commands in a NaturalX Server Environment ... 189
Natural Statements ... 190
Natural System Commands ... 191

Preface

This documentation is organized under the following headings:

About this Documentation	General information.
Using the Windows Firewall with Natural	How to run Natural in an environment protected by the Windows firewall.
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 GUI Version of the Buffer Pool Monitor	How to connect and disconnect to a buffer pool, and how to shut down and start a buffer pool server. A description of the information that can be displayed using the Buffer Pool Monitor.
Using the Command Line Version of 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.
Output Window	How to use the output window, change the output window profile and use your own icon for the output window.
Natural Runtime	How to port an application from a development workstation to a runtime workstation. How to use a service for starting Natural Runtime processes.
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.
Administrating NaturalX Applications	How to distribute applications consisting of NaturalX classes across several processes and machines using DCOM.

1 About this Documentation

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

Document Conventions

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

Online Information and Support

Product Documentation

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

Product Training

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

Tech Community

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

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

Product Support

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

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

Data Protection

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

I

- 2 About this Documentation 7
- 3 Using the Windows Firewall with Natural 9
- 4 Profile Parameter Usage 11
- 5 System Files 17
- 6 Work Files 27
- 7 Natural Buffer Pool 37
- 8 Using the GUI Version of the Buffer Pool Monitor 57
- 9 Using the Command Line Version of the Buffer Pool Monitor (NATBPMON) 81
- 10 Natural in Batch Mode 93
- 11 Output Window 103
- 12 Natural Runtime 109
- 13 Support of Different Character Sets with NATCONV.INI 123
- 14 Natural Exit Codes 127
- 15 Setting Up the Entire System Server Interface 131

2 About this Documentation

This documentation contains information for operating Natural in a Windows environment.

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.

When installing Natural fixes with the Update Manager, certain restrictions and requirements apply. Refer to *Special Considerations When Installing Fixes with the Update Manager* for further details.

Note: We would like to remind our customers who have purchased the Natural Runtime version that the Natural development tools are not included in the Natural Runtime version. In addition, not all Natural system commands are supported in the Natural Runtime version.

3 Using the Windows Firewall with Natural

In Windows, the firewall is switched on by default. To work with Natural, you have to allow Natural to communicate through the firewall. If you do not allow this, it is not possible to start Natural.

You can easily allow the communication through the firewall when you start a Natural component for the first time. In this case, a firewall warning will occur. You just have to accept this warning in order to allow this Natural component.

For detailed information on configuring the Windows firewall, see the Microsoft documentation.

Caution: It is not recommended to turn off the firewall.

4 Profile Parameter Usage

- Parameter Hierarchy 12
- Static Assignment of Parameter Values 13
- Dynamic Assignment of Parameter Values 13
- Runtime Assignment of Parameter Values 14

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

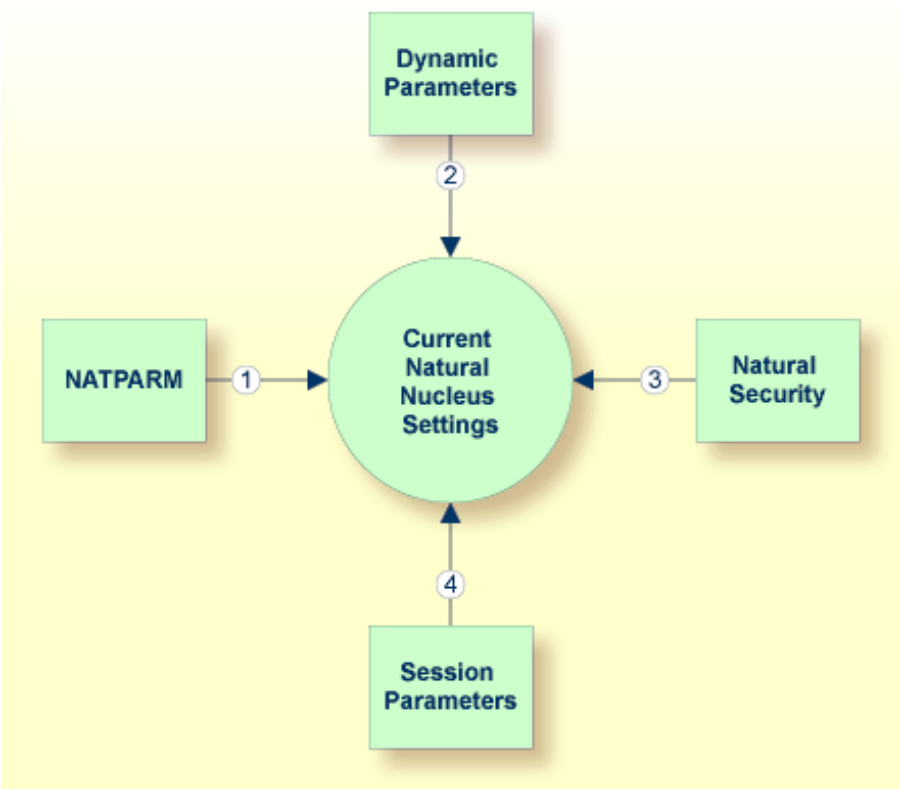
The values for the Natural parameters are taken from different sources. The priority of the parameters is as follows:

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
  ```

  Or:

  When you start Natural using a shortcut, specify the dynamic parameters after *natural.exe*.

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.

When using Natural Studio, you can set the session parameters in a window (this corresponds to issuing the system command `GLOBALS` without parameters). See *Using Session Parameters* in the documentation *Using Natural Studio*.

Session parameters can also be 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*.

5 System Files

- System File Structure 18
- System Files FNAT and FUSER 19
- System File FDDM 21
- Important Information and Warnings 24
- The File FILEDIR.SAG 24
- Portable Natural System Files 25
- Using NFS to Store Natural Libraries 26

Natural for Windows 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 Windows stores the objects in specific directories on the disk. Thus, a database such as Adabas is not required to run Natural for Windows.

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 Studio*.
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: S – for source files G – for generated programs 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
3	Dialog
4	Class
5	Command processor
7	Function
8	Adapter

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, and the FDDM system file is displayed as an inactive environment.

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. Select 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 Select the required path for the parameter FDDM from the drop-down list box.

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, or to copy and move (or drag-and-drop) the DDMs with Natural Studio as described below. 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 Studio using the modified parameter file (that is, the parameter file in which path for the parameter FDDM has been defined).

3 From the **Tools** menu, choose **Development Tools > Object Handler** to start the Object Handler.

4 From the **Options** menu of the Object Handler, choose **Settings**.

5 In the resulting dialog box, select the option button **Additional Options** and choose the **Set** button.

6 In the resulting dialog box, select the **Special** page.

7 Deactivate the check box **Use FDDM file for DDMs**.

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

8 Unload the DDMs stored in Natural libraries (either with the wizard or in advanced-user mode).

9 Activate the check box **Use FDDM file for DDMs** (see the above steps).

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 check box **Use FDDM file for DDMs**.

10 Load the DDMs into the `FDDM` system file (either with the wizard or in advanced-user mode).

To migrate DDMs to the system file `FDDM` using the copy or move function of Natural Studio

1 Start Natural with the **dynamic** parameter `FDDM=0,0` as shown below:

```
natural FDDM=0,0
```

This activates your old environment containing the DDM to migrated. If you do not override the new `FDDM` specification in your modified parameter file, you cannot access these DDMs.

2 Copy or move all required DDMs from the Natural libraries into the library `SYSTEM` in your designated `FDDM` file. This file is displayed in the inactive environment of Natural Studio.

3 Terminate Natural.

The next time you start Natural without the above-mentioned dynamic parameter, the `FDDM` system file will be used.

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 From the **Tools** menu, choose **System Information > System Files**. See also *System Files* in the documentation *Using Natural Studio*.

The **SYSPROF** dialog box appears.

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.

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 Windows 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. Information on how to import other objects can be found in the section *Importing Objects* in the documentation *Using Natural Studio*.

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

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.

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.

6 Work Files

- Defining Work Files 28
- Work File Formats 31
- Special Considerations for Work Files with the Extension NCD 34

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`.

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
- Defining Work File Names with Environment Variables
- Defining Work File Names with an Application Programming Interface

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
- Delimiters of Environment Variables
- Dollar Sign ($) in the File Name

About Defining Work File Names with Environment Variables

Work files can also be defined by using Windows 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=D:\natural
set myfile=sub\test
```

this will expand into the following file name:

```
D:\natural\sub\test
```

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	% / . \

Note: The end-of-string mark is by default a right-hand delimiter, i.e. `%TEMP` is recognized as an environment variable named `TEMP`.

Although "%" is the only valid left-hand delimiter for environment variables in Windows, Natural for Windows allows "%" and "$" as left-hand delimiters in order to preserve upward compatibility with previous versions. This setting allows Linux-like work file name assignments in a Windows session. `$TEMP` is recognized in Natural for Linux as well as in Natural for Windows as the environment variable `TEMP`.

Example:

The following lines of Natural code are interpreted as being the same:

```
DEFINE WORK FILE 1 '$TEMP\myfile.dat'
```

and

```
DEFINE WORK FILE 1 '%TEMP%\myfile.dat'
```

`TEMP` is recognized as an environment variable. The string `$TEMP` (or `%TEMP%`) is replaced at runtime by the contents of the environment variable `TEMP`.

Dollar Sign ($) in the File Name

A dollar sign ($) in a file name has two meanings:

- If the dollar sign appears on the left or in the middle of a string embedded in delimiters, it will be interpreted as the left-hand delimiter of the environment variable being used. All characters following the left-hand delimiter up to the right-hand delimiter or EOS are considered to be the name of an environment variable.
- If the dollar sign is the last character of a string, it is not considered to be a delimiter character. It is a part of the string scanned.

Example:

The following line of Natural code does not result in an error:

```
DEFINE WORK FILE 1 '\\MYPC\C$\myfile.dat'
```

`\\MYPC\C$` is considered to be a default share. `C$` is a valid directory.

However, the following line of Natural code may result in an error, depending on whether A has been defined or not:

```
DEFINE WORK FILE 1 '\\MYPC\C$A\myfile.dat'
```

A is interpreted as an environment variable since it is preceded by a dollar sign. If A has not been defined, an error will occur. If A has been defined, an error does not occur.

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 described below.

- 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 carriage return and line feed (CR/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 type: Entire Connection.

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.

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. 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 $\pm 1.79 * 10^{308}$

7 Natural Buffer Pool

- About the Natural Buffer Pool 38
- Setting up a Buffer Pool 50
- Using the Natural Buffer Pool Service 50
- Using the Utility NATBPSRV for Creating the Buffer Pool 53
- Monitoring the Buffer Pool 53
- Trouble Shooting 54

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
- Resource Handling in the Buffer Pool
- Multiple Buffer Pools
- Storing Objects in the Buffer Pool
- Fast Locate
- Read-Only Buffer Pool
- Buffer Pool with Enhanced Performance
- Restrictions of the Natural Buffer Pool

Objects in the Buffer Pool

Objects in the buffer pool can be any executable objects such as programs and dialogs. 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).

Resource Handling in the Buffer Pool

Resources are loaded into the buffer pool if they reside in a library of a Natural system file (for example, FUSER) and if their names do not exceed 32 characters (including the file extension).

Each resource that resides in the directory which is assigned to the environment variable `NATGUI_BMP` or whose name is longer than 32 characters, is loaded directly into the Natural process every time it is accessed (that is: the resource is not loaded into the buffer pool).

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.

For each buffer pool, synchronization can be enabled in the Configuration Utility (see also ***Setting up a Buffer Pool*** below). All buffer pools which contain objects from the same system file and for which synchronization has been enabled are then synchronized automatically.

Important: If the system file resides on a shared drive, synchronization only works if the file system on the server is NTFS.

The following applies when synchronization has been enabled: If an object that is loaded to more than one buffer pool is modified by one Natural process, it is first marked as invalid. When the object is no longer used by any process, it is deleted from the buffer pool. The next time this object is requested by a process, it will be loaded into the buffer pool again.

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 Table
- Fast Locate Table with BPSFI=ON
- Performance with BPSFI=ON
- Fast Locate Table with BPSFI=OFF
- Performance with BPSFI=OFF
- Performance in a Multi-User Environment
- Maintaining the Fast Locate Table

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:

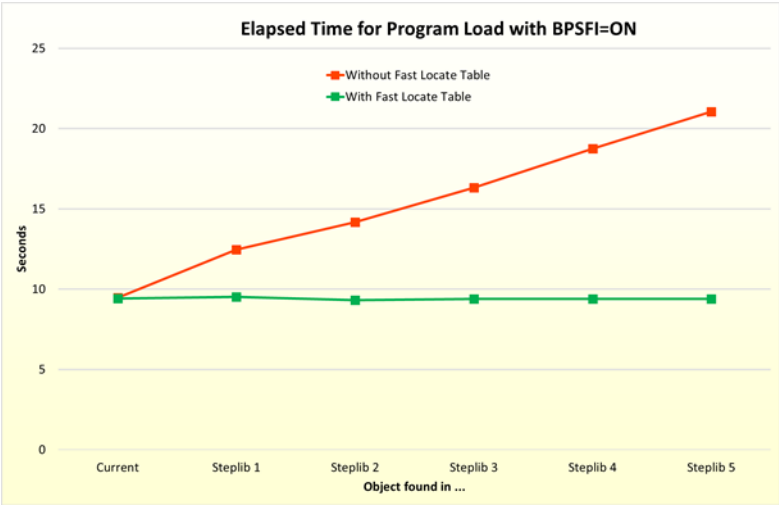
- The number of locate attempts should decrease significantly (compared with a deactivated fast locate table).
- The number of fast locate attempts should be close to the number of locate attempts.

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.

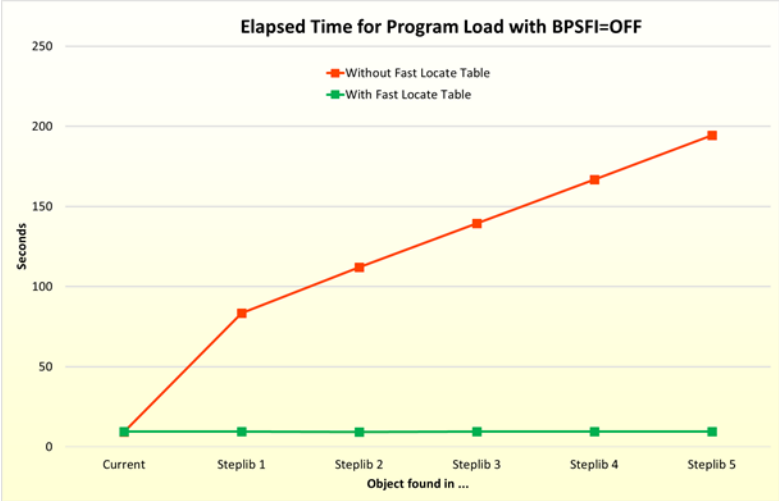
If the application is still running (not back on Program Level 0 in between) and the object is accessed again, the new version of the object will not be used.

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.



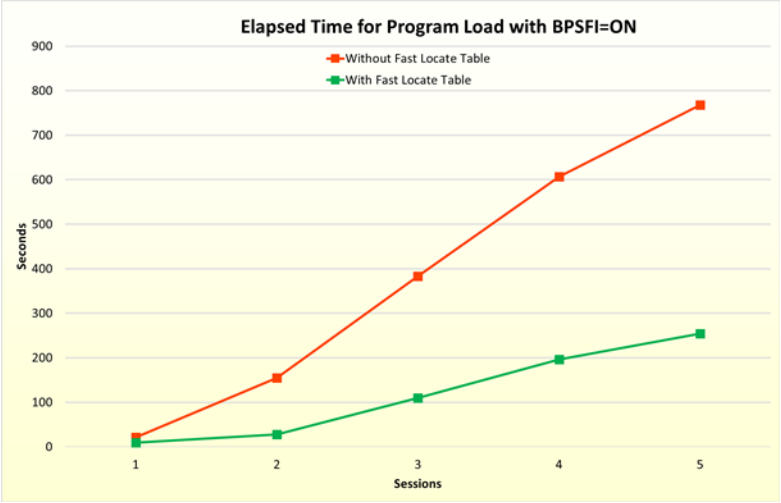
The diagram above shows that there is no performance improvement if the object is found in the current library. The more steplibs there are involved in object search operations, the higher is the performance improvement. Since the search operation on the system file is considerably slower than the search 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. 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. 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.

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.

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 Windows Task Manager.

  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.

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

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 Natural Buffer Pool Service

Natural uses a Windows service, the Natural Buffer Pool Service, to start the Buffer Pool Server when the PC is booted.

Natural is installed with the default buffer pool `NATBP`. `NATBP` is also used as the default buffer pool name at Natural startup (a different buffer pool can be defined using the profile parameter `BPID`).

You can modify the service configuration to meet your requirements. This is explained in the topics below.

- Buffer Pool Service Commands
- Starting Natural with Your Own Buffer Pool

Buffer Pool Service Commands

The file *natbpsvc.exe*, which is stored in the Natural *bin* directory, is used to execute the service commands.

The following service commands can be specified in the **Command Prompt** window of Windows:

Command	Description	
`NATBPSVC INSTALL` *`mode`*	Installs the buffer pool service. *`mode`* can be one of the following:	
	manual	Default. The service is installed and must be started manually (either with the `START` command or by starting the Natural Pool Service in Windows).
	automatic	The service is installed and is automatically started when the PC is booted.
`NATBPSVC CREATE` *`buffer-pool-name`*	Creates a new buffer pool to be started by the service. The service checks whether the buffer pool with the specified name is defined in the Natural parameter file.	

Command	Description
NATBPSVC START	Starts the service (if not yet active) and all created buffer pools (see the CREATE command) for which the start parameter has been set to "yes" (see the SET command).
NATBPSVC START *buffer-pool-name*	Starts the specified buffer pool. If the service has not been started (either automatically at boot time or manually by the user) an error message appears.
NATBPSVC SET *buffer-pool-name* start=*mode*	Defines whether the specified buffer pool is to be started when the service is started. *mode* can be one of the following: yes — The buffer pool is started. no — Default. The buffer pool is not started.
NATBPSVC STOP	Stops the service and all previously started buffer pools.
NATBPSVC STOP *buffer-pool-name*	Stops the specified buffer pool.
NATBPSVC SHOW	Displays the configuration parameters for all buffer pools that are defined for the service, that is: whether these buffer pools are to be started when the service is started.
NATBPSVC SHOW *buffer-pool-name*	Displays the configuration parameters for the specified buffer pool, that is: whether this buffer pool is to be started when the service is started.
NATBPSVC STATUS	Displays the status of all buffer pools that are defined for the service, that is: whether these buffer pools are active or not active.
NATBPSVC STATUS *buffer-pool-name*	Displays the status of the specified buffer pool, that is: whether this buffer pool is active or not active.
NATBPSVC DELETE *buffer-pool-name*	Deletes the specified buffer pool from the service. **Caution:** Do not delete the default buffer pool NATBP, as it is possible that Natural may not function properly anymore.
NATBPSVC REMOVE	Removes the service from the system.

Starting Natural with Your Own Buffer Pool

This example explains how to create a new buffer pool with the name MYBP and how to start Natural with your new buffer pool.

To start Natural with your own buffer pool

1 Use the Configuration Utility to define an additional buffer pool with the name MYBP in the local configuration file. See *Buffer Pool Assignments* in the *Configuration Utility* documentation.

Note: For this example, you can use the same values as defined for the default buffer pool NATBP.

2 Invoke the **Command Prompt** window of Windows.

3 Go to the Natural *bin* directory which contains the file *natbpsvc.exe*.

4 Enter the following command to create a buffer pool with the name `MYBP`:

```
NATBPSVC CREATE MYBP
```

The following information is shown:

```
%NATBPSVC-I: Natural n.n Buffer Pool Service
%NATBPSVC-I: New buffer pool 'MYBP' created
%NATBPSVC-I: Natural n.n Buffer Pool Service
```

Important: When the buffer pool with the specified name has not yet been defined in the local configuration file, a corresponding message is shown instead. Make sure to define the buffer pool in the local configuration file before you proceed with the steps below.

5 Enter the following command to define that your buffer pool is to be started when the service is started:

```
NATBPSVC SET MYBP start=yes
```

The following information is shown:

```
%NATBPSVC-I: Natural n.n Buffer Pool Service
%NATBPSVC-I: Configuration successfully set
%NATBPSVC-I: Natural n.n Buffer Pool Service
```

6 Enter the following command to start your buffer pool now (without having to restart the service):

```
NATBPSVC START MYBP
```

The following information is shown:

```
%NATBPSVC-I: Natural n.n Buffer Pool Service
%NATBPSVC-I: Send request to Natural n.n Buffer Pool Service
%NATBPSVC-I: Buffer pool 'MYBP' started
%NATBPSVC-I: Natural n.n Buffer Pool Service
```

7 Enter the following command to display the status of all buffer pools that are currently defined for the service:

```
NATBPSVC STATUS
```

The following information is shown:

```
%NATBPSVC-I: Natural n.n Buffer Pool Service
%NATBPSVC-I: Send request to Natural n.n Buffer Pool Service
%NATBPSVC-I:    MYBP is active
                NATBP is active
%NATBPSVC-I: Natural n.n Buffer Pool Service
```

8 Start Natural with the **dynamic** parameter `BPID` as shown below:

```
natural BPID=MYBP
```

Using the Utility NATBPSRV for Creating the Buffer Pool

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

The buffer pool server is automatically started by the **Natural Buffer Pool Service**.

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***).

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
```

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. To ensure that modified objects are also removed from other buffer pools on the same computer to which the object is currently loaded, you can enable the buffer pool synchronization in the Configuration Utility.

Natural provides two versions of the Buffer Pool Monitor: a graphical user interface and the `NATBPMON` utility which is a command line version.

For detailed information for how to use the different versions of the Buffer Pool Monitor, see ***Using the GUI Version of the Buffer Pool Monitor*** and ***Using the Command Line Version of 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.

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 cannot be started.

Example 1

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 `NATBPMON` utility, and the buffer pool is not active.

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

```
Natural Startup Error 16: Unable to open buffer pool.
Buffer pool error: "unexpected system call error occurred" (20)
Buffer pool could not attach to the global shared memory.
```

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

```
Cannot get shared memory
Buffer pool error: "unexpected system call error occurred" (20)
Buffer pool could not attach to the global shared memory.
```

Solution

Start the buffer pool service as described in ***Using the Natural Buffer Pool Service***.

Example 2

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 `NATBPMON` utility, and the buffer pool has been started with a different internal version.

- 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.

8 Using the GUI Version of the Buffer Pool Monitor

- Starting and Terminating the Buffer Pool Monitor 58
- Elements of the Natural Buffer Pool Monitor Window 59
- Disconnecting and Connecting a Buffer Pool 62
- Shutting Down a Buffer Pool Server 63
- Starting a Buffer Pool Server 63
- Changing the Properties of the Buffer Pool Monitor 64
- Global Information 65
- Buffer Pool Content 67
- Graphic Analyzer 70
- Reports 75

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.

Starting and Terminating the Buffer Pool Monitor

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

If the maximum user limit is reached, the user accesses the utility as an emergency user.

A Natural folder automatically appears in the **All Programs** folder of the **Start** menu after Natural has been installed. It contains the shortcuts for Natural, including the Buffer Pool Monitor.

To start the Buffer Pool Monitor

- From the Windows **Start** menu, navigate to **Administration > Natural Buffer Pool Monitor** *n.n*.

  The **Natural Buffer Pool Monitor** window appears.

  Note: The buffer pool can also be started using the executable file *natbpmong.exe* which is stored in the Natural *Bin* directory.

To terminate the Buffer Pool Monitor

- From the **Monitor** menu, choose **Exit**.

  Or:

  Press ALT+F4.

  Or:

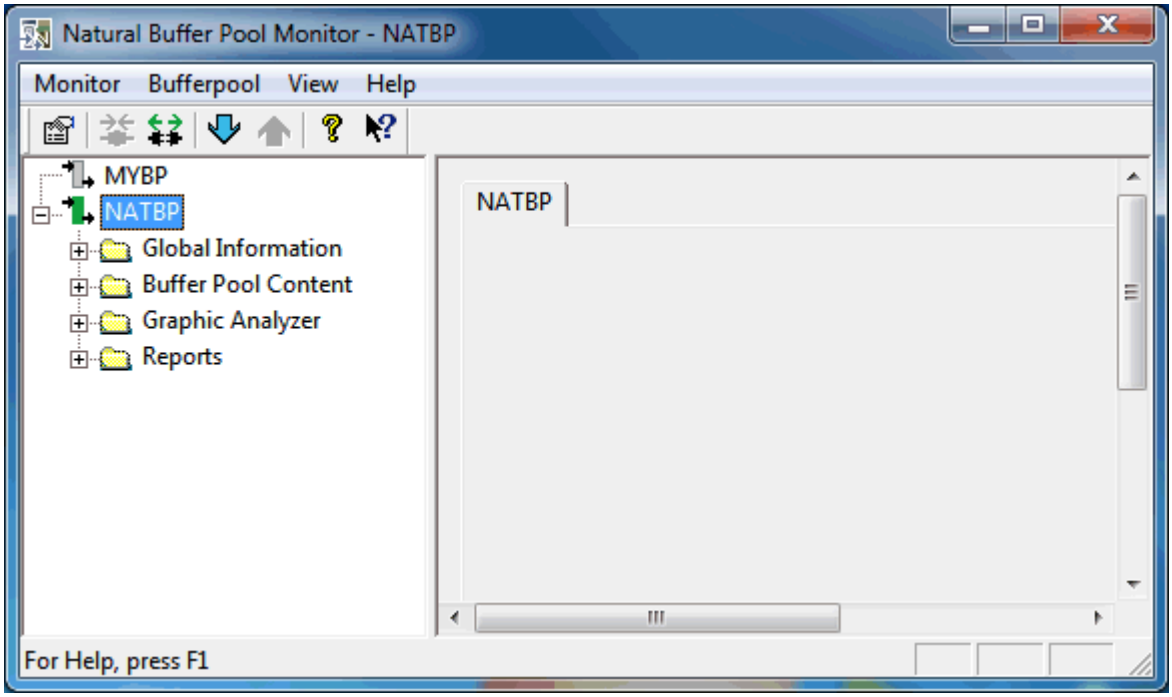
  From the Control menu, choose **Close**.

  Or:

  Choose the corresponding standard button at the right of the title bar.

Elements of the Natural Buffer Pool Monitor Window

When you start the Buffer Pool Monitor, it automatically tries to connect to Natural's default buffer pool `NATBP`. The name of the buffer pool which is currently used is shown in the title bar. It is also selected in the tree.



The following topics are covered below:

- Menu Bar
- Toolbar
- Tree
- Status Bar

Menu Bar

The following menus are available:

Menu	Using the commands in this menu, you can ...
Monitor	Change the properties or exit the Buffer Pool Monitor.
Bufferpool	Disconnect and connect a buffer pool. Shut down and start the buffer pool server.
View	Show or hide the various elements of the **Natural Buffer Pool Monitor** window.
Help	Invoke the online documentation and display information about the Buffer Pool Monitor.

Toolbar

You can execute the most important functions using a toolbar button. When you move the mouse pointer over a toolbar, a brief description for the button is shown in the status bar.

The following toolbar buttons are available:

- **Change properties**
- **Connect to another buffer pool**
- **Disconnect current buffer pool**
- **Shutdown buffer pool server**
- **Start buffer pool server**
- Display information about the Buffer Pool Monitor
- Display online help

To switch the toolbar display on and off

- From the **View** menu, choose **Toolbar**.

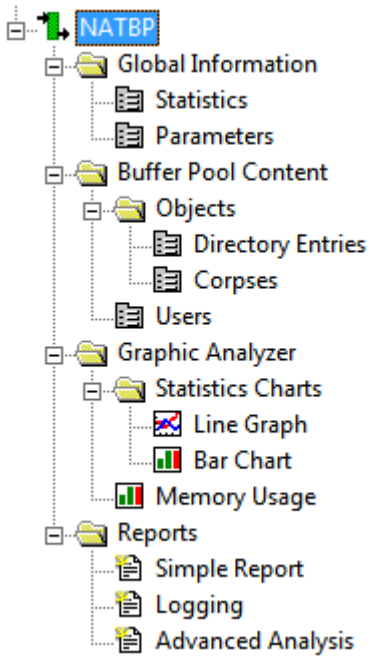
  When the toolbar is displayed in the **Natural Buffer Pool Monitor** window, a check mark is shown next to this menu command.

Tree

The tree on the left side of the **Natural Buffer Pool Monitor** window shows all buffer pools currently assigned in the local configuration file. See *Buffer Pool Assignments* in the *Configuration Utility* documentation.

Only one buffer pool can be monitored at a time. If you want to connect to a different buffer pool, see *Connecting and Disconnecting a Buffer Pool*.

When all nodes for the buffer pool which is currently used (**NATBP** in the example below) are expanded, the tree looks as follows.



When you select a node in the tree, the corresponding page is shown on the right side of the window. See the following sections for a detailed description of each page:

- **Global Information**
- **Buffer Pool Content**
- **Graphic Analyzer**
- **Reports**

Status Bar

The status bar is the horizontal information line at the bottom of the **Natural Buffer Pool Monitor** window. It shows short help texts for the commands in the menu bar and toolbar.

To switch the status bar display on and off

- From the **View** menu, choose **Status Bar**.

  When the status bar is displayed in the **Natural Buffer Pool Monitor** window, a check mark is shown next to this menu command.

Disconnecting and Connecting a Buffer Pool

Only one buffer pool can be connected at a time. To switch to another buffer pool in the environment, you disconnect the currently used buffer pool and then connect to the new buffer pool.

Note: When you connect to another buffer pool, the previously connected buffer pool is automatically disconnected. Thus, it is not necessary to use the **Disconnect** command first.

The icon next to a buffer pool name indicates one of the following states:

- The Buffer Pool Monitor is connected to the buffer pool (green icon).
- The Buffer Pool Monitor is not connected to the buffer pool (gray icon).

To disconnect the currently used buffer pool

1. Select the name of the currently connected buffer pool in the tree.
2. From the **Monitor** menu, choose **Disconnect**.

   Or:

   Invoke the context menu and choose **Disconnect**.

   Or:

   Choose the following toolbar button:

   The tree for this buffer pool is no longer shown.

To connect a buffer pool

1. Select the name of a buffer pool in the tree.
2. From the **Monitor** menu, choose **Connect**.

   Or:

   Invoke the context menu and choose **Connect**.

   Or:

   Choose the following toolbar button:

The tree for this buffer pool is shown.

Shutting Down a Buffer Pool Server

When you are connected to a buffer pool, you can shut it down. For example, if you want to initialize the buffer pool, you shut it down and then restart it.

The buffer pool server will not shut down as long as any Natural process is still connected. It will only shut down after the last process has disconnected from the buffer pool. As long as processes are connected, the buffer pool status is “shutdown pending”; this is indicated in the tree, next to the buffer pool name.

To shut down the buffer pool server

1 Select the name of the currently connected buffer pool in the tree.

2 From the **Monitor** menu, choose **Shutdown Server**.

Or:

Invoke the context menu and choose **Shutdown Server**.

Or:

Choose the following toolbar button:

The tree for this buffer pool is no longer shown.

Starting a Buffer Pool Server

A buffer pool server can only be started via the Buffer Pool Monitor if a buffer pool server has not yet been started.

To start a buffer pool server

1 Select the name of a buffer pool in the tree.

2 From the **Monitor** menu, choose **Start Server**.

Or:

Invoke the context menu and choose **Start Server**.

Or:

Choose the following toolbar button:

The buffer pool server is started.

This does not automatically connect the buffer pool. You have to connect it manually as described in *Disconnecting and Connecting a Buffer Pool*.

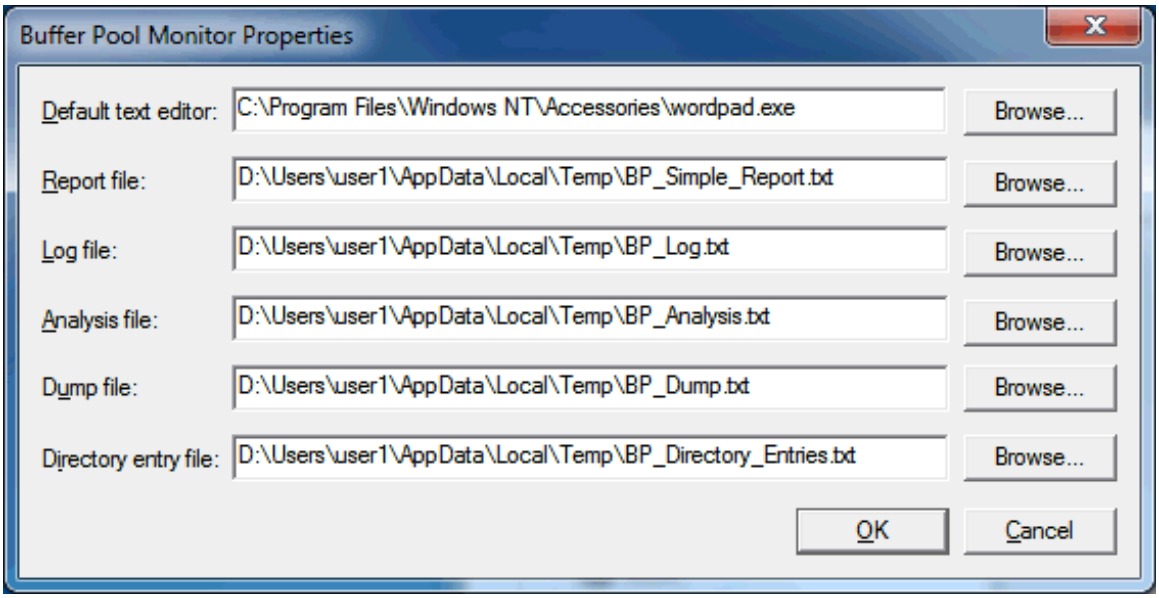
Changing the Properties of the Buffer Pool Monitor

You can define the files that are to be provided as the defaults on several pages of the Buffer Pool Monitor. You can also define the default text editor that is to be used.

To change the properties

1 From the **Monitor** menu, choose **Properties**.

The following dialog box appears. By default, a temporary directory is defined for the current user. Example:



You can change the following information:

- The default text editor to be used for opening the text files on the pages listed below.
- The report file to be used on the **Simple Report** page.
- The log file to be used on the **Logging** page.
- The analysis file to be used on the **Advanced Analysis** page.
- The dump file to be used on the **Advanced Analysis** page.
- The directory entry file to be used on the **Directory Entries** page.

2 If you want to change an entry, specify the path and file name in the corresponding text box.

Or:

Choose the corresponding **Browse** button to select the file from a dialog box.

3 Choose the **OK** button.

Global Information

When you expand the **Global Information** node in the tree, you can display statistical data of the buffer pool and its parameters.

The following pages are available.

- Statistics
- Parameters

Statistics

The following page appears when you select **Statistics** in the tree.

Statistics | Parameters

General information

BPID:	NATBP	Active since:	22-MAY-2013 9:11:00
Shutdown status:	active	Last time cleared:	22-MAY-2013 9:11:00

Memory allocation

Allocated memory:	727,192 bytes	710,15 KB
Smallest allocation:	20 bytes	0,02 KB
Largest allocation:	43,692 bytes	42,67 KB
Free memory:	2,418,548 bytes	2,31 MB
Smallest contiguous:	92 bytes	0,09 KB
Largest contiguous:	2,365,228 bytes	2,26 MB

User statistics

Current users:	1
Peak users:	2
Dead users purged:	0

Object use statistics

Dormant objects:	66
Active objects:	0
Generating objects:	0
Obsolete objects:	0

Object loading statistics

Stored objects:	0
Loaded objects:	422
Activated objects:	3368
Aborted loads:	240

Object size statistics

Largest object:	31,228 bytes	30,50 KB
Smallest object:	108 bytes	0,11 KB
Total object sizes:	668,569 bytes	652,90 KB

Locate statistics

Attempted locates:	4411
Attempted fast locates:	1788
Successful fast locates:	1202
Percent successful:	67,225951

General loading statistics

Objects purged:	0
Peak parallel activations:	2
Object reusage factor:	7,981043

Clear counters | Refresh | ☑ Automatic refresh

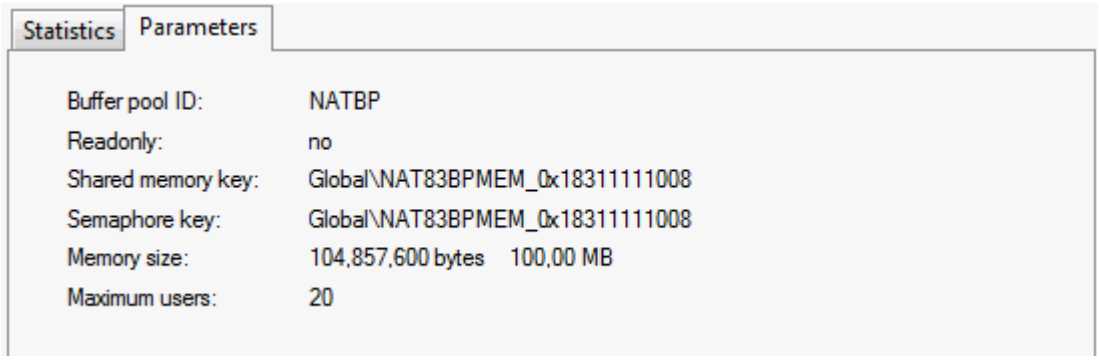
This page shows general information about the buffer pool and detailed information about memory, users and objects. It shows the same information as the `STATUS` command of the `NATBPMON` utility; see *Statistical Information About the Buffer Pool* for further information.

When the **Automatic refresh** check box is selected, the page is automatically refreshed every second. When this check box is not selected, you have to refresh the values manually by choosing the **Refresh** button.

When you choose the **Clear counters** button, the internal statistics of the buffer pool are reset to zero.

Parameters

The following page appears when you select **Parameters** in the tree.



This page shows the same information as the `PARAM` command of the `NATBPMON` utility; see *Displaying the Buffer Pool Settings*.

Buffer Pool Content

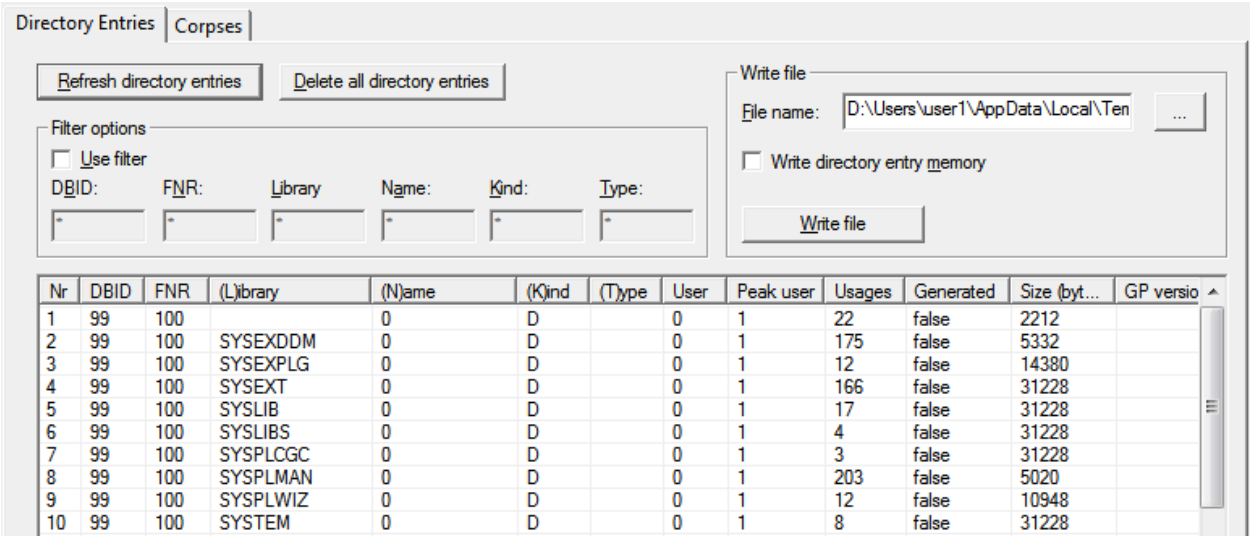
When you expand the **Buffer Pool Content** node in the tree, you can display details about the Natural objects which have been loaded into the buffer pool, as well as the users who are accessing them.

The pages described below are available.

- Directory Entries
- Corpses
- Users

Directory Entries

The following page appears when you expand the **Objects** node in the tree and select **Directory Entries**.



This page provides a table containing information on all currently loaded directory entries. It shows the same information as the `DIR` command of the `NATBPMON` utility; see also *Displaying the Objects in the Buffer Pool*.

The following command buttons are provided:

Command Button	Description
Refresh directory entries	Updates the table.
Delete all directory entries	Deletes all Natural objects which are currently loaded in the buffer pool.

When the mouse is positioned over the table, you can invoke a context menu containing the following commands:

Command in Context Menu	Description
Select all	Selects all entries in the table.
Delete	Deletes the selected entries in the table.

Filter options

Using a filter, you can reduce the number of directory entries that are shown in the table.

To define a filter

1 Activate the **Use filter** check box.

2 Specify the filter criteria in the text boxes **DBID**, **FNR**, **Library**, **Name**, **Kind** and/or **Type**.

For example, to display all programs in the libraries starting with "MY", specify "MY*" in the **Library** text box, and "P" in the **Type** text box.

3 Choose the **Refresh directory entries** button to update the table.

Write file

You can write all directory entries which are currently shown in the table to a file. If required, the memory of the directory entries can also be written to this file.

To write the directory entries to a file

1 Optional. In the **File name** text box, specify the path to the file to which the directory entries are to be written.

Or:

Use the button to the right of this text box to select the file from a dialog box.

Note: By default, the **File name** text box contains the path to the file which has been defined in the **properties**.

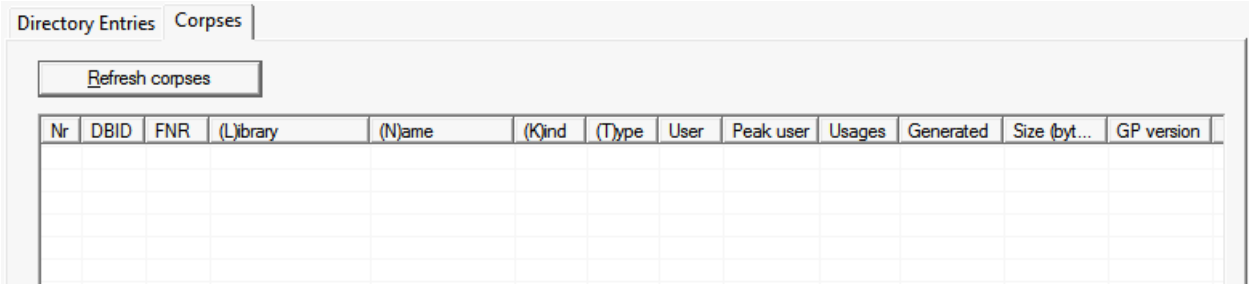
2 Optional. If the memory of the directory entries is to be written to this file, activate the **Write directory entry memory** check box.

3 Choose the **Write file** button.

The information is written to the specified file. The content of this file is automatically shown in the text editor which has been defined in the **properties**.

Corpses

The following page appears when you expand the **Objects** node in the tree and select **Corpses**.

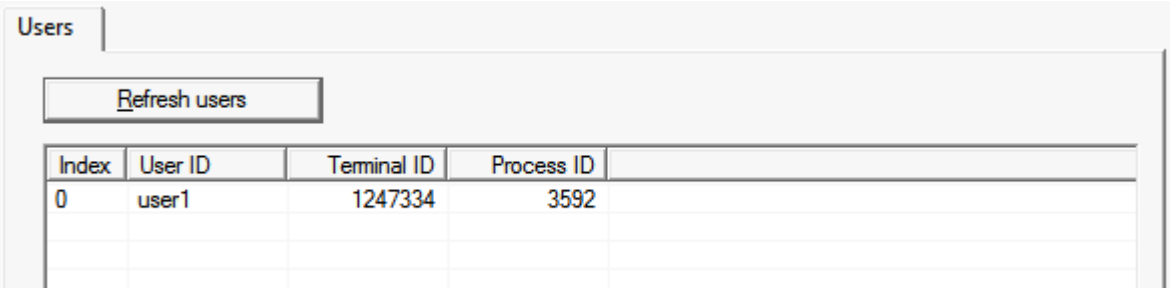


A corpse is an object which is to be deleted from the buffer pool, but is still in use. When corpses are available, they are shown in the table.

You can use the **Refresh corpses** button to update the table.

Users

The following page appears when you select **Users** in the tree.



This page shows a table containing information on the users who are currently using the buffer pool.

You can use the **Refresh users** button to update the table.

Graphic Analyzer

When you expand the **Graphic Analyzer** node in the tree, you can display graphical representations of the statistical numbers and a direct view on what is taking place inside the buffer pool memory.

The pages described below are available.

- Line Graph
- Bar Chart
- Memory Usage

Line Graph

The following page appears when you expand the **Statistics Charts** node in the tree and select **Line Graph**. When you have added data sources and have started the analyzer, this page may look as follows:

The line graph and the bar chart are both working with the same statistical data sources. When you apply one of the following actions to the line graph, this action is also applied to the bar chart, and vice versa:

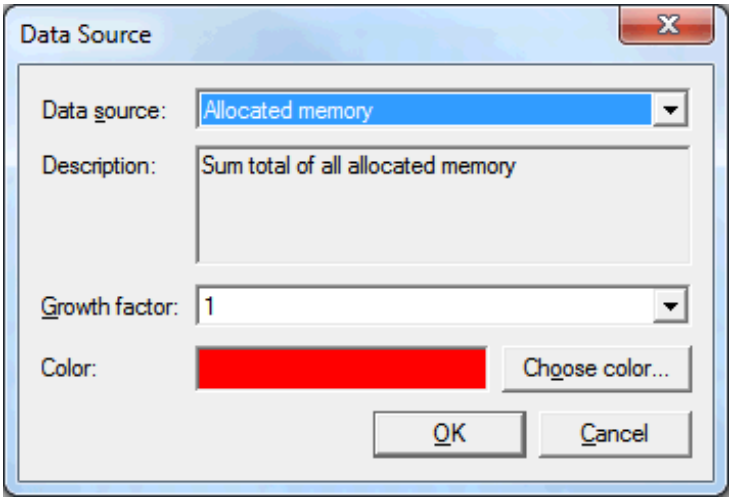
- add, modify or delete a data source,
- start, pause or reset the analyzer,
- adjust the update interval.

Note: The analyzer is also used on the **Memory Usage** page.

To add data sources

1 Choose the **Add data source** button.

The **Data Source** dialog box appears.



2 From the **Data source** drop-down list box, select the data source that is to be shown in the chart.

A description is shown for the selected data source.

3 Optional. From the **Growth factor** drop-down list box, select the required value for the selected data source.

This adjusts the range on the y-axis. The current value of the data source is multiplied by that factor to accomplish an appropriate representation on the chart.

4 Optional. If you want to define a different color for the selected data source, choose the **Choose color** button.

The standard Windows **Color** dialog box appears in which you can select or define another color to be used for the data source.

5 Choose the **OK** button to add the data source to the table which is shown at the bottom of the page. The data source is then available for both the line graph and the bar chart.

The table shows the color, name, description and growth factor of each data source that you have added. It also shows the minimum, maximum and current values of the data source.

6 Optional. Repeat the above steps, if you want to add further data sources to the table.

Managing the defined data sources

1 Select a data source in the table and invoke the context menu.

The context menu contains the following commands:

Command in Context Menu	Description
Properties	Invokes the **Data Source** dialog box for the selected data source. In this case, the dialog box can only be used to define another color.
Select all	Selects all data sources in the table.
Delete	Deletes the selected data source(s) in the table.

2 Choose one of the above commands.

To adjust the update interval

- From the **Interval** drop-down list box, select the update time (different values are provided for updating in milliseconds, seconds or minutes).

  The update interval is adjusted for the charts.

  Note: The **Interval** drop-down list box is only available when the analyzer is inactive.

To start the analyzer

- When all required data sources have been added to the table, choose the **Start analyzer** button.

  This starts the analyzer in all charts. The graphical representation of the selected data sources is painted in the line graph and in the bar chart.

To pause the analyzer

- Choose the **Pause analyzer** button.

  This freezes the current state of the graphical representation in all charts.

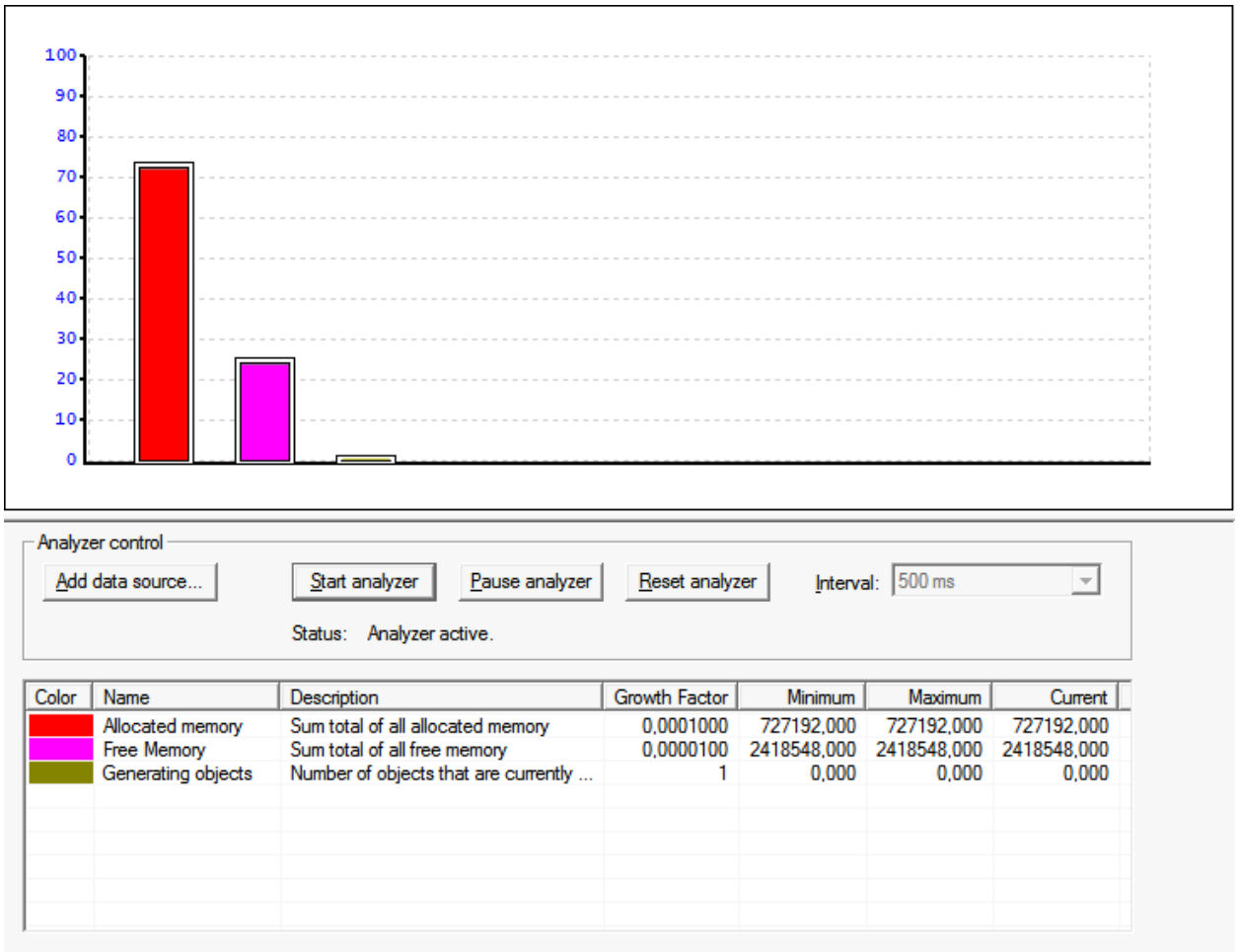
To reset the analyzer

- Choose the **Reset analyzer** button.

  This resets the graphical representation in all charts. For the line graph and bar chart, the minimum, maximum and current values are reset in the table. The time base which is shown in the line graph is also reset.

Bar Chart

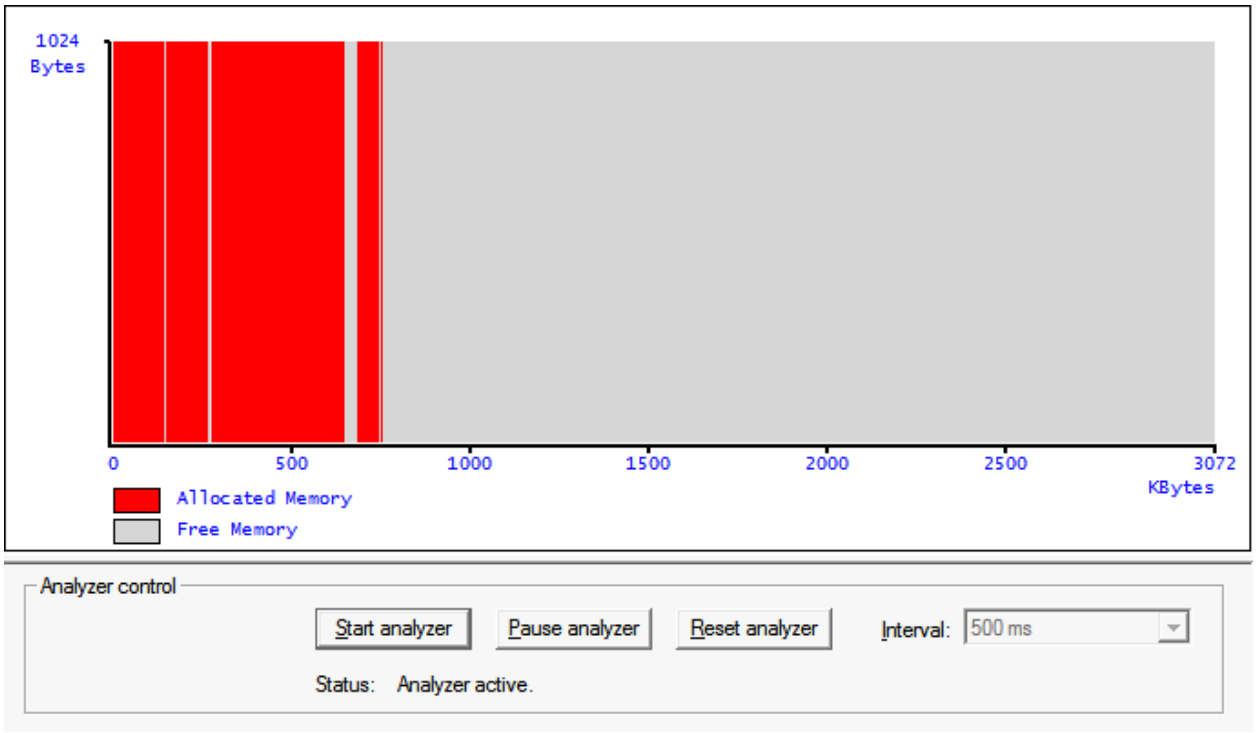
The following page appears when you expand the **Statistics Charts** node in the tree and select **Bar Chart**. When you have added data sources and have started the analyzer, this page may look as follows:



When you apply an action to the line graph, this action is also applied to the bar chart, and vice versa. See *Line Graph* for detailed information on how to add, modify and delete data sources, how to start, pause and reset the analyzer, and how to adjust the update interval.

Memory Usage

The following page appears when you select **Memory Usage** in the tree.



This chart simply shows the structure of the buffer pool memory. It shows allocated and free memory.

When the analyzer is active for a line graph or bar chart, it is also active on this page, and vice versa. See *Line Graph* for detailed information on how to start, pause and reset the analyzer, and how to adjust the update interval.

Reports

When you expand the **Reports** node in the tree, several pages are available. They can be used to write certain types of information about the buffer pool into a file.

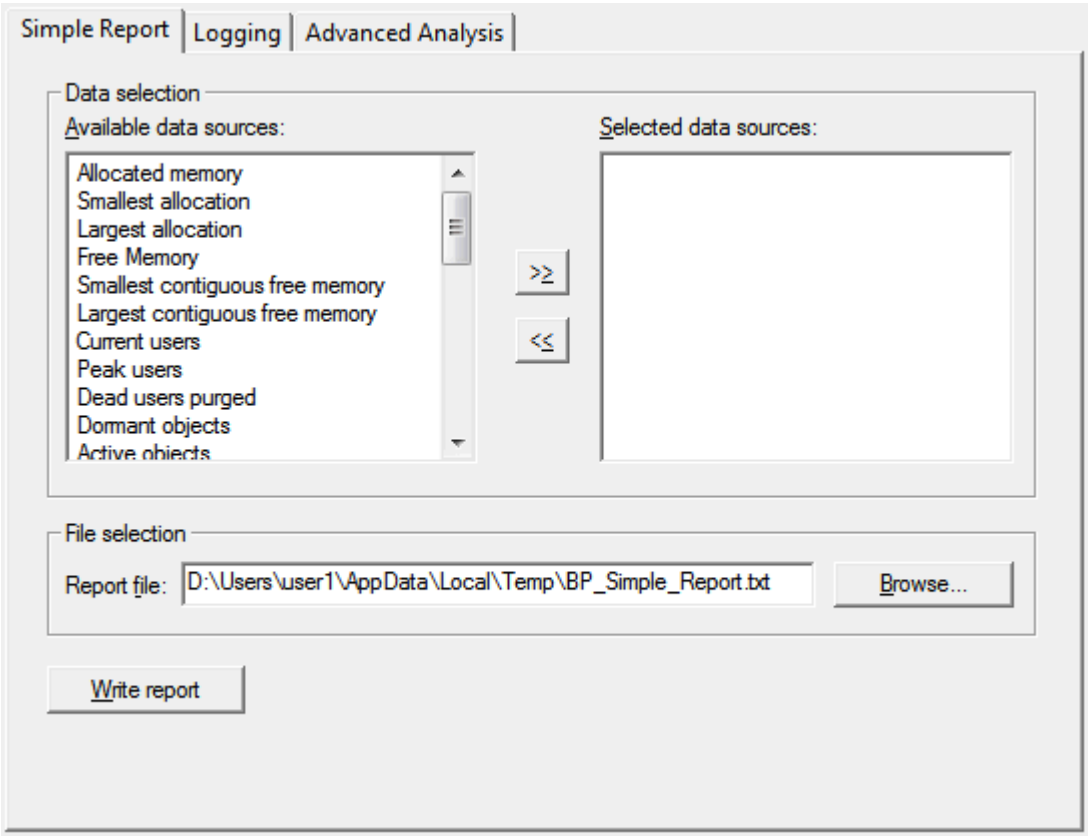
The pages described below are available.

- Simple Report
- Logging

- Advanced Analysis

Simple Report

The following page appears when you select **Simple Report** in the tree.



You can write a report which contains information on the data sources that you select on this page.

To select the data sources and write the report

1 Select one or more data sources in the **Available data sources** list box.

2 Choose the >> button.

The selected data sources are moved to the **Selected data sources** list box.

Note: If you have accidentally moved the wrong data source to the **Selected data sources** list box, you can move it back to the **Available data sources** list box by choosing the << button.

3 Optional. In the **Report file** text box, specify the path to the file to which the report is to be written.

Or:

Choose the **Browse** button to select the file from a dialog box.

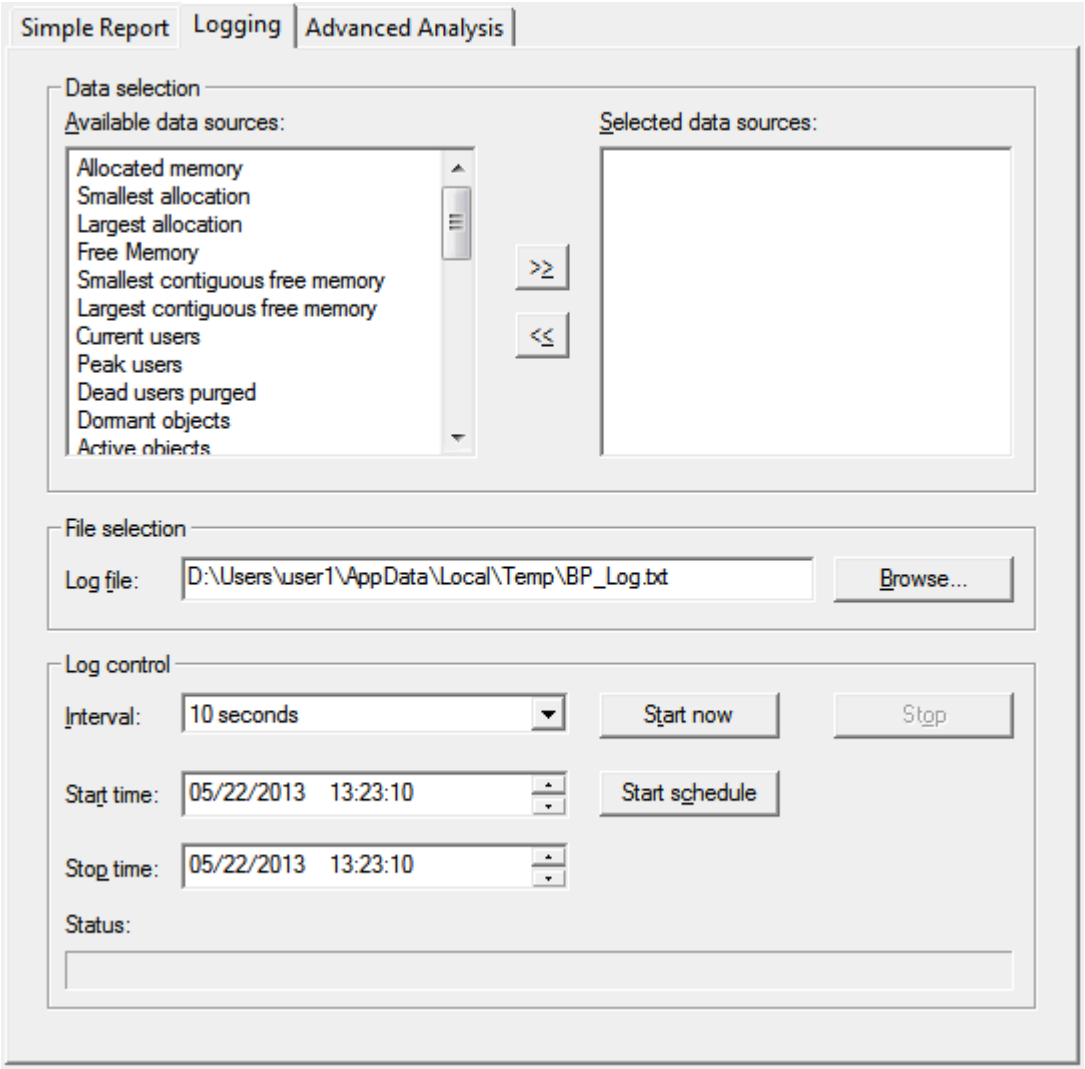
Note: By default, the **Report file** text box contains the path to the file which has been defined in the **properties**.

4 Choose the **Write report** button.

The report is written to the specified file. The content of this file is automatically shown in the text editor which has been defined in the **properties**.

Logging

The following page appears when you select **Logging** in the tree.



The upper part of this page contains the same information as the **Simple Report** page. The only difference is that a different log file is used by default.

In addition to selecting the data sources in upper part of this page, you can decide whether the log file is to be written immediately (manually) or whether it is to be scheduled for a specific time range.

To start the logging process manually

1. Select all data sources and (optionally) the log file as described for the **Simple Report** page.
2. From the **Interval** drop-down list box, select the interval which determines how often the log information is to be written to the file.
3. Choose the **Start now** button to start writing information to the log file.

   A status message indicating the number of done circles and the elasped time is shown at the bottom of the **Log control** group box.
4. Choose the **Stop** button to stop writing information to the log file.

   When the logging process has been stopped, the content of the log file is automatically shown in the text editor which has been defined in the **properties**.

To schedule the log process for a specific time range

1. Select all data sources and (optionally) the log file as described for the **Simple Report** page.
2. From the **Interval** drop-down list box, select the interval which determines how often the log information is to be written to the file.
3. Specify a start date and time.
4. Specify a stop date and time.
5. Choose the **Start schedule** button.

   A status message is shown at the bottom of the **Log control** group box. It indicates the time that is to elapse until the log process is started. When the start time is reached, a different status message is shown which indicates the number of done circles and the elasped time.

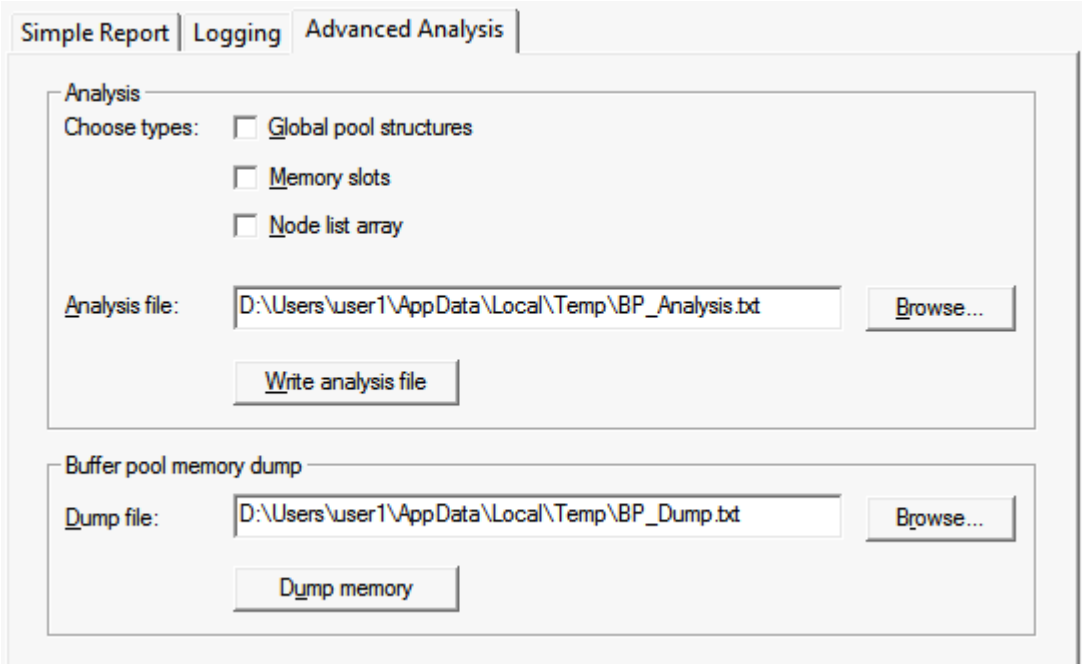
   Note: You can choose the **Stop** button to cancel the schedule before the specified start time.

   When the stop time is reached (or when you choose the **Stop** button after the start time has been reached), the content of the log file is automatically shown in the text editor which has been defined in the **properties**.

Advanced Analysis

The following page appears when you select **Advanced Analysis** in the tree. It provides information for the support team.

Important: Do not use this page unless you are requested to do so by Support.



9 Using the Command Line Version of the Buffer Pool Monitor (NATBPMON)

- Invoking the NATBPMON Utility 82
- NATBPMON Commands 83
- Displaying the Objects in the Buffer Pool 85
- Specifying a Pattern 86
- Displaying the Buffer Pool Settings 87
- Statistical Information About the Buffer Pool 88

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.

If the maximum buffer pool user limit is reached, the user accesses the utility as an emergency user.

To invoke the `NATBPMON` utility

1. Invoke the **Command Prompt** window of Windows.
2. Go to the Natural *bin* directory which contains the file *natbpmon.exe*.
3. If the default buffer pool `NATBP` is to be used, enter the following command in the Command Prompt window:

   ```
   NATBPMON
   ```

   Or:

   If another buffer pool is to be used, enter the following command in the Command Prompt window:

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

   The following prompt appears:

   ```
   NATBPMON>
   ```

4. If you want the `NATBPMON` utility to terminate with an appropriate error message, add `GIVERC` to your command in the Command Prompt window.

To use NATBPMON utility as a NATBPMON administrator

1. If you are not already defined as a `NATBPMON` administrator, define yourself as one in the NAT-ADM section of the *NATURAL.INI* file:

```
[NAT-ADM]               # section name

[USERNAME]              # replace "USERNAME" with your username in upper case
BPADM = YES             # “YES” defines user as a NATBPMON administrator
[USERNAME-END]          # replace "USERNAME" with your username in upper case

[NAT-ADM-END]
```

2 Invoke the NATBPMON utility as described in the preceding section.

When a NATBPMON administrator invokes the monitor but the regular user limit has already been reached, it will automatically use the buffer pool's reserved emergency user slot. NATBPMON administrators can also perform operations that are not available to regular users, such as recovering a stalled or deadlocked buffer pool.

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. **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 more 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. **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.
LOCKS	Displays the users (if any) who currently own the buffer pool lock, and any users who are blocking buffer pool readers. Although the latter users do not own a lock, they can still prevent progress. A process holding the buffer pool lock must wait for all readers to complete before it can modify the buffer pool.

Command	Description
	If there are no current lock owners or blocking readers, no output is produced. Otherwise, the reported users are reported with their process IDs, which can be used to terminate the corresponding processes via an external utility like the Task Manager or the Windows `taskkill` command. The information displayed by the `LOCKS` command is a snapshot in time. To distinguish persistent entries from transient activity, the command should be invoked repeatedly. **Note:** This is the only command that clears emergency mode (see below), provided it determines that the conditions which caused the emergency mode are no longer present.
`PARAM`	Displays the buffer pool settings. For more information, see ***Displaying the Buffer Pool Settings***.
`QUIT`	Exits the `NATBPMON` utility. This is the same as the `EXIT` command.
`SHUTDOWN`	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. **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. 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. **Note:** Users should not be overly concerned by an assertion failure if the buffer pool otherwise appears to be operating normally and the `STATUS` command does not report any unexpected or suspicious statistics. In this case, a restart of the buffer pool is generally not urgent.
`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).

Command	Description
WRITE	Writes a buffer pool object onto the disk. You are prompted to specify an index and a file name. **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.

Notes:

1. If the monitor utility determines that the buffer pool may be deadlocked or stalled on startup, it switches to emergency mode, in which most commands are unavailable because they would otherwise be likely to block. In this case, the LOCKS command can be used to identify the blocking user processes.
2. The same commands that are unavailable in emergency mode are also unavailable when the maximum user limit for the buffer pool has been reached. However, one additional user is allowed if the user is defined as a Natural buffer pool administrator in the local configuration file (see above).

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.

Column	Explanation	
	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*; `R`=resource).
	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 *Configuration Utility*.

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. 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. 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. 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. 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.

10 Natural in Batch Mode

- About Batch Mode 94
- Starting a Natural Session in Batch Mode 94
- Terminating a Natural Session in Batch Mode 95
- Using Natural in Batch Mode 95
- Sample Session for Batch Mode 97
- Batch Mode Detection 100
- Batch Mode Restrictions 100
- Hints for Using Natural Maps and Dialogs in Batch Mode 101

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

About Batch Mode

Natural distinguishes between two processing modes:

- interactive mode (with Natural Studio)
- 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:

```
nderun BATCHMODE
```

or

```
natural BATCHMODE
```

nderun.exe starts the runtime version of Natural. *natural.exe* starts the development version of Natural. Other than *natural.exe*, *nderun.exe* does not start plug-ins. Plug-ins are not needed when running Natural in batch mode. It is therefore recommended to use *nderun.exe* since this prevents errors due to plug-in activation failures.

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

- Code Pages for the Input and Output Files

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.
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`

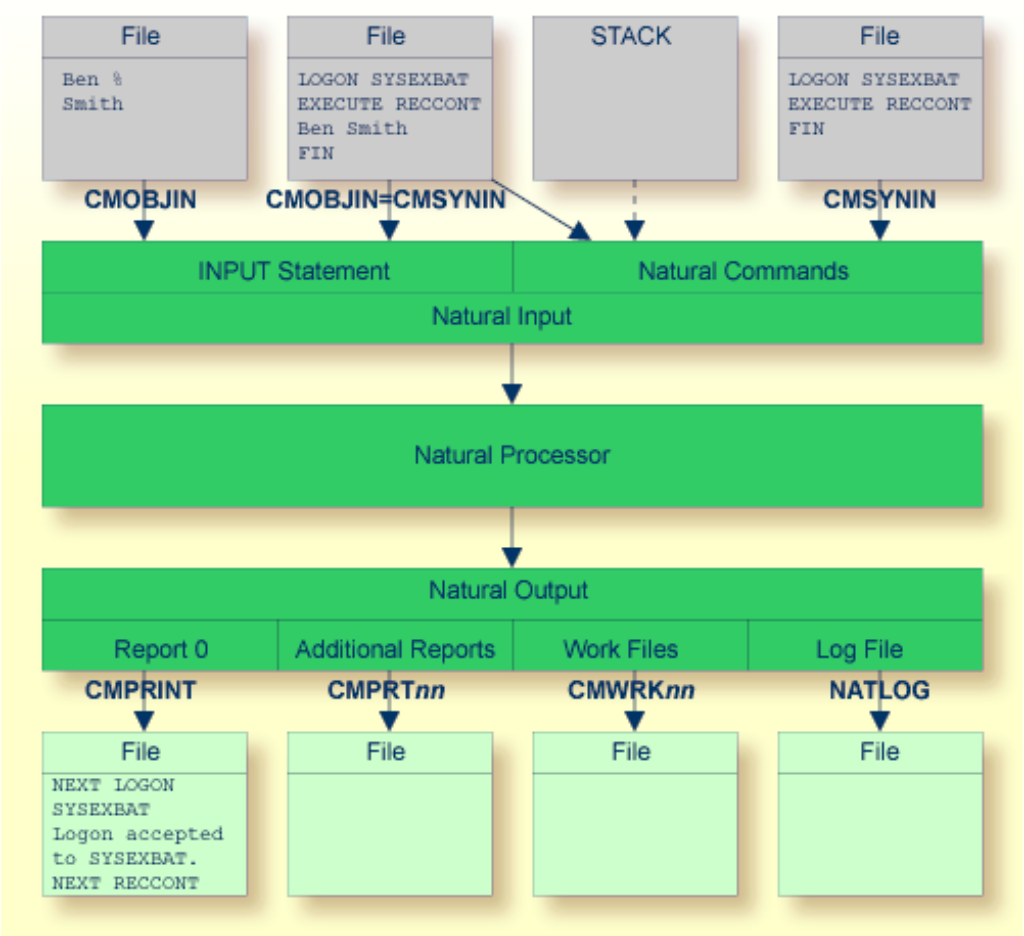
The `DISPLAY` statement in the program `RECCONT` writes the data to the batch output file *out.txt* which is created in your file system. The content of this file is shown below:

```
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.
- There is no mouse support.
- No different character fonts are available.
- Only data for an `INPUT` statement can be processed. Dialog input is only conditionally supported (see *Hints for Using Natural Maps and Dialogs in Batch Mode*).

- The output appearance is not GUI-like (it is character-oriented output).
- 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.

Hints for Using Natural Maps and Dialogs in Batch Mode

If an application is designed to run in batch mode as well as in interactive mode, the following considerations should be taken into account.

Within Natural, there are two ways to read input data:

- using a map (by using an `INPUT` statement or the Natural object Map),
- using a dialog (by using the Natural object Dialog).

In batch mode, data have to be processed using an `INPUT` statement, because a dialog does not allow data processing in batch mode. Terminal commands for navigating and controlling data are also not supported by a dialog. Nevertheless, a dialog may be executed in batch mode. In this case, however, the dialog must be altered in the following way:

- The dialog attribute `VISIBLE` must be set to `FALSE`.
- Within the event `AFTER-OPEN`, code should be inserted to read data during batch mode processing. If Natural runs in batch mode, an `INPUT` statement should be coded to get the input data. For interactive mode, the dialog attribute `VISIBLE` has to be set to `TRUE` to make the dialog visible.

  Example for the `AFTER-OPEN` event:

```
IF *DEVICE EQ "BATCH" THEN
/* Batch mode processing: call a map */
   INPUT USING MAP "BATCHINP" #p1 #p2 #p3

/* ... further data processing ... */

/* Close dialog immediately */
   CLOSE DIALOG *DIALOG-ID
   ELSE
/* Interactive mode processing: make dialog visible */
   #DLG$WINDOW.VISIBLE = TRUE
END-IF
```

- If there is a `CLOSE` event, ensure that the appropriate code does not contain any GUI actions in batch mode.

Example for the `CLOSE` event:

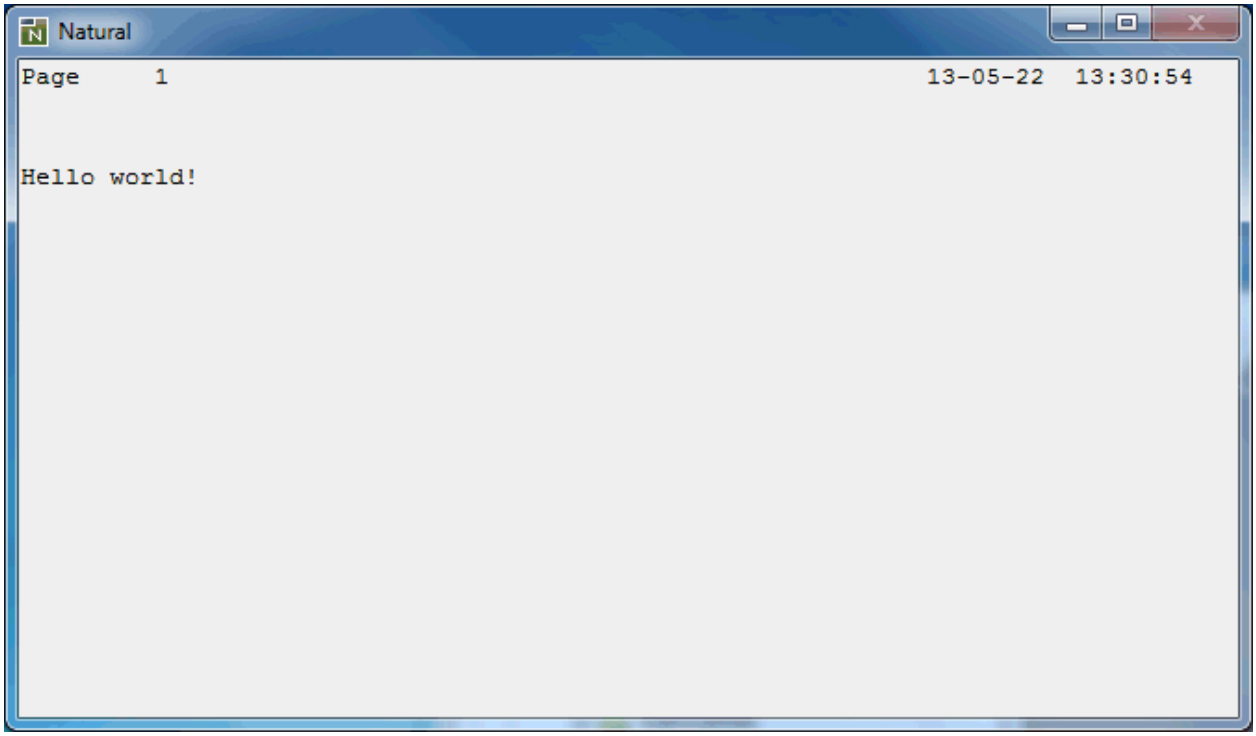
```
IF *DEVICE NE "BATCH" THEN
/* ... GUI actions ... */
END-IF
```

11 Output Window

- About the Output Window 104
- Working in the Output Window 104
- Changing the Output Window Profile 105
- Using Your Own Icon for the Output Window 106

About the Output Window

In the local environment, an output window appears when a Natural program writes output to the screen. Example:



Note: In a remote development environment, a terminal emulation window or a Natural Web I/O Interface client appears instead of an output window. See *Terminal Emulation* and *Natural Web I/O Interface Client* in the *Remote Development Using SPoD* documentation.

Working in the Output Window

A button for the output window is shown in the Windows taskbar; keep this in mind when you minimize the output window.

The output window can be resized and moved. If the output window is smaller than the Natural output page, scroll bars appear.

PF keys defined in a Natural program are converted to command buttons. You can either use these command buttons or the PF keys on your keyboard.

Information from an input field in the Natural output can be cut or copied to the Windows clipboard, and information from the Windows clipboard can be pasted into an input field of the output window.

The cursor can be positioned using the mouse. Double-clicking the left mouse button simulates the ENTER key. The system variables `*CURSOR`, `*CURS-COL` and `*CURS-LINE` are always set to the current mouse position.

Windows created using the statement `DEFINE WINDOW` or the terminal command `%W` are placed into the output window. They are moveable, sizeable, and scrollable child windows of the output window.

The output window is automatically closed when the program terminates. It cannot be closed manually using the standard Windows close button. If you want to close the output window before the regular termination of the program, you can press ESC (provided that this feature has not been disabled; see below).

Changing the Output Window Profile

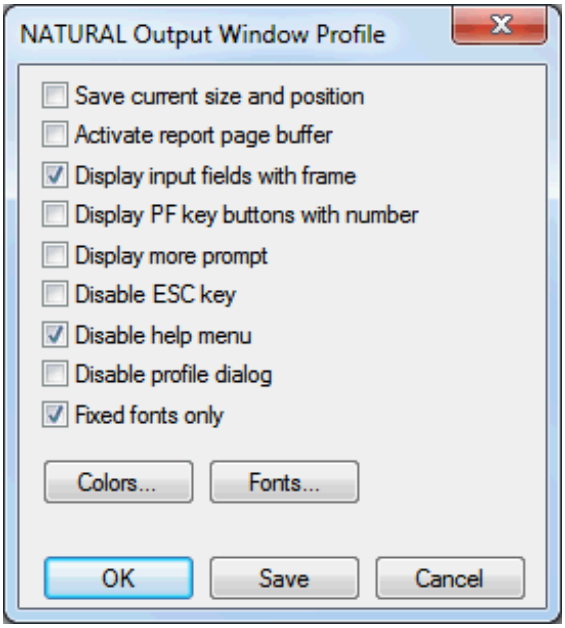
Several options can be set for the output window. These options can either be set directly in the output window (as described below) or in Natural Studio (see *Setting the Options* in the documentation *Using Natural Studio*).

To change the output window profile

1 From the control menu of the output window, choose **Profile**.

Note: This command is not available when the display of the profile dialog has been disabled. In this case, the output window options can only be set in Natural Studio.

The following dialog box appears.



This dialog box contains the same options as the **Options** dialog box of Natural Studio. There is one exception: the following command is only available in the above dialog box since it can only be specified for a running application:

Save current size and position
When this check box is selected, the current size and position of the output window is saved.

2 Set all required options. For a detailed description of each option, see *Output Window Options* in the documentation *Using Natural Studio*.

3 Choose the **Save** button to save your changes permanently.

Or:

Choose the **OK** button to save your changes for the current session only.

Using Your Own Icon for the Output Window

Instead of using the Natural icon for the output window, you can use your icon. This icon will be shown in the control menu of the output window and in the Windows taskbar.

To use your own icon

1 Create an icon file (*.*ico*).

2 Store your icon file in a directory where it can be found by Natural.

Natural tries to find the icon file first in the *RES* subdirectory of the logon library, then in the *RES* subdirectory of each steplib and then in the directory assigned to the Windows environment variable `NATGUI_BMP`.

3 To use your own icon file, use the following statement in your program:

```
SET CONTROL 'I=icon-file-name.ICO'
```

See also the description of the terminal command `%I`.

12 Natural Runtime

- Unsupported Functions by Natural Runtime 110
- Porting Procedure Overview 111
- Step 1 - Packaging the Application on the Development Workstation 111
- Step 2 - Installing Natural Runtime 115
- Step 3 - Installing the Application on the Runtime Workstation 116
- Step 4 - Starting the Application on the Runtime Workstation 118
- Using the Natural Runtime Startup Service 119

Natural Runtime is used to execute applications that have been written using the development version of Natural for Windows.

This chapter tells you how to port an application from a Natural development workstation to a Natural Runtime workstation. This porting process can be used for a first-time installation of Natural Runtime and for Natural Runtime workstation updates.

You will also learn how to use a service for starting Natural Runtime processes.

Important: Before porting an application to a Natural Runtime workstation, ensure that all objects have been compiled using identical Natural and Natural Runtime versions.

Unsupported Functions by Natural Runtime

System Commands

The following Natural system commands are not supported by Natural Runtime:

```
CATALL
CATALOG
CHECK
CLEAR
COMPOPT
DEBUG
DELETE
EDIT
GLOBALS
PURGE
READ
RENUMBER
RUN
SAVE
SCAN
SCRATCH
STOW
STRUCT
SYSDDM
SYSMAIN
UNCATALOG
UNLOCK
```

Editors

Natural editors are not supported by Natural Runtime.

Utilities

Natural utilities providing developer functionality are not supported by Natural Runtime.

Porting Procedure Overview

To port an application to a runtime workstation, the following steps (which are described in detail later in this section) are required:

1. Package the application on the development workstation:
   a. Create a collecting directory in your file system.
   b. Customize the global configuration file and copy it to the collecting directory.
   c. Customize the Natural parameter file and copy it to the collecting directory.
   d. Copy or unload all required objects to the collecting directory.
   e. Copy the contents of the collecting directory to a transfer medium (for example, to a CD).
2. Install Natural Runtime on the runtime workstation.
3. Install the application on the runtime workstation:
   a. Copy the global configuration file from the transfer medium to the runtime workstation.
   b. Copy the Natural parameter file from the transfer medium to the runtime workstation.
   c. Copy or load the Natural objects from the transfer medium to the runtime workstation.
4. Start the application on the runtime workstation.

See also *Transferring Natural Generated Programs* in the *Programming Guide*.

Step 1 - Packaging the Application on the Development Workstation

The following topics are covered below:

- Creating a Collecting Directory
- Customizing and Copying the Global Configuration File
- Customizing and Copying the Natural Parameter File
- Copying or Unloading the Objects

- Copying the Collecting Directory to a Transfer Medium

Creating a Collecting Directory

Use the Windows Explorer to create a new directory in the file system of the development workstation. Use this temporary directory to collect all files which belong to the application.

Customizing and Copying the Global Configuration File

You have to create a global configuration file which contains all settings required to run your application on the runtime workstation. To do so, you create a backup version of your current global configuration file, make all required changes for the runtime version, copy the customized global configuration file to the collecting directory and then restore your old global configuration file. This is described in detail below.

To customize and copy the global configuration file

1 Use the Windows Explorer to back up the existing global configuration file.

> **Note:** If you do not know where to find the global configuration file, invoke the Configuration Utility, expand the **Local Configuration File** node and select the **Installation Assignments** node. The full path and name of the global configuration file is then shown.

2 Invoke the Configuration Utility and expand the **Global Configuration File** node.

3 Adjust the settings of the global configuration file as required for your application and save your changes. See also the notes below.

> **Important:** As long as the global configuration file with the settings for the runtime environment is active, you cannot work with the development version of Natural.

4 Use the Windows Explorer to copy the customized global configuration file to the collecting directory.

5 Use the Windows Explorer to restore the backup version of the global configuration file.

Note:

Make sure that with every new application you are porting, the new settings are compatible with the old settings.

Example: Your first application accesses an SQL database and the DBID entry applies to this SQL database. Your second application, which you are porting at a later date, accesses an Adabas C database. In this case, you must add a second DBID entry for Adabas C. If you do not add a second

entry, the new global configuration file will overwrite the SQL database's DBID and your first application will no longer be able to access its database.

Customizing and Copying the Natural Parameter File

You have to create a Natural parameter file which contains all settings required to run your application on the runtime workstation.

To customize the Natural parameter file

1 Invoke the Configuration Utility and expand the node for the required parameter file.

2 Adjust the settings of the parameter file as required for your application to run in the runtime environment.

Ensure that the parameter file contains the name of the program to be started. The examples below show different possibilities for this purpose.

Example 1:

Parameter	Required Setting
AUTO	Must be set to ON so that an automatic logon is executed at the start of the Natural session.
INIT-LIB	The name of the library into which the application is to be moved.
STARTUP	The name of the program that is to be started.
USER	The default user ID that is to be set when Natural is started.

Example 2:

Parameter	Required Setting
STACK	Must contain the library and the program to be started. For example: LOGON MYLIB;EXECUTE MYAPP

3 Save the modified parameter file with the name that you want to use in the runtime environment (for example, with the name RUNPARM).

4 Use the Windows Explorer to copy the customized parameter file (which has the extension *SAG*) to the collecting directory.

Note: If you do not know where to find the parameter file, expand the **Local Configuration File** node of the Configuration Utility and select the **Installation Assignments** node. The location of the Natural parameter files is then listed as **Path to parameter**.

Copying or Unloading the Objects

To make compiled code available with Natural Runtime, you have to copy the cataloged objects from the Natural development environment to the runtime environment.

If the Natural application consists of complete Natural libraries, you can copy the libraries with the copy-and-paste functionality of the Windows Explorer.

Another way for porting the objects is use the Object Handler for unloading the objects in the Natural development environment and for loading them in the runtime environment.

To copy the objects

1 Use Natural Studio to create a new library which is to contain all objects for the runtime version.

Important: If the application consists of more than one library, create a new library for each library that is used by the application and proceed as described below.

2 Use Natural Studio to copy all cataloged objects, resources and error messages from the development library to the new library. Do not copy the sources.

3 Use the Windows Explorer to copy the entire directory which corresponds to the new library (including the file *FILEDIR.SAG* and the subdirectories *GP*, *RES* and *ERR*) to the collecting directory.

Notes:

1. If you do not know where to find this directory, execute the system command `SYSPROF` in Natural Studio. The **Files in File System** tab of the resulting dialog box shows the path to the directory that has been created for the system file `FUSER`. Your new library is a subdirectory of the `FUSER` directory; it has the same name as defined in Natural Studio.
2. You can also find out the path to the `FUSER` directory by using the Configuration Utility: select the parameter file that you have created in a previous step (that is: the parameter file that will be used to start the application in the runtime environment), expand the **Natural Execution Configuration** node and select the **System Files** node. The path to the `FUSER` directory is shown on the **FUSER** tab.

4 If required, rename the copied directory in in the collecting directory: enter the name of the library that is to be used in the runtime environment.

To unload the objects

1 From the **Tools** menu, choose **Development Tools > Object Handler** to start the Object Handler.

2 Start the Unload Wizard.

3 In the first dialog of the unload wizard, select the option button **Unload objects into Natural work file(s)**.

4 In the next dialog in which you have to specify the options settings, define a Natural work file in the **Unload file** text box. This work file must be located in the collecting directory which you have created previously.

5 If the application uses the same library names in both environments, do not specify any information in the next dialog (which can be used to specify parameters).

   However, if the application uses library names in the runtime environment which are different from those used in the development environment, select the option button **Use global parameters**, choose the **Set** button and set the name in the resulting dialog box.

6 In the next dialog in which you have to specify the object type, select the option button **Natural library objects**.

7 In the next dialog in which you have to select the Natural library objects to unload, choose the **Details** button.

8 In the resulting dialog box, specify all cataloged objects, resources and error messages contained in the application. Do not unload the sources: from the **S/C-Kind** drop-down list box, choose **Gp**.

9 Proceed to the next dialog and unload the objects.

10 After a successful unload, check the work file that has been created in the collecting directory. Use the Load Wizard to scan the work file for all objects.

Copying the Collecting Directory to a Transfer Medium

When all files in the collecting directory are ready for porting, use the Windows Explorer to copy the contents of the collecting directory (including all subdirectories) to the transfer medium (for example, to a CD).

Step 2 - Installing Natural Runtime

Install Natural Runtime on the runtime workstation. See the *Installation* documentation for further information.

Note: This step is not required when updating applications on the Natural Runtime workstation.

Step 3 - Installing the Application on the Runtime Workstation

The following topics are covered below:

- Copying the Global Configuration File
- Copying the Natural Parameter File
- Copying or Loading the Objects

Copying the Global Configuration File

Use the Windows Explorer to copy the global configuration file (in which the required DBID has been defined) from the transfer medium to the directory on the runtime workstation which contains the global configuration file.

Notes:

1. An existing global configuration file will be overwritten!
2. If you do not know where to find the global configuration file, invoke the Configuration Utility on the runtime workstation, expand the **Local Configuration File** node and select the **Installation Assignments** node. The full path and name of the global configuration file is then shown.

Copying the Natural Parameter File

Use the Windows Explorer to copy the Natural parameter file from the transfer medium to the directory on the runtime workstation which contains the Natural parameter files.

Notes:

1. An existing parameter file will be overwritten!
2. If you do not know where to find the parameter files, select the **Installation Assignments** node of the Configuration Utility as described above. The location of the Natural parameter files is then listed as **Path to parameter**.

Copying or Loading the Objects

Depending on how the objects have been packaged (see *Copying or Unloading the Objects*), the transfer medium contains either complete libraries or a Natural work file.

If complete libraries have been copied using the Windows Explorer, the transfer medium contains directories with Natural library names. Each directory reflects the Natural library structure: it contains the file *FILEDIR.SAG* and the subdirectories *GP*, *RES* and *ERR*. In this case, you have to copy the libraries as described below.

If the objects have been unloaded into a Natural work file using the Object Handler, the transfer medium contains this work file. In this case, you have to load the objects using the Object Handler as described below.

To copy the libraries

- Use the Windows Explorer to copy the libraries (including all subdirectories) to the directory for the `FUSER` system file.

  Note: If you do not know where to find this directory, invoke the Configuration Utility, select the parameter file that you have copied in a previous step, expand the **Natural Execution Configuration** node and select the **System Files** node. The path to the `FUSER` directory is shown on the **FUSER** tab.

To load the objects

1. Invoke the Configuration Utility and make sure that the `FUSER` settings of the parameter file `NATPARM` have the same DBID and FNR as the parameter file that you have copied in a previous step.

   Note: The DBID and FNR of the `FUSER` are shown on the **FUSER** tab which is invoked as described above.

2. From the Windows **Start** menu, navigate to **Tools > Natural Runtime** *n.n*.

   This invokes Natural Runtime with the standard parameter file `NATPARM`.

3. Logon to the library `SYSOBJH`.
4. Execute the program `MENU` in the library `SYSOBJH`.

   The Object Handler window appears.

5. In the first dialog of the load wizard, select the option button **Load objects from Natural work file(s)**.

6 In the next dialog in which you have to specify the options settings, define your Natural work file in the **Load file** text box. This must be the work file which is located on the transfer medium.

> **Note:** If DBID and FNR of the new parameter file differ from the standard `NATPARM` settings, enter the values used by the new parameter file in the next dialog (which can be used to specify parameters): Select the option button **Use global parameters** and choose the **Set** button. In the resulting dialog box, select the **Load Target** tab and enter the corresponding values for DBID and FNR in the**Load FUSER** group box.

7 In the next dialog in which you have to specify the object type, select the option button **Load all objects from work file**.

8 Proceed to the next dialog and load the objects.

9 Exit the Object Handler and then exit Natural Runtime.

Step 4 - Starting the Application on the Runtime Workstation

When all required files have been copied to the runtime workstation, you can start your application. It is recommended that you create a shortcut for each application. You can then define the name of the parameter file which is required to run the application in the shortcut.

On the runtime workstation, you can start the application in different ways:

- **With *naturalr.exe***
  The user interface, which appears when you do not specify the name of a parameter file with the *naturalr.exe* command, runs invisibly in the background. For example:

  ```
  naturalr.exe PARM=file-name
  ```

  where *`file-name`* is the name you have assigned to your **customized Natural parameter file** (without any file extension).

  The user interface becomes visible only if the application for which it was started does not terminate this runtime process properly (for example, if the application does not issue a `TERMINATE` statement). This user interface increases the consumption of system resources, even if it does not appear.

- **With *natrt.exe***
  This so-called "mini runtime" does not have a user interface that would allow the user to select an application for execution. It requires that the name of the program that is to be started is defined in the parameter file. For example:

  ```
  natrt.exe PARM=file-name
  ```

where `file-name` is the name you have assigned to your **customized Natural parameter file** (without any file extension).

The mini runtime terminates as soon as all commands in the parameter file have been processed.

If a program name is not specified in the parameter file, the mini runtime terminates immediately.

When you use the mini runtime, the consumption of system resources is decreased and it is ensured that the runtime process terminates at the end of the application processing.

Using the Natural Runtime Startup Service

When the Natural Runtime startup service has been installed and is active, it is possible to start one or more Natural Runtime processes automatically when the PC is booted.

You can define parameter templates which are used to hold Natural parameters. It is thus possible to start a Natural Runtime process with all parameters that are defined in the template.

A Natural Runtime process is normally used to run a Natural application. For this purpose, the Natural Runtime process has to be started with a template in which the `STACK` parameter has been defined as follows:

```
STACK=(LOGON library-name; program-name)
```

When the `STACK` parameter has not been defined, Natural is started without running any application.

By default, the Natural Runtime startup service is not installed. You have to install it as described below.

The following topics are covered below:

- Natural Runtime Startup Service Commands
- Starting a Natural Process Automatically

Natural Runtime Startup Service Commands

The file *natrtsvc.exe*, which is stored in the *bin* directory of Natural Runtime, is used to execute the service commands.

The following service commands can be specified in the **Command Prompt** window of Windows:

Command	Description
`NATRTSVC INSTALL` *`mode`*	Installs the Natural Runtime startup service. *`mode`* can be one of the following: manual — Default. The service is installed and must be started manually (either with the `START` command or by starting the Natural Runtime Startup Service in Windows). automatic — The service is installed and is automatically started when the PC is booted.
`NATRTSVC REMOVE`	Removes the Natural Runtime startup service from the system.
`NATRTSVC START`	Starts the Natural Runtime startup service if it had not been started yet. The service searches for previously created parameter templates for which the `start` parameter has been set to "yes". In addition, it starts a Natural Runtime process with the Natural parameters which are also stored in the template.
`NATRTSVC START` *`template-name`*	Starts a Natural Runtime process with the Natural parameters stored in the specified template. If the Natural Runtime startup service has not been started (automatically at boot time or manually by the user) an error message is displayed.
`NATRTSVC STOP`	Stops the Natural Runtime startup service and all Natural Runtime processes that have been started by the Natural Runtime startup service.
`NATRTSVC STOP` *`template-name`*	Stops the Natural Runtime processes that have been started by the Natural Runtime startup service with the Natural parameters stored in the specified template.
`NATRTSVC CREATE` *`template-name`*	Creates a new parameter template to be started by the Natural Runtime startup service.
`NATRTSVC DELETE` *`template-name`*	Deletes the specified template from the Natural Runtime startup service.
`NATRTSVC SET` *`template-name`* `start=`*`mode`*	Defines whether a Natural Runtime process with the Natural parameters stored in the specified template is to be started when the Natural Runtime startup service is started. *`mode`* can be one of the following: yes — A Natural Runtime process is started. no — Default. A Natural Runtime process is not started.
`NATRTSVC SET` *`template-name`* *`Natural-parameters`*	Stores the Natural parameters in the specified template. For valid Natural parameters, refer to the *Parameter Reference*. When you specify more than one parameter, you have to separate the parameters with blanks. Instead of parameters, it is also possible to specify the name of a Natural parameter file.
`NATRTSVC SHOW`	Displays the startup settings and the stored Natural parameters for all templates.
`NATRTSVC SHOW` *`template-name`*	Displays the startup settings and the stored Natural parameters for the specified template.

Command	Description
`NATRTSVC STATUS`	Displays the status of all templates, that is: whether these templates are active or not active.
`NATRTSVC STATUS` *`template-name`*	Displays the status of the specified template, that is: whether this template is active or not active.

Starting a Natural Process Automatically

This example explains how to install the Natural Runtime startup service, create a new template and start the corresponding Natural process each time the PC is booted.

To start a Natural process when the PC is booted

1 Invoke the **Command Prompt** window of Windows.

2 Go to the Natural *bin* directory which contains the file *natrtsvc.exe*.

3 Enter the following command to install the Natural Runtime startup service:

```
NATRTSVC INSTALL automatic
```

From now on, the Natural Runtime startup service will be started automatically each time the PC is booted.

4 Enter the following command to create an empty parameter template with the name "exa_temp":

```
NATRTSVC CREATE exa_temp
```

The following information is shown:

```
%NATRTSVC-I: Natural n.n Startup Service (1)
%NATRTSVC-I: New Natural instance 'exa_temp' created
%NATRTSVC-I: Natural n.n Startup Service (1)
```

5 Enter the following command to define that a Natural Runtime process with the Natural parameters stored in the parameter template "exa_temp" is to be started when the Natural Runtime startup service is started:

```
NATRTSVC SET exa_temp start=yes
```

The following information is shown:

```
%NATRTSVC-I: Natural n.n Startup Service (1)
%NATRTSVC-I: Configuration successfully set
%NATRTSVC-I: Natural n.n Startup Service (1)
```

6 Enter the following command to store the contents of the Natural parameter file "myparm" in the parameter template "exa_temp":

```
NATRTSVC SET exa_temp parm=myparm
```

The following information is shown:

```
%NATRTSVC-I: Natural n.n Startup Service (1)
%NATRTSVC-I: Configuration successfully set
%NATRTSVC-I: Natural n.n Startup Service (1)
```

7 Reboot your PC.

Since you have defined the automatic startup mode for the Natural Runtime startup service, the defined Natural Runtime processes are started automatically after Windows has been started.

8 Enter the following command to display the status of all parameter templates that are currently defined:

```
NATRTSVC STATUS
```

The following information is shown:

```
%NATRTSVC-I: Natural n.n Startup Service (1)
%NATRTSVC-I: Send request to Natural n.n Startup Service (1)
%NATRTSVC-I:    exa_temp is active
%NATRTSVC-I: Natural n.n Startup Service (1)
```

13 Support of Different Character Sets with NATCONV.INI

- Why Support Different Character Sets 124
- How to Use Different Character Sets 124

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:

- 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*).

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.

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. If you choose a different character set, subsections for this character set must be contained in the sections described below.
`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: ■ the terminal command `%U`, ■ the field attribute `AD=T`,

Section	Description
	■ the statement `EXAMINE TRANSLATE`. 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: ■ `[ISO8859_5->UPPER]` ■ `[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. 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. **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 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: x*digitdigit* X*digitdigit*
x00-x0f = x00,x01	The number of specified values does not correspond to the number of elements in the key range.

14 Natural Exit Codes

- Natural Startup Errors 128

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 Studio.

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	DCOM 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.
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*.
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.
67	The specified port number is already in use.
68	Invalid syntax ... encountered.
71	Listen on specified port failed.
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...
74	...Natural error message '*`nnnn`*' received during startup...
75	The port number is not specified.
76	Wrong RPC version.
77	Invalid `FDDM` assignment.
78	The specified server session ... is not accessible.
79	The port number ... exceeds upper limit (99999).
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.
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.

Note: In order to receive the return code, you must run *nderun.exe* (as opposed to *naturalr.exe*).

15 Setting Up the Entire System Server Interface

- Prerequisites 132
- Activation 132
- Changing the Database ID for the Entire System Server DDMs 133

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.

II Administrating NaturalX Applications

On Windows platforms, an application consisting of NaturalX classes can be distributed across several processes and machines using DCOM.

This part covers the following topics:

NaturalX Servers
Activation Policies
Registration
Type Information
Configuration Overview
Security with NaturalX
DCOM Configuration on Windows
NaturalX System Registry Entries
Using Statements and Commands in a NaturalX Server Environment

On Windows platforms, a sample application is provided in the library `SYSEXNXX`. For information on how to run this application, see the text `A-README` in the library `SYSEXNXX`.

See also *NaturalX* in the *Programming Guide*.

16 NaturalX Servers

- COM Classes and Servers 138
- NaturalX Classes and Servers 138
- NaturalX Servers and Natural Sessions under Windows 138
- The Role of the Server ID 139
- Organizing Server IDs 140

COM Classes and Servers

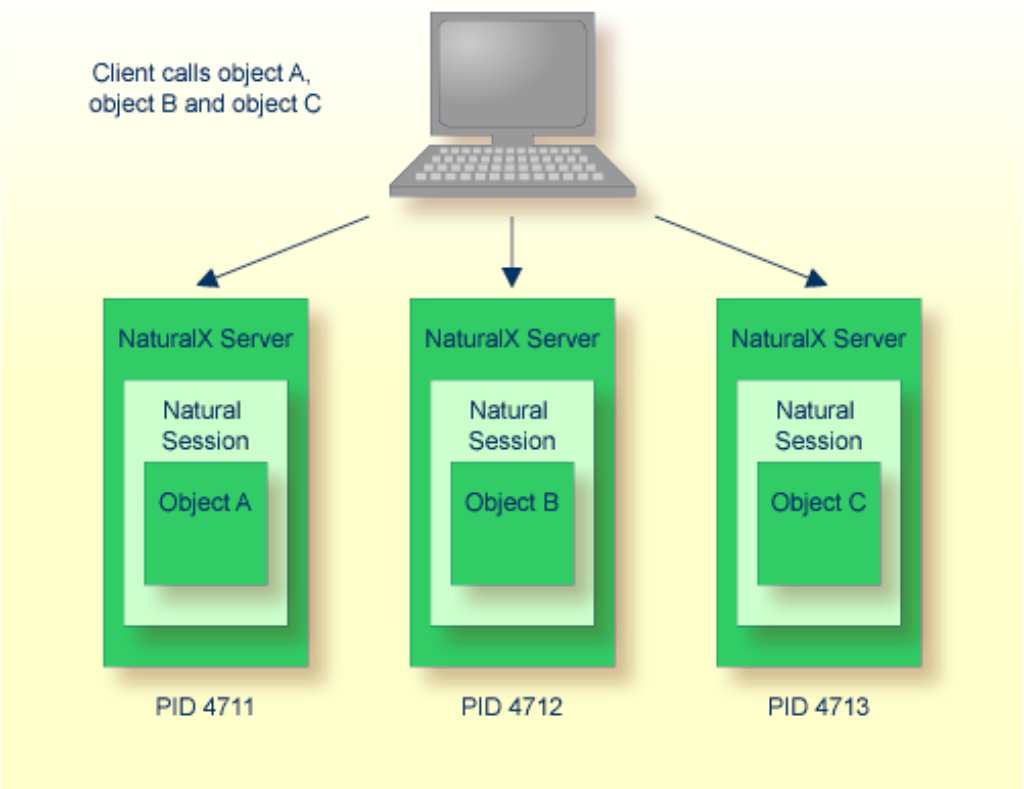
Each COM class must be hosted by a server process. The server process has a number of administrative and technical responsibilities, such as making the class and its interfaces available to DCOM and maintaining the memory occupied by the objects created. Whenever a client requests a new object of a certain class, DCOM checks whether the corresponding server process is already running. If this is not the case, DCOM launches it and passes the request to the server. When the server starts up, it makes its classes available to DCOM. While the server is running, it executes client requests for creation and deletion of objects and execution of methods. When the last object maintained by a server is deleted, the server shuts down automatically. For more detailed information about DCOM classes and servers, please refer to the Microsoft DCOM specification.

NaturalX Classes and Servers

Classes implemented with Natural can be made accessible as DCOM classes. But with Natural, it is not necessary to implement DCOM servers to host the classes. Instead, NaturalX itself performs the tasks of a DCOM server. NaturalX acts as a generic DCOM server for all classes written in Natural. The task that remains for a Natural class developer is just to implement the classes and to assign them to a NaturalX server.

NaturalX Servers and Natural Sessions under Windows

Under Windows, each Natural session runs in its own exclusive NaturalX server process.



The Role of the Server ID

One of the tasks of a DCOM server is to make its classes available to DCOM during startup. But since NaturalX acts as a generic DCOM server, it has no built-in knowledge about the classes it shall provide. Instead, it finds the list of these classes in the system registry under the key of its server ID. The server ID is a Natural-owned key in the system registry, keeping together all classes that belong to a given NaturalX server. It is an arbitrary alphanumeric string of 32 characters which does not contain blanks and which is not case sensitive.

How does a NaturalX server know under which server ID it is running? The server ID is defined with the Natural parameter `COMSERVERID`. This parameter is either passed to a NaturalX server as a **dynamic** parameter, or it is defined in the Natural parameter file.

How are classes assigned to server IDs? Assume Natural has been started with a certain server ID. Then every class that a user registers during this Natural session is entered into the system registry under the current server ID.

Server IDs provide a means of grouping classes created in Natural and assigning them to different NaturalX server processes. The use of server IDs is, however, not compulsory: if Natural is started without a server ID, all Natural classes are registered under the predefined server ID "Default".

Example

Consider the example Employees application consisting of the classes `DepartmentList`, `EmployeesList` and `Employee` (this application is contained in the example library `SYSEXCOM`). These three classes are to be hosted by a NaturalX server called Employees.

1. Start Natural with the desired server ID.
2. Logon to the library `SYSEXCOM`.

```
LOGON SYSEXCOM
```

3. Register the classes with the `REGISTER` command on the Natural command line.

```
REGISTER *
```

The three classes are now registered under the server ID "Employees".

Whenever an object of one of these classes is requested, DCOM will start a NaturalX server process with the server ID "Employees", which will then provide the classes.

Organizing Server IDs

The server ID represents the set of all classes that are made available to DCOM when the corresponding NaturalX server is started. It is recommended that you group under one server ID those classes that form an application from the business point of view, or that otherwise belong together logically. Similarly, classes that are never used in the same context should be registered under different server IDs. Another criterion for the assignment of classes to server IDs is security (see the section ***Security with NaturalX***). From this aspect, it makes sense to group under the same server ID those classes for which common authorizations will be defined.

17 Activation Policies

- Activation Policies on Windows Platforms 142
- Setting Activation Policies 142
- When to Use Which Activation Policy 143

Activation Policies on Windows Platforms

If a client makes a request to create an object of a certain class, it is DCOM's task to start a server process that provides the class and to direct the request to this process. For Natural classes, the responsible server process is a NaturalX server. DCOM recognizes different options that control when a new server process is started or when an object is created in a server process that is already running. For further information, see the section ***Registration***. While registering a Natural class with the `REGISTER` command, you can control which activation options DCOM shall use for this class. NaturalX combines the different options supported by DCOM in the form of the following three activation policies:

- **ExternalMultiple**
  If a Natural class is registered with the activation policy "ExternalMultiple", and a client requests an object of that class, DCOM tries first to create the requested object in the current process. Remember that the client itself might at the same time be a NaturalX server and might provide the class itself. If the current process is not a server for the class, DCOM starts a new NaturalX server process and creates the object in that process. If a second object of the same class is created later, this object is also created in that server process. This means that the same server process can contain several objects of the class.
- **ExternalSingle**
  If a Natural class is registered with the activation policy "ExternalSingle", DCOM starts a new NaturalX server process each time an object of this class is created. One server process can contain only one object of the class.
- **InternalMultiple**
  If a Natural class is registered with the activation policy "InternalMultiple", DCOM always creates objects of this class in the current process. The same server process can contain several objects of the class.

The default activation policy is "ExternalMultiple". This default is defined with the Natural parameter `ACTPOLICY` and can be changed with the Configuration Utility.

Setting Activation Policies

The activation policy of a class can be set in three different ways, in the following order of precedence:

- Explicity as part of the `REGISTER` command.
- In the `DEFINE CLASS` statement.
- With the profile parameter `ACTPOLICY`.

When to Use Which Activation Policy

Non-trivial DCOM applications will mostly deal with “persistent” objects, i.e. objects stored in databases. For such applications, some considerations concerning database access, transaction handling and user isolation must be made. Consider the following scenario: clients A and B both create an object of a class that is provided by a certain NaturalX server process. Assume that the NaturalX server uses a database to load and store its objects. If both clients were served by the same server process, they would appear to the database as one single user. This would have the consequence that a transaction started by a method call from Client A can be committed or backed out by a method call from Client B. Such interferences are obviously to be avoided.

There are two approaches to avoid this interference: either the clients do not use persistent objects, or each of them is served by its own NaturalX server process. Both approaches have their advantages in different situations; for a class or application that does not access databases or other shared resources, it is useful to serve several clients with a single server process. For classes that access databases or other shared resources, it is necessary to isolate different clients in different server processes. Hence both approaches should be possible. Activation policies give an administrator the means to control the activation behavior for each class at registration time.

Example for the Activation Policies

This example illustrates how the various activation policies can be used. Let us consider parts of an imaginary travel agency application. The application contains the business classes `Trip`, `Skipper` and `RoutePlanner`. The `Trip` class represents a sailing trip to be planned; the `Skipper` class represents the skippers available to lead the trips. `RoutePlanner` is a class that determines an optimal route for a trip. Assume that the `Trip` and `Skipper` classes use a database to read and store their objects. The `RoutePlanner` class just performs some calculations on a given `Trip` object and does not use a database.

Since some of the business classes use transactional access to a database, and a transaction might span several method calls, each active client needs to be served with its own NaturalX server process. This can be done by defining an additional class `SagTours`, which represents an application session. This class can be used, for example, to keep general information about the session status, but the main task will be to create business objects on behalf of a client.

Class SagTours

```
* Represents a SagTours application session.
  *
  define class SagTours
    local using tour-ids
    id clsid-sagtours
  *
    interface Create  /* Used to create application objects. */
      id iid-sagtours-create
  *
      method newTrip  /* Creates a new Trip object. */
        is trip-n
        parameter
        1 trip handle of object by value result
      end-method


      method newSkipper   /* Creates a new Skipper object. */
        is skip-n
        parameter
        1 skipper handle of object by value result
      end-method
  *
  end-interface
  *
  end-class
  end
```

This class will be registered as "ExternalSingle". This means that each creation of a `SagTours` object starts a NaturalX server process for the client that requested the object. A client will create a `SagTours` object only once and will use its methods later to create the business objects it needs. In order to create a `Trip` object, the client will call the method `newTrip`, which is implemented as follows.

Method newTrip

```
* This method creates a new Trip object.
*
define data parameter
1 trip handle of object by value result
end-define
*
create object trip of class "Trip"
*
end
```

The `Trip` class itself will be registered as "InternalMultiple". This ensures that the `Trip` objects created by the method `newTrip` are created in the NaturalX server process just started for this client.

Now let us look at the class `RoutePlanner`.

Class RoutePlanner

```
* Plans optimal routes for sailing trips.
  *
  define class RoutePlanner
    local using tour-ids
    id clsid-planner
  *
    interface routing
      id iid-planner-routing
  *
      method plan   /* Plans a sailing trip. */
        is plan-n
        parameter
        1 trip handle of object by value
      end-method
  *
    end-interface
  *
  end-class
  end
```

Method plan

```
* This method plans a sailing trip.
*
define data parameter
1 trip handle of object by value
end-define
*
* Perform some operations on the given Trip object.
*
end
```

This class can be registered as "ExternalMultiple". In this case, all `RoutePlanner` objects created by different clients would be created in the same NaturalX server process. This does not do any harm if the methods of this class do not access databases, or if each database transaction is fully contained in a method (i.e. if each method subprogram ends with either a `BACKOUT TRANSACTION` statement or an `END TRANSACTION` statement).

Now let us look at a sample client program.

Sample Client Program

```
define data local
 sagTours handle of object
 trip handle of object
 planner handle of object
end-define
*
* Start the application session.
create object sagTours of "SagTours"
*
* Create a Trip object.
send "newTrip" to sagTours return trip
* Create a RoutePlanner object.
create object planner of "RoutePlanner"
* Plan the trip.
send "plan" to planner with trip
*
end
```

The client first creates a `SagTours` object. This starts a new NaturalX server process exclusively for this client. The client then uses the `SagTours` object to create a `Trip` object in the context of this application session. Note that the client creates the `RoutePlanner` object directly. This is possible because the class is registered as "ExternalMultiple", but it is not necessary: the `SagTours` class could also provide a method for the creation of `RoutePlanner` objects. Afterwards it lets the business objects do their jobs. The objects are automatically released at program end. The deletion of the `SagTours` object causes the NaturalX server to shut down.

Note: This example shows only the NaturalX techniques needed to illustrate the usage of activation policies. A real-world application would require a lot more. The classes would use object data areas and they would surely have globally unique IDs assigned. Also parameter data areas would be used instead of inline parameter declarations.

18 Registration

- Registration with Natural 148
- Automatic Registration 148
- Manual Registration 149
- Registration Files and Type Library 151
- Client Registration 152
- Registration Hints 153

If a class is to be made accessible to DCOM clients, it is necessary to add some information about the class to the system registry. DCOM clients will mostly address a class with a meaningful name, the so-called programmatic identifier (ProgID) as in the following example:

```
CREATE OBJECT #O1 OF CLASS "Employee"
```

For a Natural class, the class name defined in the `DEFINE CLASS` statement is written into the registry as a ProgID.

System registry entries map this ProgID to the globally unique ID (GUID) of the class, allowing DCOM to uniquely locate all information about the class. Further information that is stored in the registry includes the path and name of the responsible DCOM server, the path and name of the type library, and interface information.

Registration with Natural

Natural classes can be registered (or unregistered) manually with the system command `REGISTER` (or `UNREGISTER`), automatically after the class is stowed (or deleted), or by running the .reg files, which are generated every time a class is registered.

In order to register classes, you must have the rights to modify the system registry and your system environment must be able to use COM.

It is usually not advisable to change the Natural entries in the system registry directly in the registry editor because this can lead to inconsistent registry entries.

A class is always registered for the server ID under which Natural was started.

Automatic Registration

If the profile parameter `AUTOREGISTER` is set to `ON`, a Natural class is automatically registered when it is stowed (cataloged), and unregistered automatically when it is deleted. This means that the user can test the class directly after stowing it.

Automatic registration uses the activation policy setting defined in the `WITH ACTIVATION POLICY` clause of the `DEFINE CLASS` statement of the class. If this clause is not specified, the setting from the profile parameter `ACTPOLICY` is used.

If automatic registration is set and a class is stowed (cataloged), the class is unregistered before it is stowed and registered after the stow has finished so that all old registry entries are removed.

Manual Registration

The following topics are covered below:

- The REGISTER Command
- The UNREGISTER Command

The REGISTER Command

The system command `REGISTER` is used to register Natural classes. They are registered for the server ID under which Natural was started.

```
REGISTER { class-module-name } [ { library-name } [ { ES } ] ]
         {                    }   {             }   { IM }
         { *                  }   { *           }   { EM }
```

class-module-name

This defines which class or classes are to be registered by specifying the appropriate Natural object module name.

library-name

This defines which library or libraries are to be searched for the class or classes.

ES, IM, or EM

This defines the activation policy, which is registered for the class or classes.

You can set one of the following parameters:

Parameter	Description
`ES`	Sets activation policy "ExternalSingle".
`IM`	Sets activation policy "InternalMultiple".
`EM`	Sets activation policy "ExternalMultiple".

The following table shows which classes will be registered for all possible class/library combinations:

Class Module Name Specification	Library Name Specification		
library-name	*	-	
class-module-name	class with class module name *class-module-name* of library *library-name*	all classes with the class module name *class-module-name* which are found in the current step libraries	class with class module name *class-module-name*
*	all classes which are found in the library *library-name* are registered	all classes which are found in the current step libraries are registered	all classes of the current logon library are registered

If this parameter is not specified in the `REGISTER` command or the `DEFINE CLASS` statement, the default activation policy defined in the parameter file is used.

The UNREGISTER Command

The system command `UNREGISTER` is used to unregister Natural classes.

```
UNREGISTER { class-module-name } [ { library-name } [server-id] ]
           { *                 }   { *            }
```

class-module-name

This defines which class or classes are to be unregistered by specifying the appropriate Natural object module name.

library-name

This defines the library or libraries which are to be searched for the class or classes.

server-ID

This defines the server ID of the class or classes.

The following table shows which classes will be unregistered for all possible class/library/server ID combinations:

Class Name Specification	Library Name /Server ID Combination				
	--	*library-name*-	*library-name* *server-ID*	*-	
class-module-name	class with *class-module-name* in the current logon library if it is registered for the current server ID	class with *class-module-name* of library *library-name* if it is registered for the current server ID	class with *class-module-name* of library *library-name* if it is registered for the server *server-ID*	all classes with *class-module-name* found in the current step libraries if they are registered for the current server ID	all nar *cla* fou ste are ser
*	all classes of the current logon library which are registered for the current server ID	all classes found in the library *library-name* which are registered for the current server ID	all classes found in the library *library-name* which are registered for the server *server-ID*	all classes found in the current step libraries which are registered for the current server ID	all cur wh for *ser*

A REGISTER or UNREGISTER system command will return an error message if *class-module-name* or *class-module-name* and *library-name* are specified but either the class or library is not found. If only an asterisk (*) is given in the REGISTER or UNREGISTER system command, no error message is returned if no class has been registered or unregistered.

If a class without class GUIDs or interface GUIDs is specified in the REGISTER system command, an error message will be returned. Such a class can only be used in the local Natural session.

Note: Under Natural Security, this command can only be called for a single library. This means the library name has either to be omitted or a specific library has to be used. It is not possible to use an asterisk (*).

Registration Files and Type Library

Registration files (*.reg* files) enter information in the system registry when they are executed.

Natural will automatically create registration files for the server and the client side when a class is registered.

The server *.reg* file contains the same information that was entered in the system registry and the client *.reg* file contains all information, which is generated for the client side. When a class is unregistered, the *.reg* files will be deleted. If a *.reg* file is not to be deleted with the unregistration, the file has to be renamed before unregistering the class because Natural deletes only files with the default *.reg* file names.

The *.reg* files will be named *classmodule_name_S.reg* (for the server) and *classmodule_name_C.reg* (for the client) and, to activate a different version, *classmodule_name_V.reg*.

A type library is created automatically when a class is registered, and it is deleted when a class is unregistered. A reference to the type library is also entered in the registry.

The default type library name is `classmodule_name`.*tlb*. A new name will be generated if a type library with this name exists already.

The registration files and the type library are stored in the Natural *etc* directory as follows:

```
<install-dir>/etc/serverid/classname/v<version-number>
```

Example

The files for version one of a class `MY.TEST.CLASS` registered for the server ID `SERVER01` are located as follows:

```
<install-dir>/etc/SERVER01/MY.TEST.CLASS/v1
```

Client Registration

Natural does not enter the registration information for the clients automatically in the system registry, but creates a registration file for the client. The client registration file contains an entry (`RemoteServerName`) that tells DCOM on which machine the DCOM server class can be found. This entry is not filled from Natural. It can be entered in either of two ways:

1. The `RemoteServerName` can be entered in the registration files. In this case the line

   ```
   "RemoteServerName"=
   ```

   has to be changed to

   ```
   "RemoteServerName"="server_machine_name"
   ```

   After this, the registration file has to be executed on the client machine.
2. The registration file is executed first, and then the `RemoteServerName` is changed using the `DCOMCNFG` tool or the Registry Editor (see the section ***DCOM Configuration on Windows***).

Registration Hints

The following points should be taken into account when registering and unregistering classes:

- The class GUID should never be changed for an existing class: Natural displays an error message if a class that is already found in the registry is registered again with another GUID. The old class must first be unregistered in this case.
- The same class should never be registered for more than one server ID: there is a one-to-one relationship between the server ID and the AppID, and a class has only one AppID defined, which means that a registration for a second server ID overwrites the AppID. Furthermore, if the class is unregistered for one server ID, all entries of the class are removed without checking whether it is registered for a second server.
- Except for client registration, you should always use the Natural system commands `REGISTER` and `UNREGISTER` to change registry entries for a class because they remove redundant registry entries.

  For example, if a client class has been registered for "server1" and a server registration file with a registration of the same class for "server2" is run, the AppID key of the class is changed and all references to the old AppID key are lost. So this old AppID key can never be deleted. When a class is registered with the system command `REGISTER`, a check is made to see whether the AppID has been changed, and the old AppID is removed if no other class needs it.
- If Natural is not available on the client machine and registry entries for a Natural class are to be removed from the system registry, you should do this with the registry editor. If Natural is available on the client machine, it is easier to register the class first with the Natural system command `REGISTER` and unregister it afterwards with the system command `UNREGISTER`.
- The registration information for a class is taken from the catalogued class object, so that it is not possible to register or unregister a class that is only available in source format.
- If you want to register classes during a Natural session, the session must be started with the parameters `PARM` and `COMSERVERID` only as shown below. This is because only these two parameters are stored in the registry key "LocalServer32". If a class is tested with other parameter settings, there is no guarantee that it will run later when it is started from a DCOM client.

  ```
  NATURAL.EXE PARM=COMPARM COMSERVERID=SERVER1
  ```

- Usually only users with administrator rights can change the system registry. So if you receive an error when trying to register a class, check to see whether you have the rights required to change the registry.
- When a Natural class is registered, some additional information is entered in the registry that is only needed by Natural (not by DCOM). The information which is stored in the additional registry keys is the server ID (see section ***NaturalX Servers***), the activation policy (see section ***Activation Policies***) and the location (Natural class module name and library of class) of the

class. This information is necessary, for example, if all classes of a specified server ID are to be unregistered or to make the served classes available when Natural is started.

- There is a one-to-one relationship between the server ID and the AppID (under *HKEY_CLASSES_ROOT/AppID*) of a class. When a class is registered for a new server ID, a new GUID - the AppID, is generated and assigned to this server ID. The AppID is used by DCOM to group the DCOM classes. Security settings and (for client registrations) the remote machine name are defined for an AppID, i.e. all classes, which belong to one AppID, have the same security settings (see the sections ***Configuration Overview*** and ***Security with NaturalX***).

19 Type Information

- About Type Information 156
- NaturalX and Type Information 156
- Using Type Information 156

About Type Information

Type information is a means to completely describe a class along with all of its interfaces, down to the names and types of the methods. It contains the necessary information about classes and their interfaces, for example, which interfaces exist on which classes, which member functions exist in those interfaces, and which argument those functions require.

This information is used by clients to find out details about a class and its methods, for example, by type-information browsers to present available objects, interfaces, methods and properties to an end user.

Another important area for using type information is the widely-used OLE automation technique which is also used by NaturalX.

There are several ways to store type information. A common way is generating the type information in type library (.TLB) files.

NaturalX and Type Information

Creating Type Information

For each Natural class, a type library file is created when the class is registered.

The type library is generated in the `<install-dir>`/*etc*/`<serverid>`/`<classname>`/`<version>` directory and connected to the class via an entry in the registry.

The name of the class module is used, and the *.tlb* extension is appended unless the type library file name conflicts with an existing name. Then a number is attached to the class module name.

Using Type Information

Each interface defined in a Natural class is seen by clients as a dynamic interface (also called a "dispatch interface"). Each method of an interface is seen by clients under the name defined in the `METHOD` statement.

The first interface in a Natural class is marked as the default dispatch interface.

The support of type information also makes it possible to define multiple interfaces with identical method/property names. The Natural client simply addresses the corresponding method by using the interface name (as defined in the Natural class) as the prefix of the method name, as shown in the following example:

```
CREATE OBJECT #O3 OF CLASS "DepartmentList"
SEND "Iterate.PositionTo" TO #O3 WITH "C" RETURN #DEPT
```

Natural clients use type information to find out to which interface a method or property belongs.

Note: Natural clients do not use type information at catalog time to perform syntax checks.

Data Type Conversions

The following topics are covered below:

- Natural Data Formats to OLE Types
- OLE Types to Natural Data Formats

Natural Data Formats to OLE Types

In order to receive data from clients or to pass data to classes written in different programming languages, the Natural data formats are converted to so-called OLE Automation-compatible types. This table shows how COM clients see the method parameters or properties of a Natural class. For example, if a Natural class has a method parameter or a property with the format A, this is seen by a COM client as `VT_BSTR`.

Natural Data Format	Automation-Compatible Type
A	`VT_BSTR`
B1	`VT_UI1`
B2	`VT_UI2`
B4	`VT_UI4`
B*n* (*n* != 1, 2, 4)	`SAFEARRAY` of `VT_UI1`
C	not supported
D	`VT_DATE`
F4	`VT_R4`
F8	`VT_R8`
I1	`VT_I2`
I2	`VT_I2`
I4	`VT_I4`
HANDLE OF GUI	not supported
HANDLE OF OBJECT	`VT_DISPATCH`
L	`VT_BOOL`
N15.4	`VT_CY`
N*n.m* (*n.m* != 15.4)	`VT_R8`
P15.4	`VT_CY`

Natural Data Format	Automation-Compatible Type
P*n.m* (*n.m* != 15.4)	VT_R8
T	VT_DATE
U	VT_BSTR

An array of a given Natural data format is mapped to a SAFEARRAY of the corresponding VT type.

There are, however, some special cases:

- A variable of format B*n* with fixed length, where *n* is not 1, 2 or 4, or an array of such a variable, is mapped to a one-dimensional SAFEARRAY of VT_UI1. This is for compatibility with previous versions of Natural, where large and dynamic variables were not yet supported. Therefore, large binary variables had to be simulated by arrays of variables of type B with fixed length.
- A dynamic variable of format B is mapped to a one-dimensional SAFEARRAY of VT_UI1.
- An array of dynamic variables of format B is mapped to a SAFEARRAY of variants, each containing a one-dimensional SAFEARRAY of VT_UI1.
- Attribute control variables are not mapped. They have no meaning outside of Natural. Variables of format HANDLE OF GUI are also not mapped. There is no corresponding Automation-compatible type. Therefore properties of the formats Attribute control variable or HANDLE OF GUI cannot be accessed by clients through COM/DCOM. Method parameters of these types should be marked as optional in the parameter data area, so that clients can omit the parameters when calling the method through COM/DCOM.

OLE Types to Natural Data Formats

This table shows how parameters or properties of an external class can be addressed by Natural. For example, if an external class has a method parameter or property with type VT_R4, this parameter or property can be addressed in Natural as F4 or with a format that is MOVE-compatible to F4.

Automation -Compatible Type	Natural Data Format
VT_BOOL	L
VT_BSTR	A or U
VT_CY	P15.4
VT_DATE	T
VT_DISPATCH	HANDLE OF OBJECT
VT_UNKNOWN	HANDLE OF OBJECT
VT_I1	I1
VT_I2	I2
VT_I4	I4
VT_INT	I4

Automation -Compatible Type	Natural Data Format
VT_R4	F4
VT_R8	F8
VT_U1	B1
VT_U2	B2
VT_U4	B4
VT_UINT	B4

A SAFEARRAY of up to three dimensions is converted into a Natural array with the same dimension count and the corresponding format. SAFEARRAYs with more than three dimensions cannot be used from within Natural.

There are, however, some special cases:

- A VT_BSTR maps either to a Natural variable of format A or to a one-dimensional array of Natural variables of format A with fixed length. The additional dimension is then used to store strings longer than 253 characters. This is for compatibility with previous versions of Natural, where large and dynamic variables were not yet supported. This mapping should no longer be used. Instead, a dynamic variable of format A should be used.
- A SAFEARRAY of VT_BSTRs maps either to an array of Natural variables of format A with the same dimension count, or to an array of Natural variables of format A with fixed length with one more dimension. The additional dimension is then used to store strings longer than 253 characters. This is for compatibility with previous versions of Natural, where large and dynamic variables were not yet supported. This mapping should no longer be used. Instead an array of dynamic variables of format A should be used.
- A SAFEARRAY of VT_UI1 can be mapped to an array of Natural variables of format B with fixed length that has a matching total size. This is for compatibility with previous versions of Natural, where large and dynamic variables were not yet supported. This mapping should no longer be used. Instead a dynamic variable of format B should be used.

20 Configuration Overview

- Server Configuration - General Settings 162
- Server Configuration - Application-Specific Settings 163
- Client Configuration - General Settings 163
- Client Configuration - Application-Specific Settings 164

Once all classes of an application have been registered on the client and server machines, certain aspects of the application's behavior can be controlled and configured with system registry settings. This section summarizes the relevant registry entries and their meaning for NaturalX applications. For detailed background information about the registry keys and their administration, please refer to the specific DCOM registry documentation of the appropriate platform.

The registry keys relevant in this context are maintained with commonly-used tools like `DCOMCNFG` or the Registry Editor (`REGEDIT`). These tools present the registry keys in a different way. Therefore only the names of the registry keys are mentioned here. The section ***DCOM Configuration on Windows*** describes how to set registry keys.

Note: "HKLM" is the common short form of the registry key *HKEY_LOCAL_MACHINE*, where "HKCR" stands for *HKEY_CLASSES_ROOT*.

Server Configuration - General Settings

This section discusses general server configuration settings.

- The registry entry *HKLM\Software\Microsoft\OLE\EnableDCOM* must be set to "Y" to enable access to the server machine via DCOM.
- If guests (users who do not have their own account on the server machine) are to be able to access applications on the server machine, the predefined account "Guest" must be enabled in the User Manager (Windows 2000 only).
- The registry entries *HKLM\Software\Microsoft\OLE\DefaultLaunchPermissions* and *HKLM\Software\Microsoft\OLE\DefaultAccessPermissions* define which users or groups are allowed or not allowed to launch DCOM applications and to access their classes. The authorizations defined here apply for all applications for which no application-specific settings are defined.
- The registry entry *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the level of authentication that is performed for clients that access DCOM applications on this machine. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least "Connect". Choose "None" if no authentication is to take place. In this case, the NaturalX server does not retrieve the client's user ID. Instead it performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.
- The registry entry *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least "Identify". The settings "Impersonate" or "Delegate" have the same effect for a NaturalX server. Choose "Anonymous", if the server is not to be able to retrieve the client's user ID. In this case, the server performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Server Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. It is important to remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

- The registry entries *HKCR\AppID\<APPID>\LaunchPermission* and *HKCR\AppID\<APPID>\AccessPermission* define which users or groups are allowed or not allowed to launch the DCOM application with the specified AppID and to access its classes.
- The registry entry *HKCR\AppID\<APPID>\RunAs* defines the user account this NaturalX server will run when it is launched by DCOM. There are three options:
  - **Interactive user:**
    The NaturalX server is started under the account of the user that is interactively logged in on the server machine. This is usually not desirable but can be useful for test reasons.
  - **Launching user:**
    The NaturalX server is started under the account of the client that creates the first object on this server (remember that the first request for an object forces DCOM to launch the server). This setting should be used if each client is to be served by its own server process. Obviously, the client must have permission to launch the server.
  - **This user:**
    The server is started under the account of a given user. This setting should be used if all clients are to be served by the same server process. The user entered here must have permission to launch the server.

Client Configuration - General Settings

This section discusses general client configuration settings.

- The registry key *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the degree of authentication that is performed for clients running on this machine when they access DCOM applications. For a client that accesses a NaturalX server, a similar consideration to that in the section ***Server Configuration - General Settings*** applies: only if it specifies at least "Connect", will the NaturalX server be able to use its user ID against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.
- The registry key *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. For a client that accesses a NaturalX server, a similar consideration to that in the section ***Server Configuration - General Settings*** applies: only if

it specifies at least "Identify", will the NaturalX server be able to retrieve its user ID and use it against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Client Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. Remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

The registry key *HKCR\AppID\<APPID>\RemoteServerName* defines on which remote machine DCOM should start the server when a class hosted by this server is requested. If the server is to be started locally, "Run on this computer" and no RemoteServerName must be specified.

21 Security with NaturalX

- About NaturalX Security 166
- Activation Security 166
- Call Security 167

Information on how to configure NaturalX is given in the section *DCOM Configuration on Windows*.

About NaturalX Security

In a distributed environment, security is an especially important topic. A server must be sure that no unauthorized clients use the services it provides. A client must be sure that it is connected to the server it expects, and that the server does not misuse its (the client's) authorizations.

In the context of DCOM, two levels of security can be distinguished:

- Activation security controls who is allowed to launch and access the server process that provides the class.
- Call security controls who is allowed to use the individual methods a class provides.

In many cases, activation security may be sufficient to define authorizations. This security level is supported by DCOM itself on the basis of Windows Security. The necessary authorizations are maintained in the system registry. This is described in the section *Activation Security*.

In other cases it may be necessary to control authorizations in more detail at the level of individual methods. This security level cannot be maintained with registry definitions. It is, therefore, provided by NaturalX with the help of Natural Security. This is described in the section *Call Security*.

Activation Security

This section covers the following topics:

- Applications
- Authorizations using the Registry

Applications

Activation security controls who is allowed to launch and access a server process. In principle, this could be done by defining authorizations for each individual class. For practical reasons, however, and to reduce administration efforts, authorizations are normally maintained at the application level. In the system registry, each application is defined by an AppID. The AppID is the key under which the authorizations for an application are maintained. To maintain these authorizations, each DCOM enabled platform provides the tool `DCOMCNFG`. This tool can be used for NaturalX applications as well as for other DCOM applications.

In order to understand the meaning of AppIDs in NaturalX, recall for a moment how NaturalX organizes classes to applications (see the section *Organizing Server IDs*). With the Natural para-

meter `COMSERVERID`, a name can be given to a certain NaturalX server. When Natural is started with a given value of `COMSERVERID`, all Natural classes that are registered during this Natural session are registered under this server ID. At the same time, they are all registered under the same AppID key in the system registry. This means that each different value of *`server-ID`* corresponds to a different AppID key in the system registry.

As an example, assume Natural is running with the server ID "Employees". All classes registered during this Natural session will then form the "Employees" application. The `REGISTER` command registers them all under one AppID key - the one that corresponds to the "Employees" application.

Authorizations using the Registry

When configuring Activation Security, the following registry keys are of interest: *LaunchPermissions*, *AccessPermissions*, *DefaultLaunchPermissions* and *DefaultAccessPermissions*. The keys *DefaultLaunchPermissions* and *DefaultAccessPermissions* exist only once in the registry and define authorizations for all applications for which no individual authorizations have been defined. The keys *LaunchPermissions* and *AccessPermissions* exist for each application (i.e. for each AppID) and define the authorizations for an individual application.

Call Security

This section covers the following topics:

- Authorizations using Natural Security
- Call Security Hints and Suggestions

Authorizations using Natural Security

Call security is used to control who is allowed to use the individual methods that a class provides. Authorizations on this level cannot be maintained by registry definitions. Call security is therefore provided by NaturalX with the help of Natural Security.

In order to understand how call security is achieved with Natural Security, consider how a class in NaturalX is implemented: each class is a Natural module of type class, each method is a Natural module of type subprogram. For all Natural modules, the execution can be controlled by authorizations defined in Natural Security. Please refer to the *Natural Security* documentation for further information about how to do this.

The authorizations defined for class modules and method subprograms are evaluated whenever a class module is used to create objects and whenever a method subprogram is executed in response to a method call. The following rule applies: a user who is allowed to execute the class module is allowed to create objects of that class, and a user who is allowed to execute a method subprogram is allowed to use the corresponding method.

In order to perform the necessary authorization checks, a NaturalX server must know the client's user ID. It must also be sure that the user ID is authentic. Therefore the following requirements must be met to use call security:

- The client must have identified itself with a logon on its local machine or on a Windows domain server.
- *Authentication level* must be set to at least "Connect" (either on the client or on the server machine).
- *Impersonation level* must be set to at least "Identify" (either on the client or on the server machine).

If the above requirements are met, a NaturalX server that is going to process a request takes the client's user ID and places it into the Natural system variable `*USER`. The request is then performed under this user ID, including all necessary Natural Security authorization checks. After having processed the request, the Natural system variable `*USER` reverts to the value that it had at the startup of the NaturalX server.

If one of the requirements is not met, `*USER` remains unchanged during execution of the request. The request is then executed under the user ID under which the NaturalX server was started.

In addition to `*USER`, also the system variable `*NET-USER` is filled during execution of a request. It contains the user ID qualified with the domain name for clients belonging to a Windows domain and can be used for additional application-specific security checks.

Call Security Hints and Suggestions

The following points should be taken into consideration when using NaturalX with Natural Security:

- In a Natural Security environment, a NaturalX server must be started with the Natural parameter `AUTO=ON`. This is because the authentication already takes place on the client side. The setting should be entered in the Natural parameter file.
- In a Natural Security environment, it is a good idea to let a NaturalX server always start under a specific user ID. This user ID is then automatically used for all requests of unauthenticated users, and it is up to the Natural Security administrator to define minimal authorizations for this user ID.
- Remember that Natural and Natural Security cannot handle user IDs which are longer than 8 characters or which contain blanks.

22 DCOM Configuration on Windows

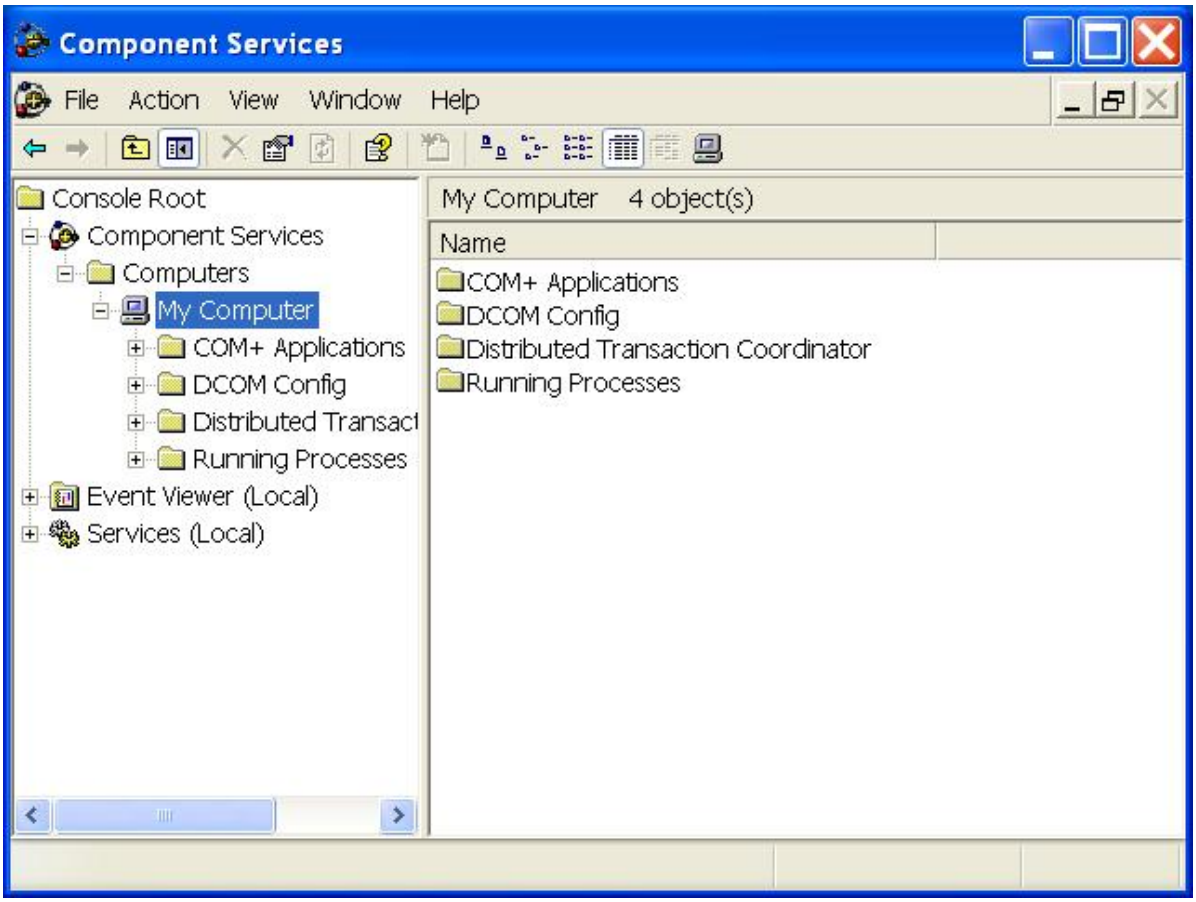
- Configuring NaturalX Servers 170
- Configuring NaturalX Clients 180

This chapter describes how to configure NaturalX applications on Windows. All settings are applied with the tool *DCOMCNFG.EXE* or Component Services. The dialog examples shown in the following sections appear as with Component Services under Windows XP.

Configuring NaturalX Servers

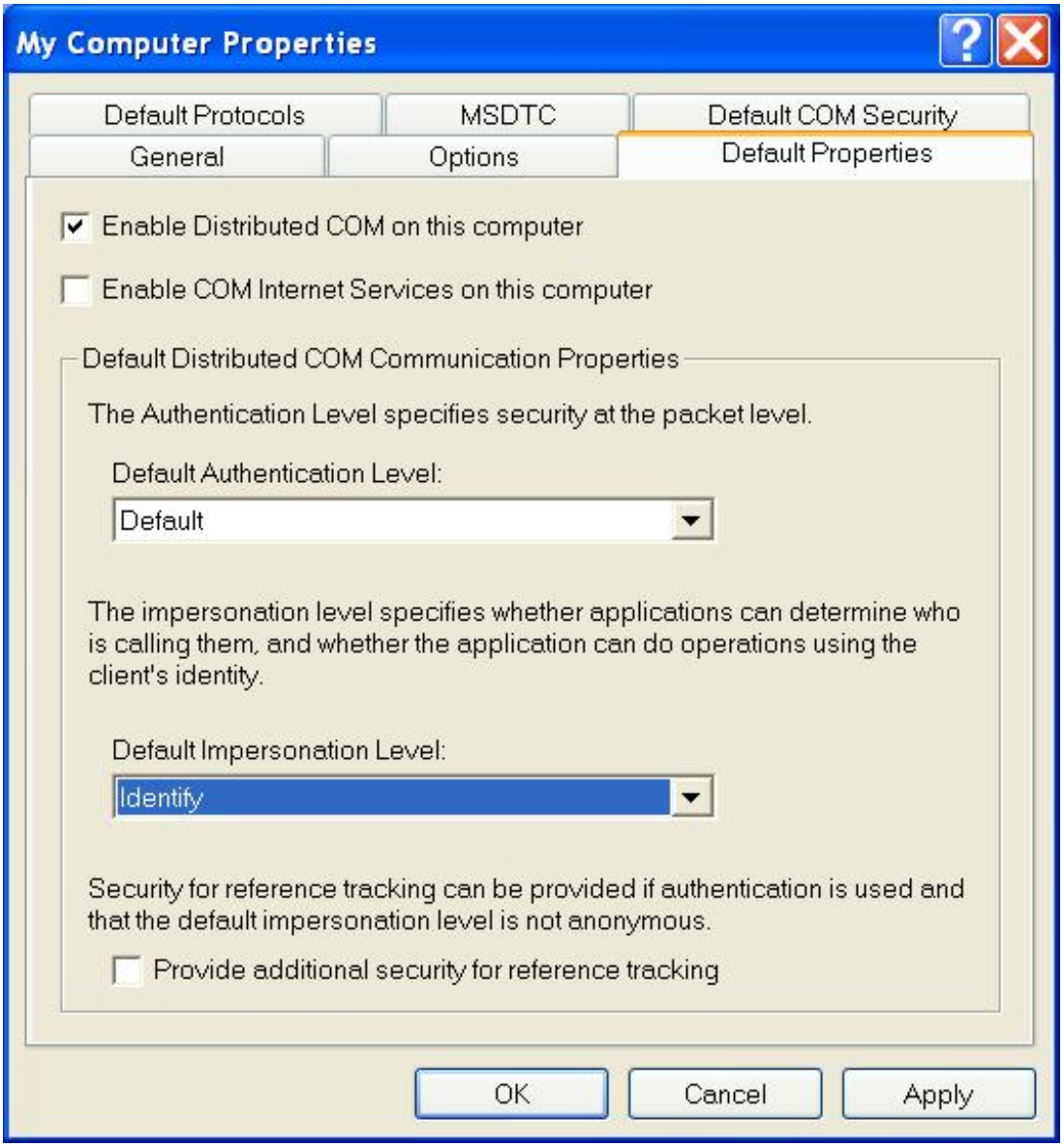
To configure NaturalX servers

1 Invoke Component Services.



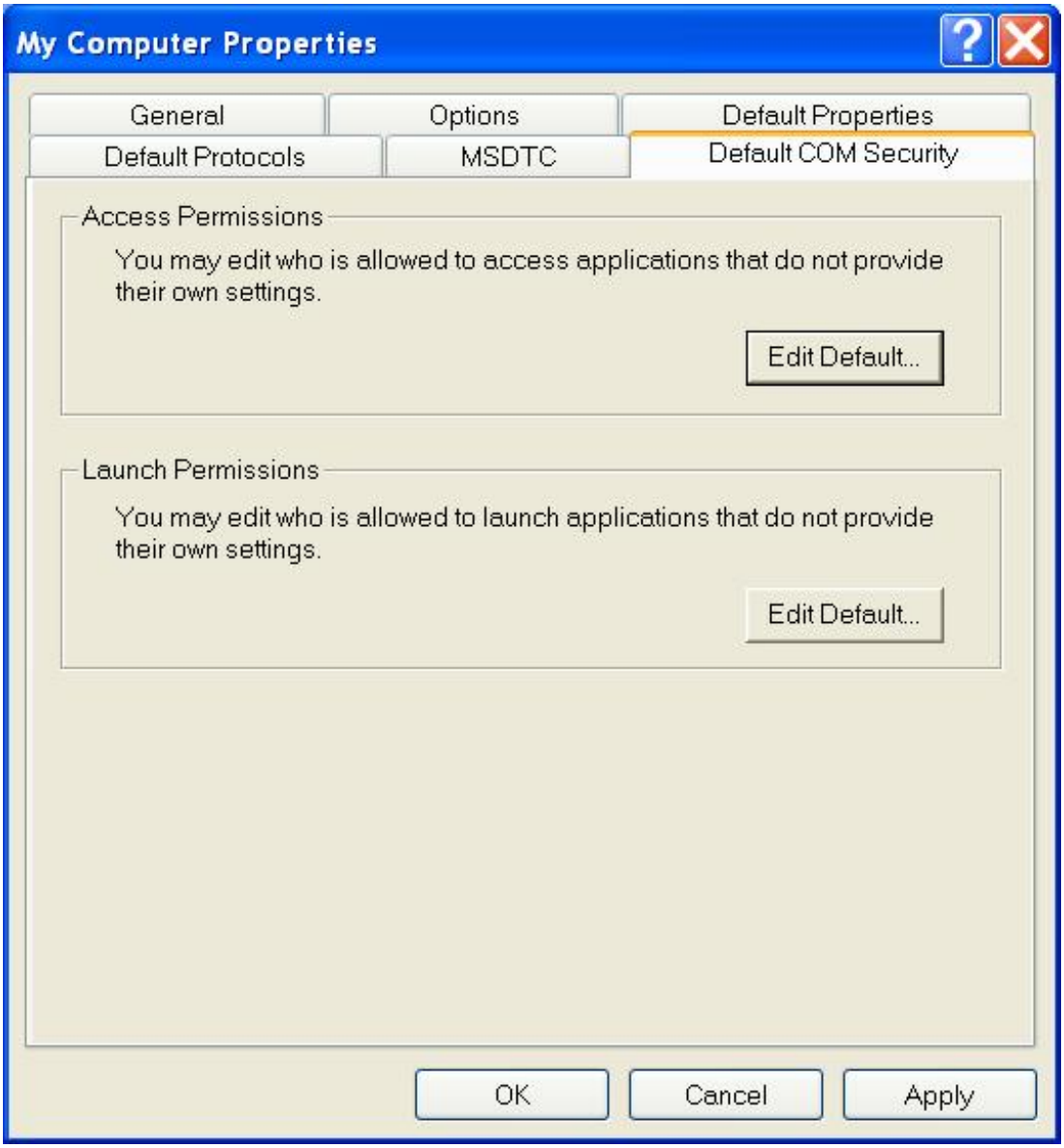
2 In the **Properties** dialog of **My Computer**, select the **Default Properties** tab and activate the check box **Enable Distributed COM on this computer**.

3 Set **Default Authentication Level** to **Default** and **Default Impersonation Level** to **Identify**.



This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable `*USER` in order to let Natural Security checks run against this user ID.

4 Now set up the default security configuration.



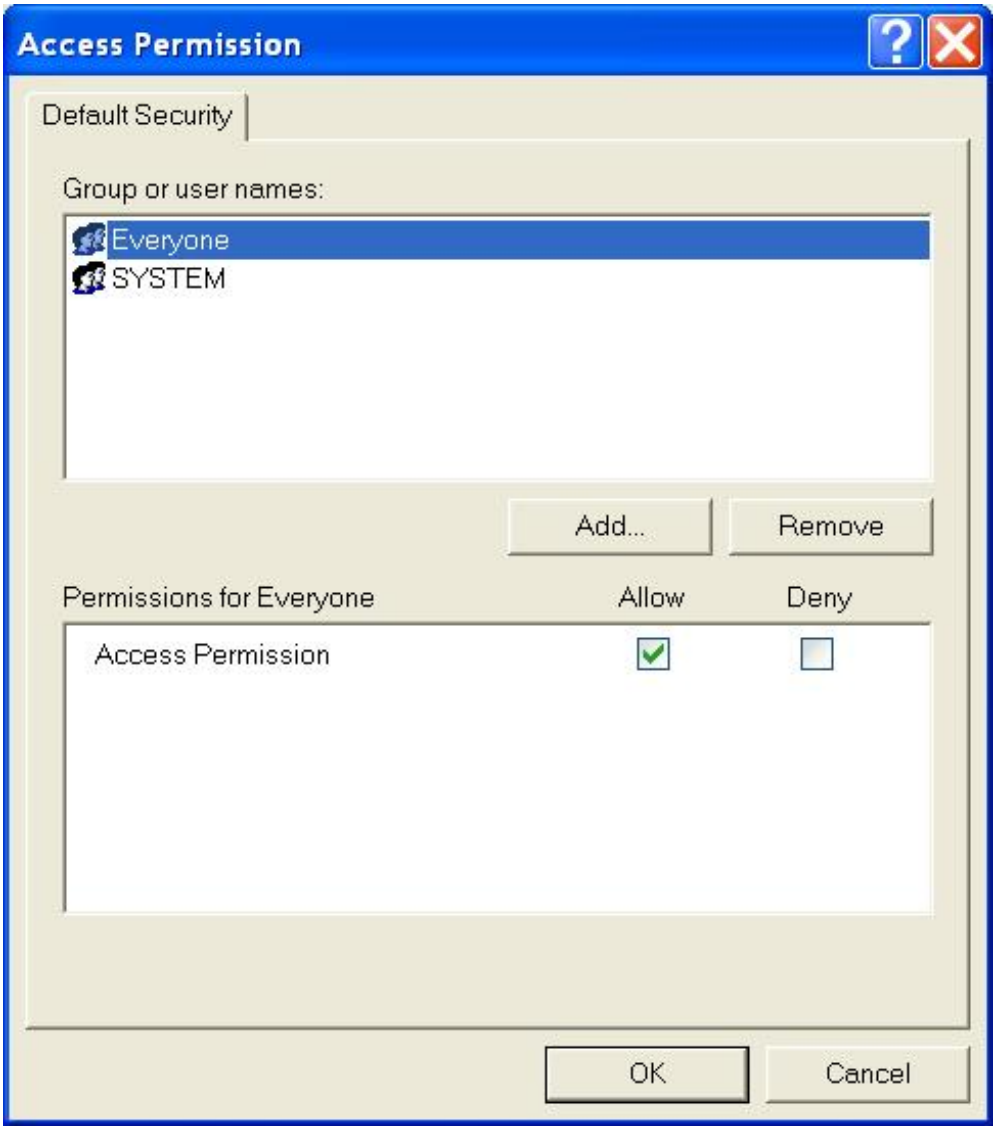
In the **Default COM Security** tab, choose **Edit Default** in the **Access Permissions** group box.

The **Access Permission** dialog box appears.

5 Use the **Add** button to define which users and groups may access NaturalX servers.

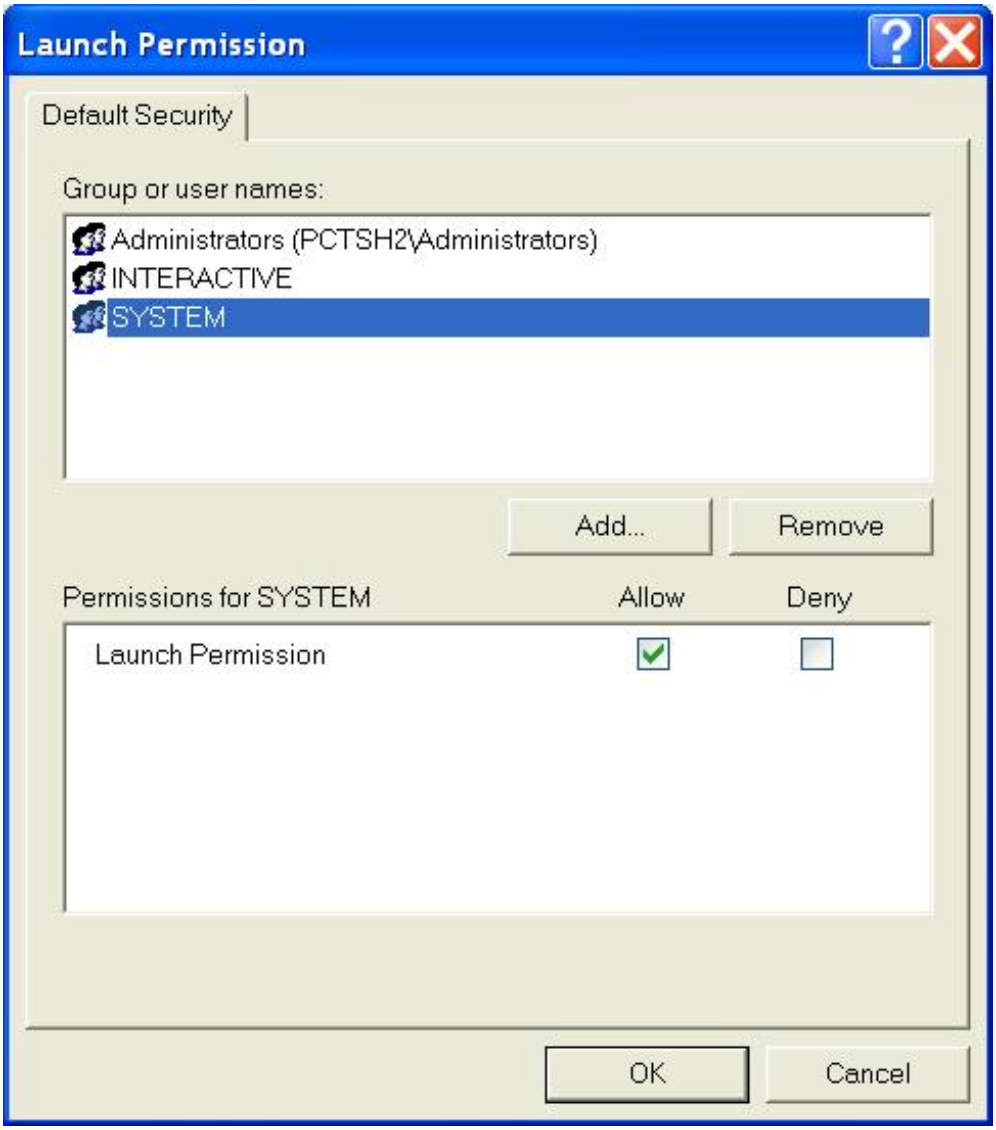
Note: You must allow access at least to the account "SYSTEM".

In most cases you will define a group of all users to whom you want to allow access and enter this group here. In the example, the built-in group "Everyone" is entered. This allows access to every user that is defined on the server machine. If the built-in account "Guest" is enabled in the User Manager, this setting allows access to users not defined on the server machine (guests) as well.



6 In the **Default COM Security** tab, choose **Edit Default** in the **Launch Permission** group box.

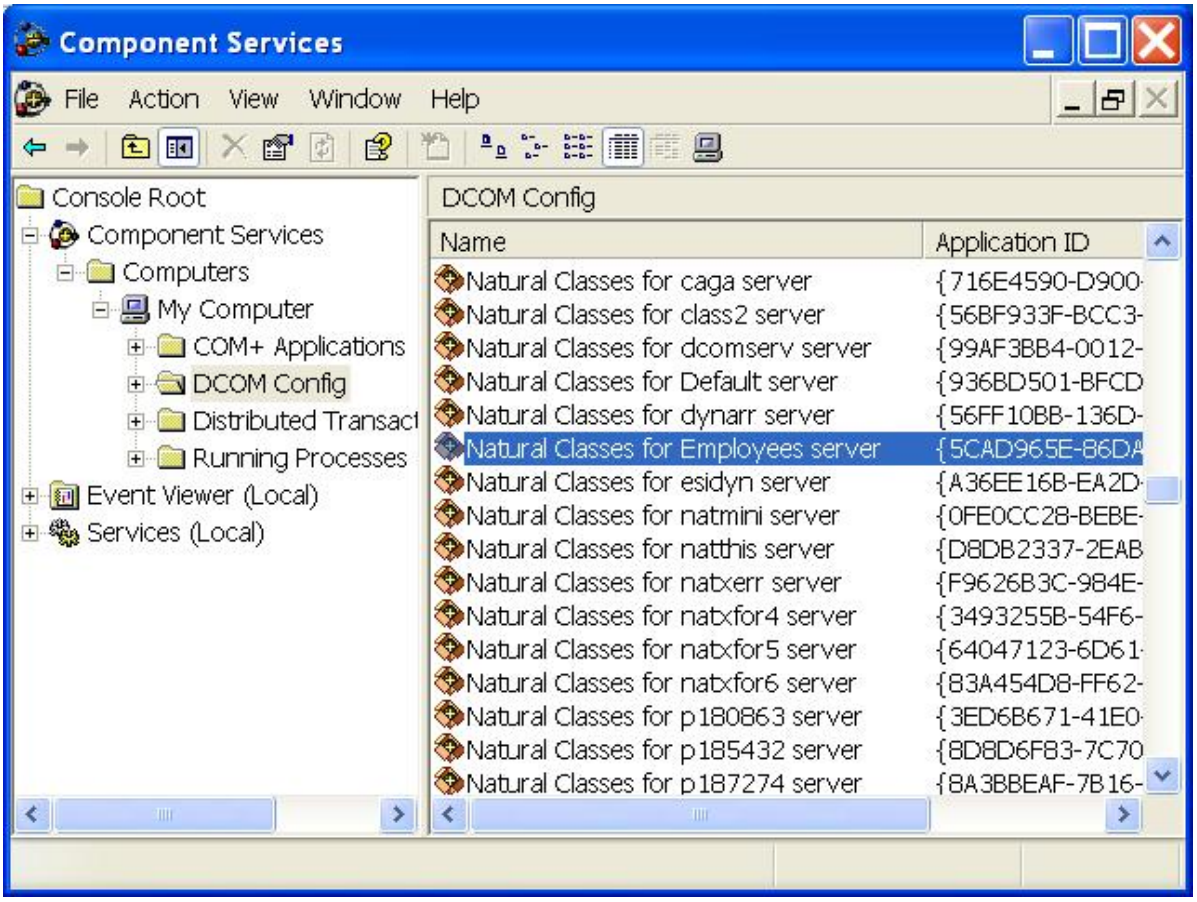
The **Launch Permissions** dialog box appears.



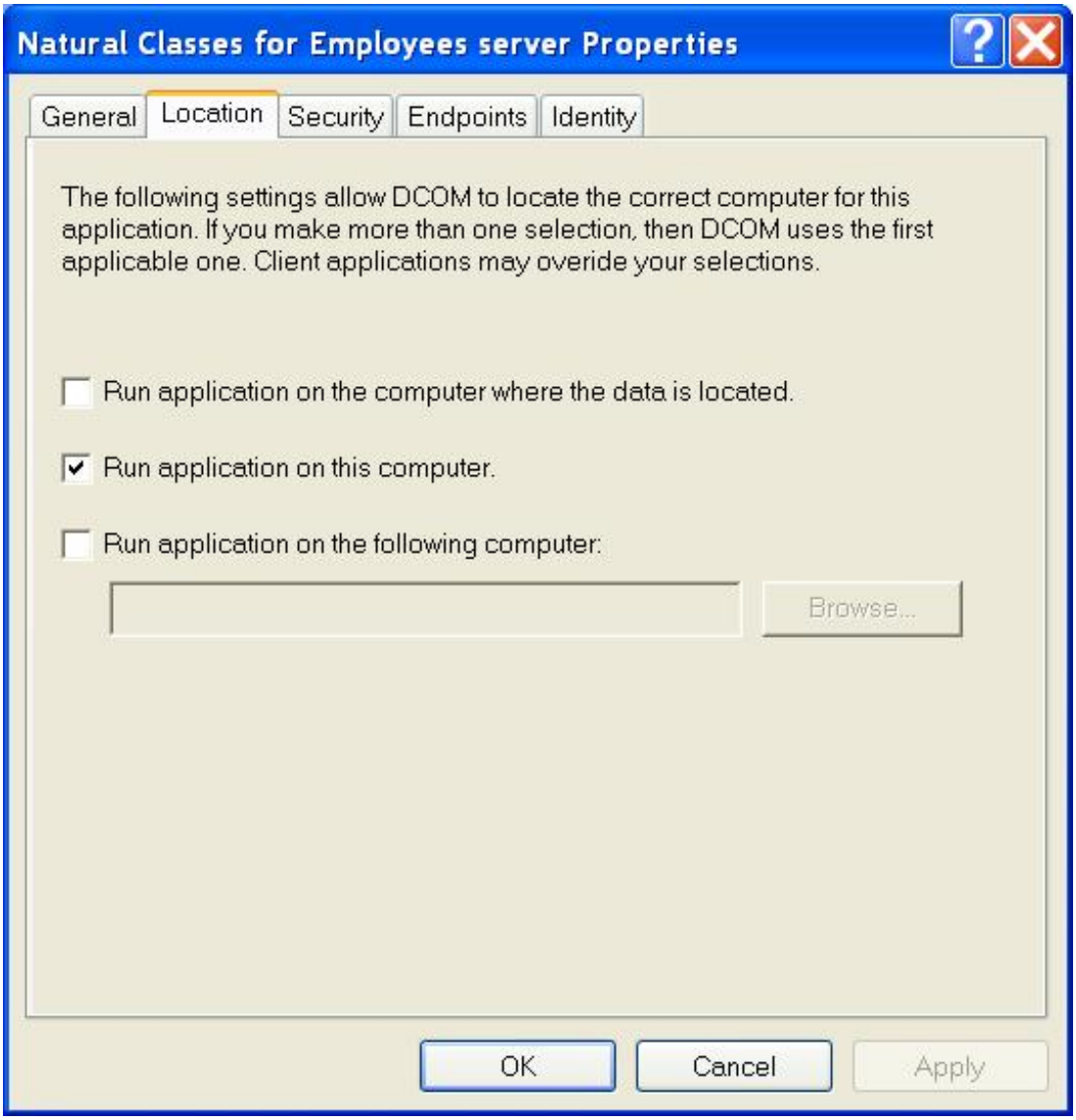
Note: You must allow launch at least to the accounts "SYSTEM" and "INTERACTIVE" and the group "Administrators".

7 Now set up the configuration for a specific NaturalX server. Select the node **DCOM Config** and locate your NaturalX server in the **DCOM Config** list box (in the example "Natural classes for Employees server").

8 Select your server and choose **Properties**.

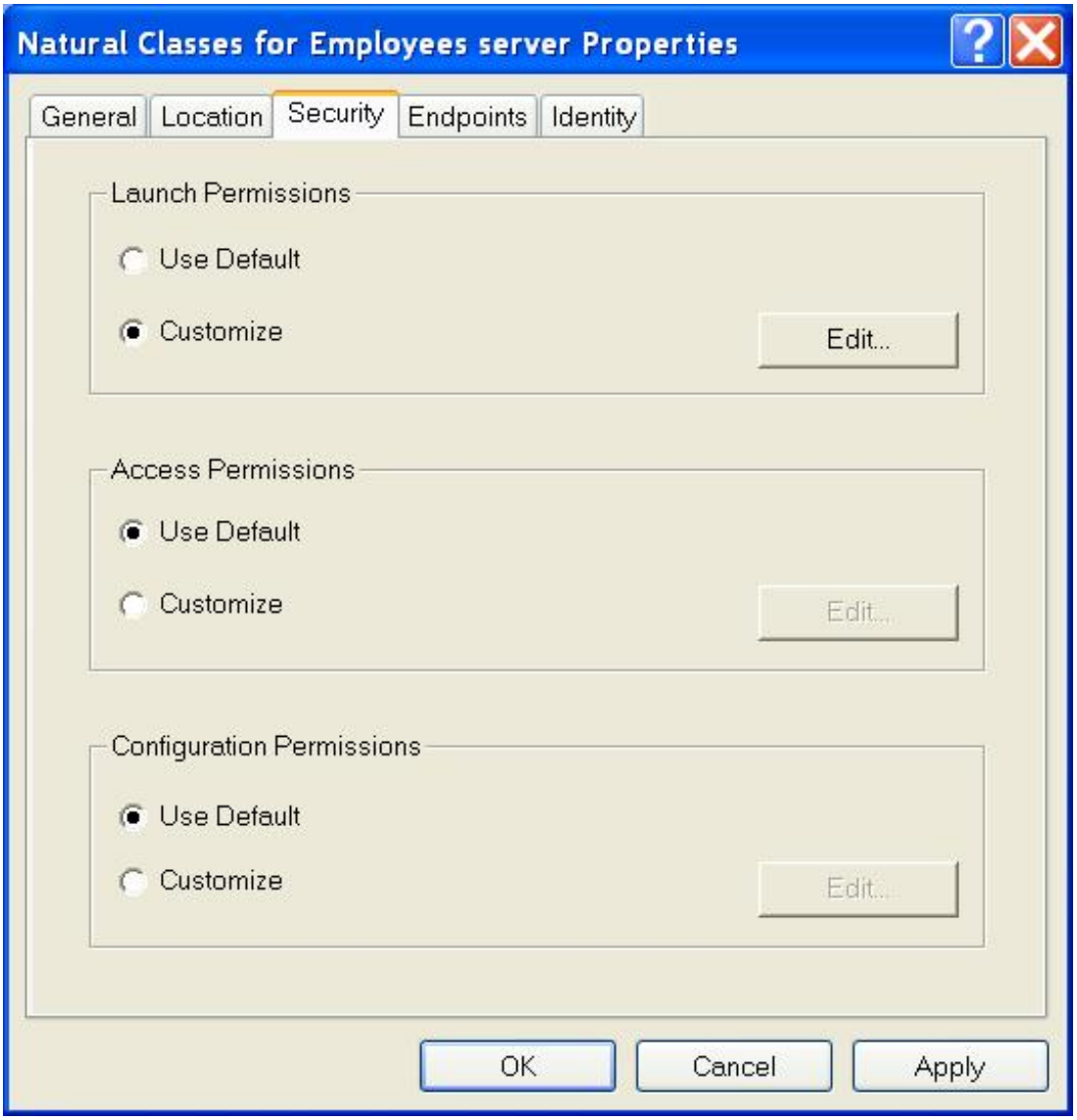


9 In the **Location** tab, activate the check box **Run application on this computer**.



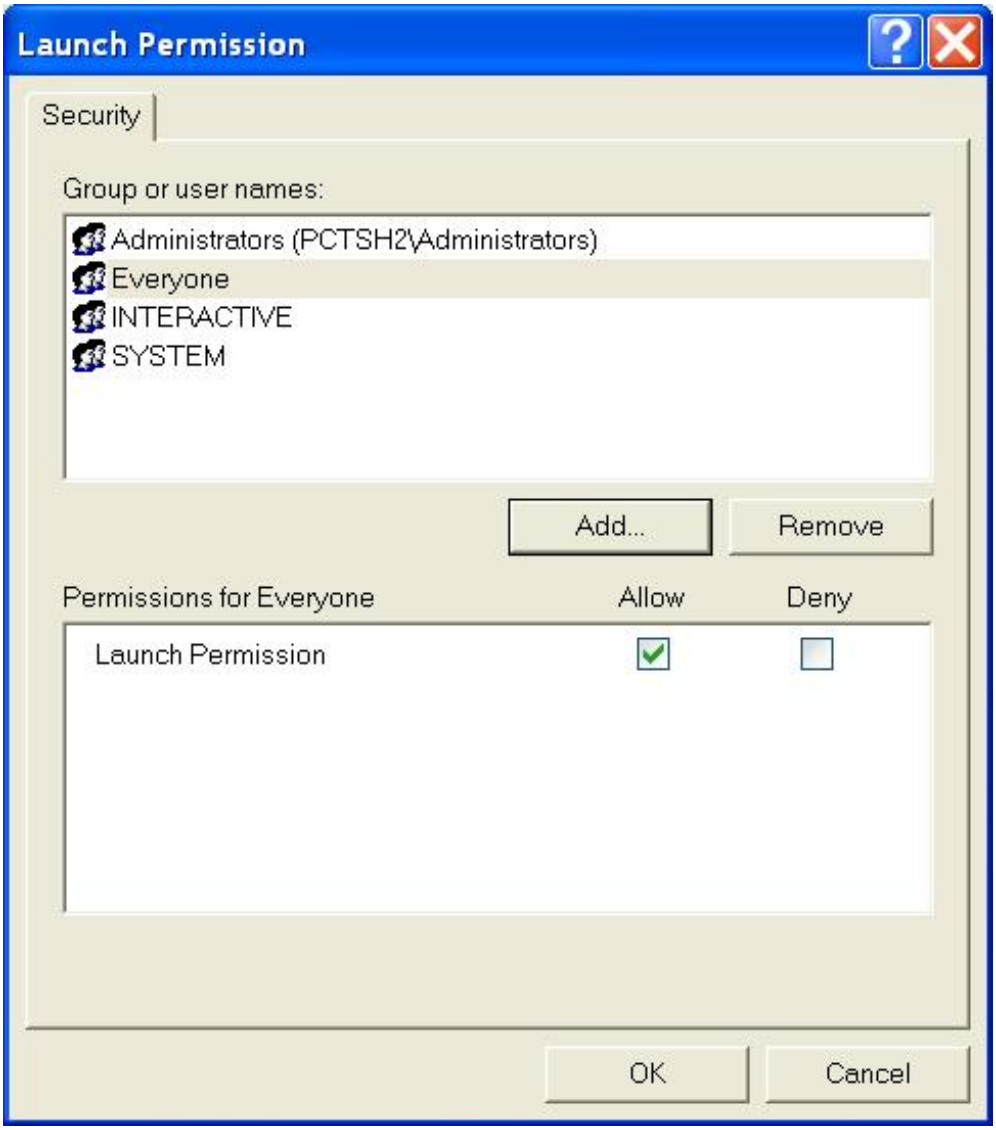
10 In the **Security** tab, make sure that **Access Permissions** is set to **Use Default** and **Launch Permissions** is set to **Customize**.

11 Choose **Edit** in the **Launch Permissions** group box to modify the application-specific launch permissions.



The list **LaunchPermission** will contain at least the accounts "SYSTEM" and "INTERACTIVE" and the group "Administrators".

12 Add the users and groups to be allowed to launch your NaturalX server. In most cases, you will define a group of all users to whom you want to allow launch and enter this group here. In the example, the built-in group "Everyone" is entered. This allows launch to every user that is defined on the server machine. If the built-in account "Guest" is enabled in the User Manager, this setting allows launch to users not defined on the server machine (guests) as well.



13 In the **Identity** tab, define the account under which the NaturalX server will be launched.

- If you select **The launching user**, a server process will be launched for each client. The server process will be launched under the account of the client user.
- If you select **The interactive user**, only one server process will be launched for all clients.

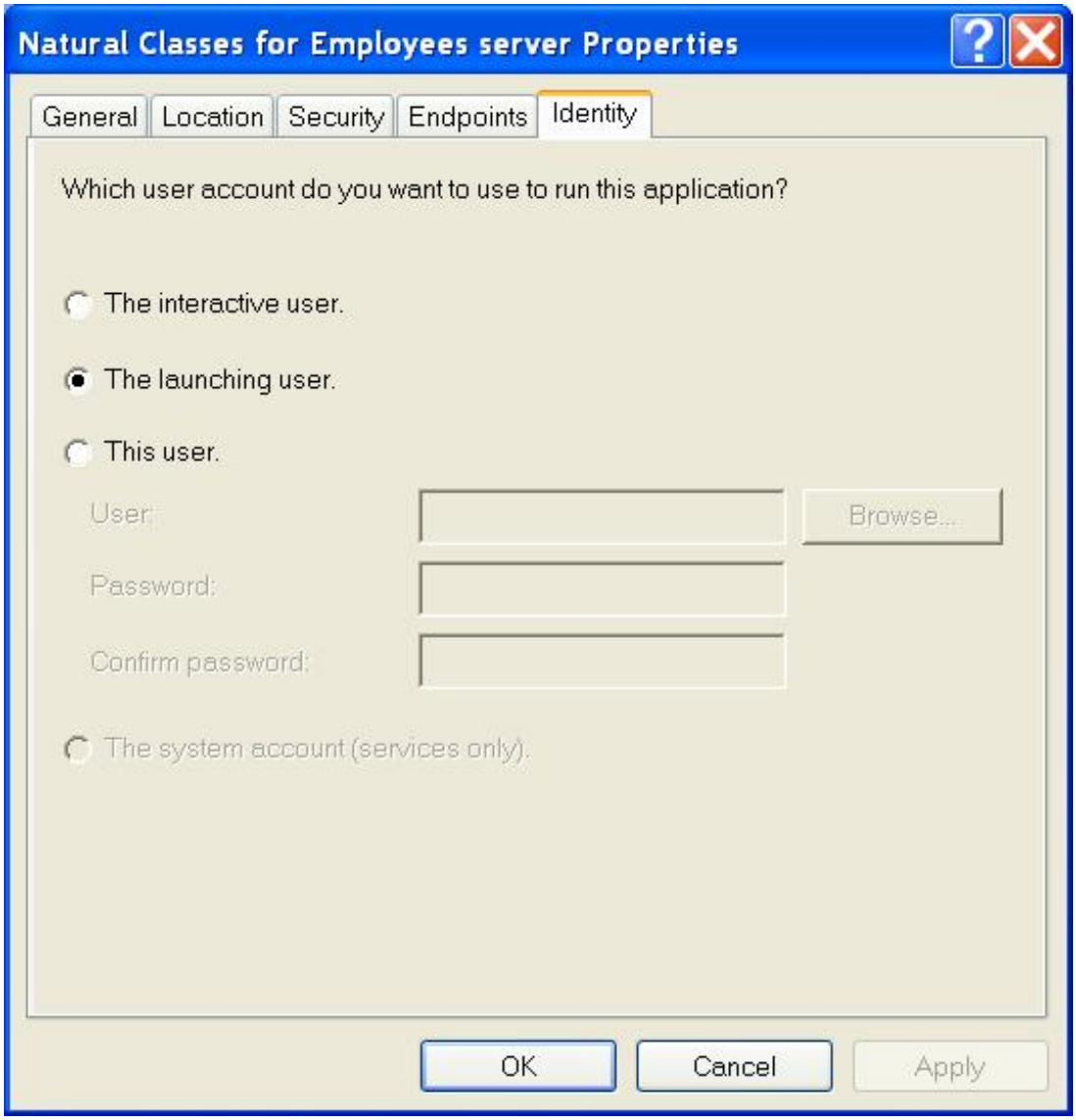
  Note: This is true only for classes that have been registered in Natural as "ExternalMultiple". If a class is registered as "ExternalSingle", a server process is created for each object of this class that is created.

  The server process will be launched under the account of the user that is interactively logged in on the server machine. If no user is currently logged in on the server machine, this setting behaves like **The launching user**.

- If you select **This user** and select a specific user account, only one server process will be launched for all clients.

  Note: This is true only for classes that have been registered in Natural as "ExternalMultiple". If a class is registered as "ExternalSingle", a server process is created for each object of this class that is created.

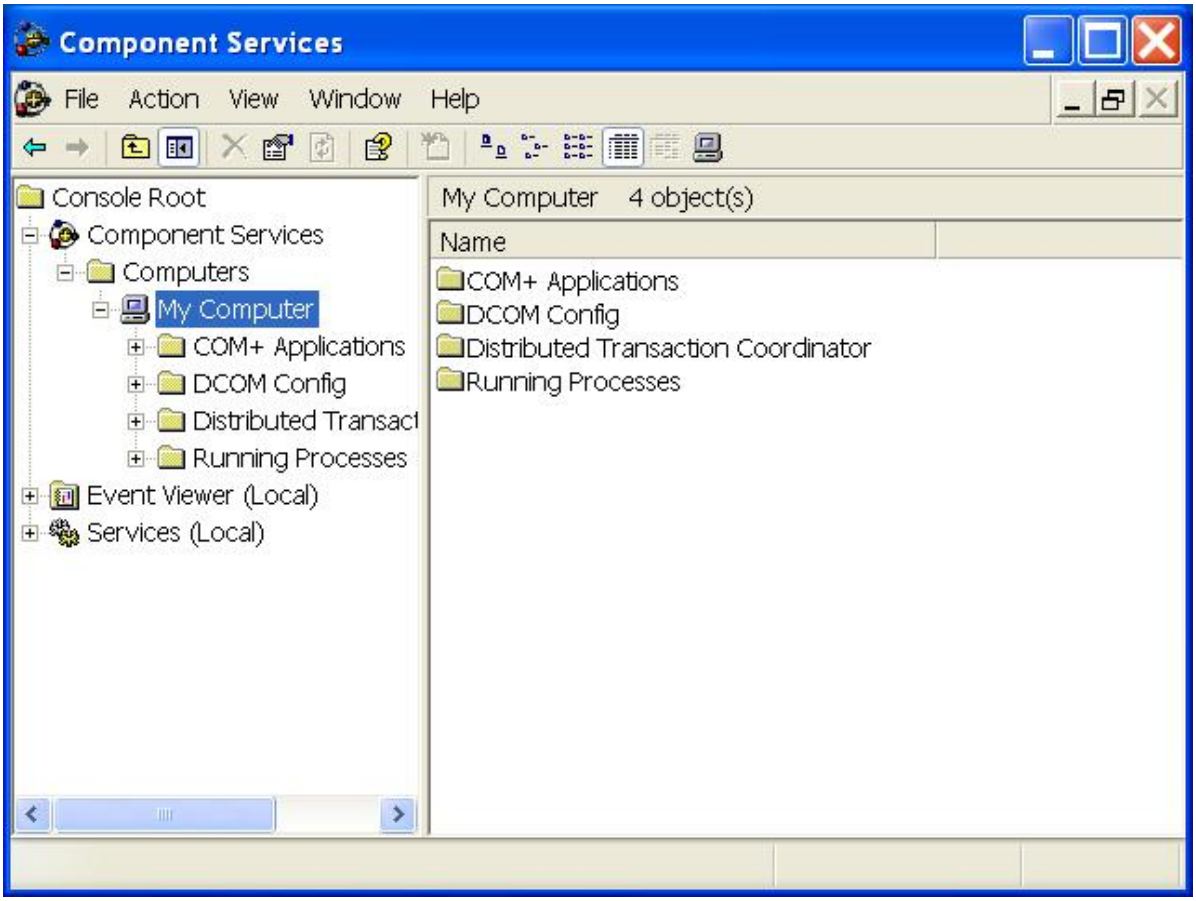
  The server process will be launched under the specified user account.



Configuring NaturalX Clients

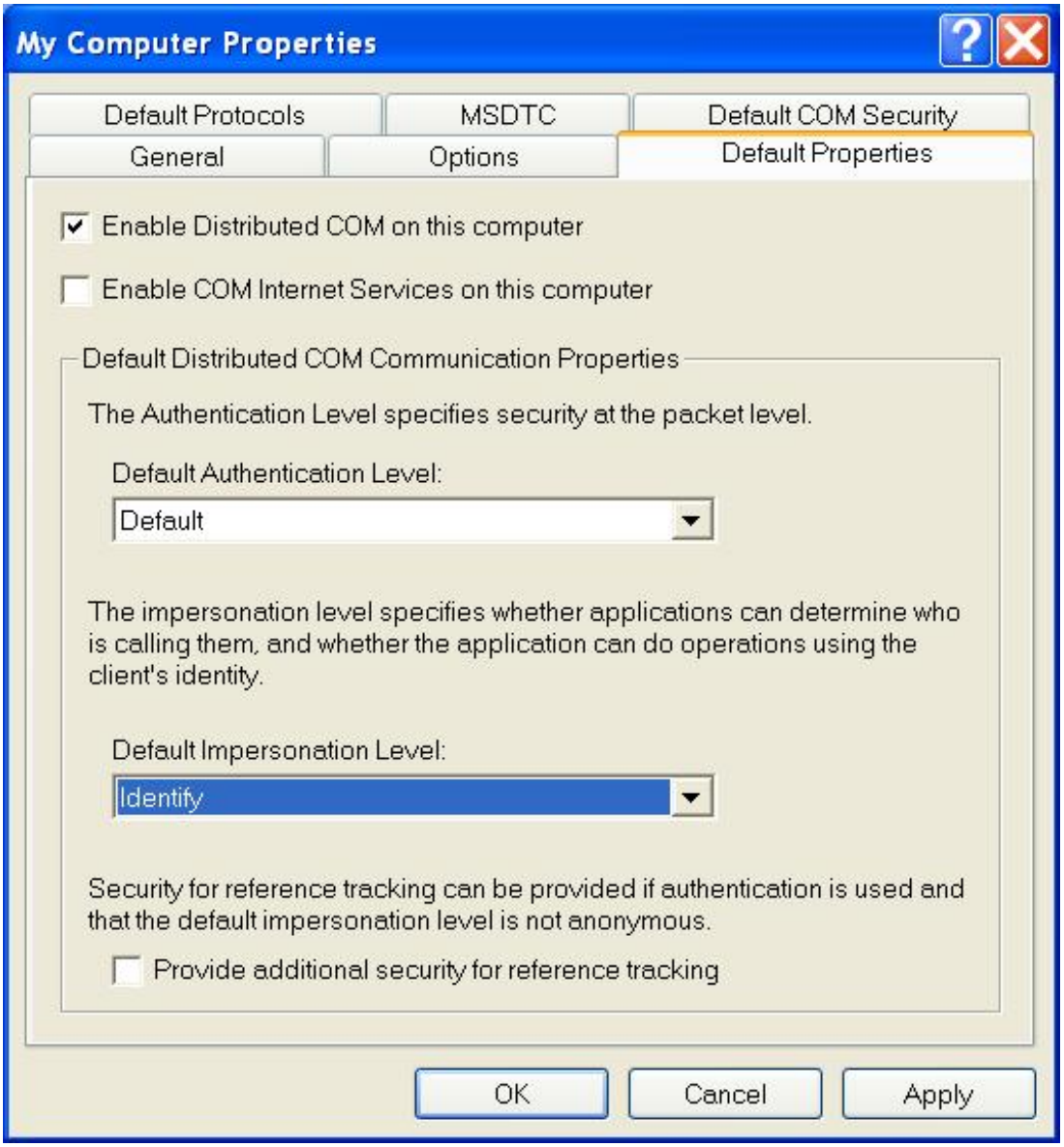
To configure NaturalX clients

1 Invoke **Component Services**.



2 In the **Properties** dialog of **My Computer**, select the **Default Properties** tab and activate the check box **Enable Distributed COM on this computer**.

Set **Default Authentication Level** to **Default** and **Default Impersonation Level** to **Identify**.

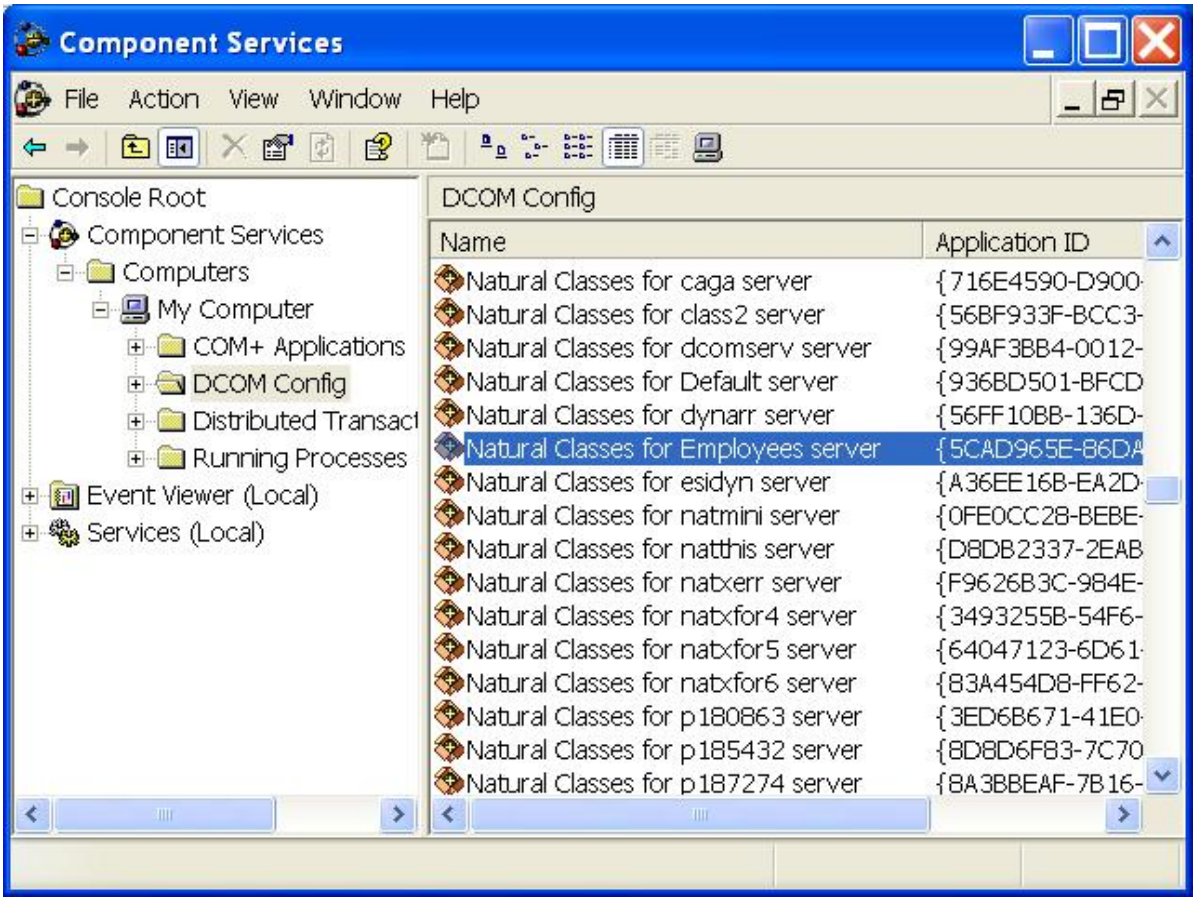


This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable `*USER` in order to let Natural Security checks run against this user ID.

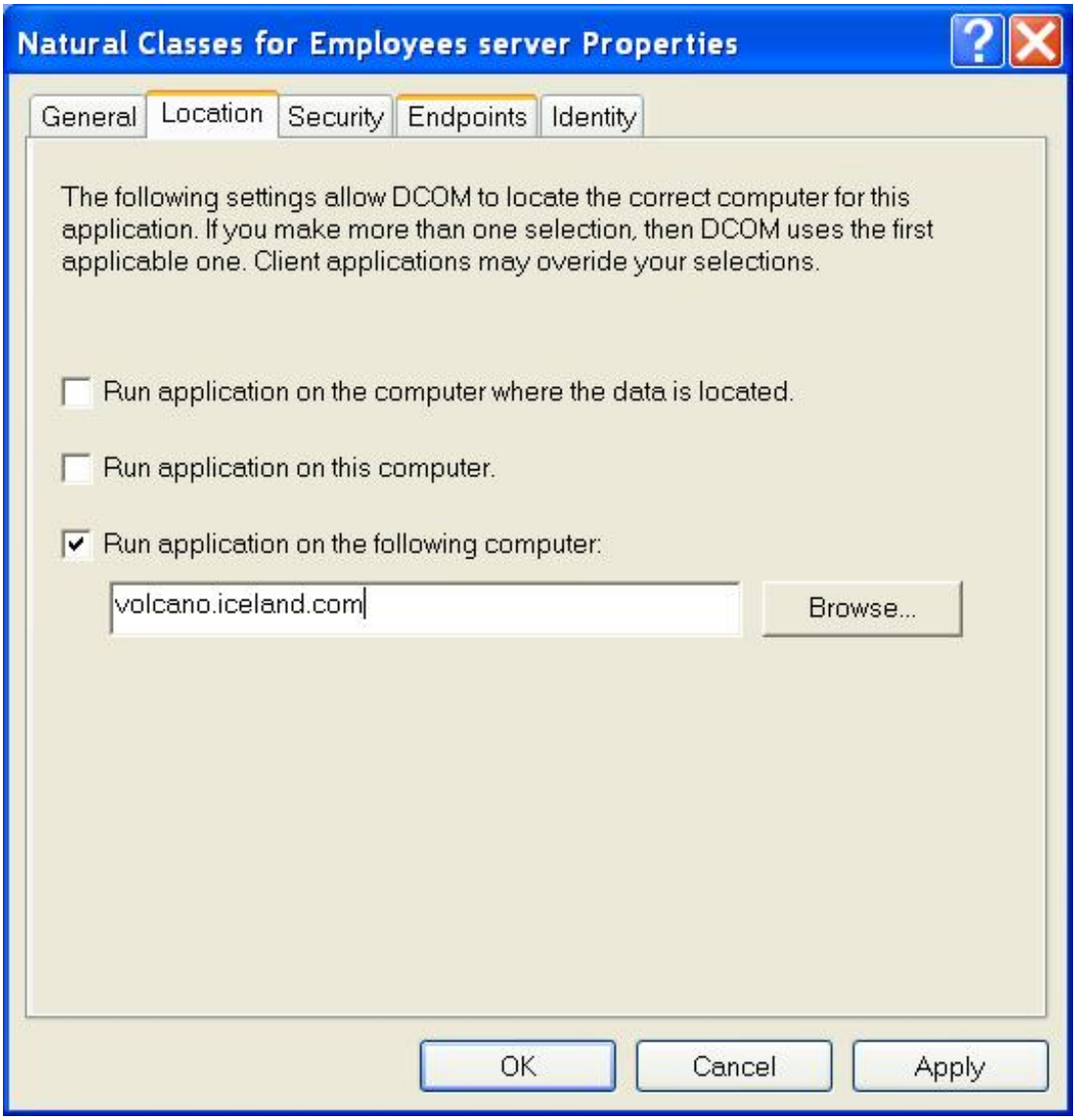
3 Now set up the configuration to access a specific NaturalX server.

Select the node **DCOM Config** and locate your NaturalX server in the **DCOM Config** list box (in the example **Natural classes for Employees server**).

Select your server and choose **Properties**.



4 In the **Location** tab, activate the check box **Run application on the following computer**. Enter the name of the remote machine on which the NaturalX server is installed.


Natural Classes for Employees server Properties
General
Location
Security
Endpoints
Identity
The following settings allow DCOM to locate the correct computer for this application. If you make more than one selection, then DCOM uses the first applicable one. Client applications may overide your selections.
Run application on the computer where the data is located.
Run application on this computer.
Run application on the following computer:
volcano.iceland.com
Browse...
OK
Cancel
Apply

23 NaturalX System Registry Entries

- Registry Entries for Servers 186
- Registry Entries for Clients 187

Registry Entries for Servers

The following tables show a summary of the keys and values that are added in the system registry of the server when a new class is registered.

The column “parent key” shows under which key the new key is created. The key which is added is listed in the column “subkey”, and the columns “value name” and “value” show the value of the new entry.

Note: `<class_name>` and `<class_ID>` are the name and the class GUID of the class respectively. They are defined in the `DEFINE CLASS` statement of the class module.

The following topics are covered below:

- Keys Needed by DCOM
- Keys Needed by Natural

Keys Needed by DCOM

parent key (*HKEY_CLASSES_ROOT...*)	subkey	value name	value
...	<ProgID> (<class_name>.1)	-	<class_name> "1.0"
... \<ProgID>	CLSID	-	<class_GUID>
...	<VersIdProgID> (<class_name>)	-	<class name> "1.0"
... \<VersIdProgID>	CLSID	-	<class GUID>
... \AppId	<APPID>	-	"Natural classes for" <server_ID> "server"
... \CLSID	<CLSID>	-	<class_name> "1.0"
... \CLSID	<CLSID>	AppId	<GUID for server>
... \CLSID \<CLSID>	LocalServer32	-	<Natural path>
... \CLSID \<CLSID>	ProgID	-	<ProgID>
... \CLSID \<CLSID>	TypeLib	-	<GUID for type library>
... \CLSID \<CLSID>	Version	-	"1.0"
... \CLSID \<CLSID>	VersionIndependentProgID	-	<VersIDProgID>
... \CLSID \<CLSID> (applies for Version 4.1.2 and all subsequent releases)	Programmable	-	-
... \TypeLib	<TLID>	-	-
... \TypeLib\<TLID>	1.0 <version>	-	"Natural" <class_name> "class"

parent key (*HKEY_CLASSES_ROOT...*)	subkey	value name	value
... \TypeLib\<TLID>\1.0	0 (langcode)	-	-
... \TypeLib\<TLID>\1.0\0	win32 (platform)		<type library path>
For every interface:			
... \Interface	<IID>	-	<interface name>
... \Interface\<IID>	ProxyStubClsid32	-	<GUID of proxy dll for IDispatch>
... \Interface\<IID>	BaseInterface	-	<GUID of IDispatch>

Keys Needed by Natural

parent key (*HKEY_LOCAL_MACHINE\ SOFTWARE\SoftwareAG\ Natural\Servers...*)	subkey	value name	value
...	<server_ID>	AppId	<GUID for server>
... \<server_ID>\	CLSID	-	-
... \<server_ID>\CLSID	<CLSID> (<class_ID>)	NatMember	<Natural class module name>
... \<server_ID>\CLSID	<CLSID>	NatLibrary	<Natural library of class module>
... \<server_ID>\CLSID	<CLSID>	NatContext	"ExternalSingle" or "InternalMultiple" or "ExternalMultiple" (see Activation Policies)

Registry Entries for Clients

The following table shows the keys which are added in the client system registry when the client registration file is executed:

parent key (*HKEY_CLASSES_ROOT...*)	subkey	value name	value
...	<ProgID> (<class_name>.1)	-	<class_name> "1.0"
... \<ProgID>	CLSID	-	<class GUID>
...	<VersIdProgID> (<class_name>)	-	<class_name> "1.0"
... \<VersIdProgID>	CLSID	-	<class GUID>
... \<VersIdProgID>	CurVer	-	<ProgID>

parent key (*HKEY_CLASSES_ROOT*...)	subkey	value name	value
... \AppId	<APPID>	-	"Natural classes for server" <server_ ID> "server"
... \AppId	<APPID>	RemoteServerName	has to be entered by user
... \CLSID	<CLSID>	-	<class_name> "1.0"
... \CLSID	<CLSID>	AppId	<GUID for server>
... \CLSID \<CLSID>	ProgID	-	<ProgID>
... \CLSID \<CLSID>	Version	-	"1.0"
... \CLSID \<CLSID>	VersionIndependent ProgID	-	<VersProgID>
... \CLSID \<CLSID> (applies for Version 4.1.2 and all subsequent releases)	Programmable	-	-
For every interface:			
... \Interface	<IID>	-	<interface name>
... \Interface\<IID>	ProxyStubClsid32	-	<GUID of proxy dll for IDispatch>
... \Interface\<IID>	BaseInterface	-	<GUID of IDispatch>

24 Using Statements and Commands in a NaturalX Server Environment

- Natural Statements 190
- Natural System Commands 191

The behavior of some Natural statements and Natural system commands changes in a server environment.

Natural Statements

This section covers the following topics:

- DISPLAY, INPUT, PRINT, REINPUT and WRITE Statements
- WRITE WORK FILE and READ WORK FILE Statements
- STOP and TERMINATE Statements

DISPLAY, INPUT, PRINT, REINPUT and WRITE Statements

- Output to a screen (output to Report 0) is not appropriate in a server environment, and in some cases is not possible. Therefore, in the case of an interactive I/O in the server environment, the error NAT0723 is returned to the client. Redirecting the I/O by using the `MAINPR` parameter is, of course, possible and is fully supported.
- When output is written to a report by a method, the report is opened at the start of the method and closed at the end. The report is not kept open between method calls to avoid interference between clients.

WRITE WORK FILE and READ WORK FILE Statements

When you access a work file in a method, the file is opened at the start of the method and closed at the end. The file is not kept open between method calls to avoid interference between clients.

STOP and TERMINATE Statements

- The behavior of the `TERMINATE` statement matches that of the `STOP` statement. Processing of return values is not supported.
- The `STOP` and `TERMINATE` statements behave in the same way as the `ESCAPE ROUTINE` statement during method execution. Method execution is terminated immediately without producing any return value.

Natural System Commands

Only the following Natural commands are allowed in the server environment:

- `CATALOG`
- `CLEAR`
- `EXECUTE`
- `LOGOFF`
- `LOGON`
- `READ`
- `RETURN`
- `RUN`
- `SAVE`
- `SETUP`
- `STOW`

From this list, the commands `CATALOG`, `CLEAR`, `READ`, `RUN`, `SAVE` and `STOW` are only allowed if the server is running under a development Natural (*natural.exe*).

All commands that are not allowed will be rejected with the error NAT0082.