

Operations Environment

- System-File Simulation
 - System Files FNAT and FUSER
 - Natural Root Directory
- Configuration Files
 - Local Configuration File (NATURAL.INI)
 - Global Configuration File (NATCONF.CFG)
- Work-Files
 - Defining Work-Files
 - Accessing Work-Files
 - Work-File Types
 - Record-File Format for OpenVMS
 - Special Considerations for Work-Files with Extension NCD
- Natural in Batch Mode
 - General Information
 - Batch-Mode Simulation
 - Real Batch Mode
 - Input and Output Channels
 - Sample Session for Batch Mode
 - Hints for Migration from Natural for OpenVMS Version 2.1 to Version 5.1
- Natural Exit Codes (for OpenVMS only)
- Startup Errors
- Invoking Natural Subprograms from 3GL Programs
 - Passing Parameters from the 3GL Program to the Subprogram
- Natural TX Interface (for UNIX only)
 - Overview
 - Operational Requirements
 - Natural with TOP END
 - Restrictions
- Using Versioning Software
- Issuing OpenVMS System Commands from a Natural Program
- Issuing UNIX System Commands from a Natural Program
- Tuning SQL Database Access
 - SQLRELCMD
 - SQLMAXSTMT
 - Example
- User Exit for Computation of Sort Keys - NATUSKnn

System-File Simulation

To ensure database independence, Natural does not store objects in a database system file. The system file is simulated by a structure of directories on the disk where Natural is installed.

Since the operating system is used instead of a database system, you can support database systems other than Adabas. Therefore, any text editor can be used to write source code files which are stored as ASCII text files on the disk.

Note for UNIX:

Semaphores are used to synchronize access to the Natural system files. Since this requires additional operating-system resources, you should consider incrementing the kernel parameters SEMMNI and SEMMNS by the number of system files to be emulated (refer to Change Kernel Parameters appropriate to your environment) in Activating the Natural Buffer Pool on UNIX in the Natural Installation Guide.

The Natural libraries are created as subdirectories below your current system file root directory. The command LOGON is therefore similar to the Change Directory command.

Under OpenVMS and UNIX, Natural requires ".NS n " as source filename extension, where n stands for the type of source. ".NSP", for example, stands for Natural source programs, and ".NSM" for Natural source maps.

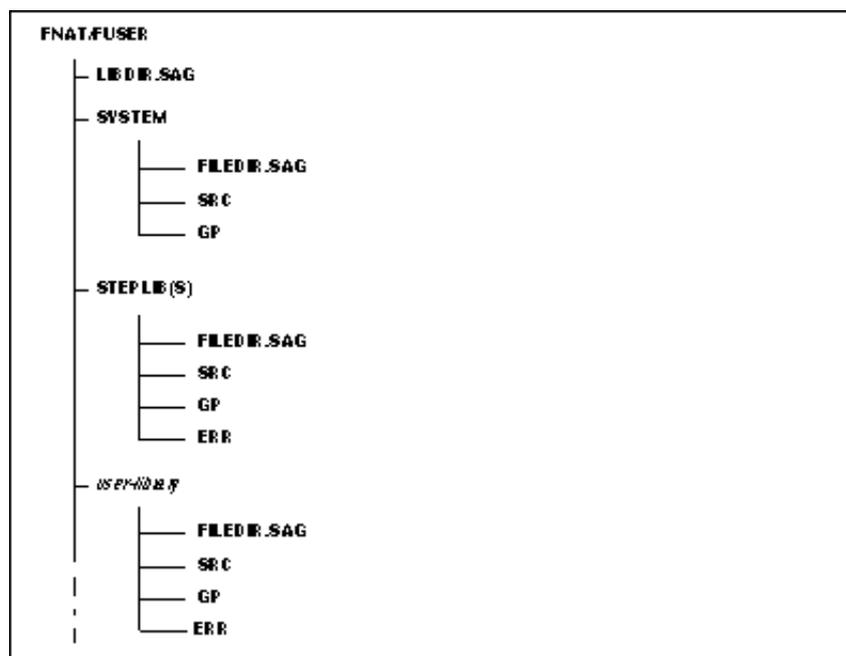
The available types of sources and the corresponding letters to be used for the extension are as follows:

P	Program
N	Subprogram
S	Subroutine
C	Copycode
H	Helproutine
T	Text
M	Map
L	Local data area
G	Global data area
A	Parameter data area
4	Class

Under OpenVMS and UNIX, a Natural library is modelled as a subdirectory of a system file root directory (for example, "/NATURAL") with the same name as the library. This subdirectory contains an additional subdirectory named "SRC" in which all sources and data areas are stored.

System Files FNAT and FUSER

The Natural system files FNAT (for system programs) and FUSER (for user-written programs) can be located in different subdirectories. They assume the following directory structure:



This directory structure is generated during the installation of Natural.

The file LIBDIR.SAG contains information on all further installed Software AG products using Natural. This information can be displayed by using the SYSPROD system command.

The directory representing the system libraries SYSTEM, STEPLIB, and each *user-library* contain the following:

- A file FILEDIR.SAG containing library information used by Natural including in particular 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), or 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 actual object name and an arbitrary number.
Both the actual object name and the corresponding internal object name are documented in FILEDIR.SAG. Even if an object is located in the correct directory, it can only be used by Natural after this library information is included in FILEDIR.SAG. For objects created within Natural, the library information is included automatically. For all other objects, the Import function of the SYSMAN utility should be used.
Also externally Natural generates a unique file name when a name is to be used that already exists in the system file, but not in the FILEDIR.SAG. That could be the case if a file was copied using operating system utilities instead of using Natural or its utilities. Natural will not overwrite the already existing file. The utility FTOUCH can be used to update FILEDIR.SAG without entering Natural.
- A subdirectory SRC containing the source objects stored in the library.
- A subdirectory GP containing the generated programs stored in the library.
- A subdirectory ERR containing the error messages stored in the library (this subdirectory is optional).

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

Natural Root Directory

During the installation, the logical names NATDIR and NATVERS (for OpenVMS) or the environment variables \$NATDIR and \$NATVERS (for UNIX) are created automatically. They point to the Natural version-dependant root directory.

For UNIX:

```
$NATDIR/$NATVERS
```

or

For Open VMS:

```
NATDIR: [ ' F$TRNLNM( "NATVERS" ) ' ]
```

Configuration Files

The base directory for FNAT/FUSER is found by looking in the following configuration files:

- NATURAL.INI
- NATCONF.CFG

Every user should have read-access to these files; however, only system administrators should be allowed to modify them (write-access is granted by using NATURAL.INI).

Notes:

If you use multiple Natural versions, the NATURAL.INI file is located in the appropriate "etc" directory for each installed Natural version. It is strongly recommended that only one NATCONF.CFG exists on a single computer or cluster. Maintaining only one file ensures a consistently defined database and/or system file for all users.

The configuration files contain plain ASCII text and should only be changed with the NATPARM utility.

To invoke the NATPARM utility, enter "NAT51PARM" at the OpenVMS DCL prompt or at the UNIX system prompt. The Natural Parameter Setting menu is displayed. On this menu, select "Configuration". A window appears with the two options:

- Local Configuration File
With this option, you can modify the configuration file NATURAL.INI.
- Global Configuration File
With this option, you can modify the configuration file NATCONF.CFG.

Note: If the "Configuration" option is not displayed on your Natural Parameter Setting menu, this means that you have no permission to modify the configuration files.

The contents of the two files are described on the following pages.

Local Configuration File (NATURAL.INI)

The local configuration file NATURAL.INI contains the following assignments:

Buffer-Pool Assignments

Various specifications related to the Natural Buffer Pool (BPID, Maxusers, MEMSIZE, Shared Memory Key, Semaphore Key and others).

Installation Assignments

- **Name of Global Configuration File (CONF_NAME)**
The name and path of the global configuration file (default name is NATCONF.CFG).
- **Path to Error File (ERROR_FILES)**
The location of the Natural error files.
- **Name of Natural I/O Conversion Table (NATCONV)**
The name of the file which contains the character translation tables used with the internal character set "ISO-8859-1". By default, this file is called NATCONV.INI; see Support of Different Character Sets for details.
- **Path to Binary Libraries of Software AG Products (NATEXTLIB)**
The location of binary libraries of further Software AG products using Natural.
- **Name of Natural Terminal Database (NATTCAP)**
The name of the database which contains the descriptions of the terminal capabilities for each terminal type supported by Natural. Refer to the NATTERMFCAP utility for details.
- **Path to Profile Parameters (PARM_PATH)**
The location of the Natural parameter files.

- **Path to Natural Profiles (PROFILE_PATH)**
The location of DDM editor (*.D00) profiles.
- **Path to TMP Directory (TMP_PATH)**
The location of Natural temporary output. In order to support separate user-specific TMP directories, the directory name may include embedded environment variables. These will be evaluated and substituted at run time. For example, C:\temp\%usertmp%\.%usertmp% will be replaced at Natural start-up with the contents of this environment variable.
- **Path to Text Files (TXT_PATH)**
The location of messages, choices and help texts used by the editors and by the NATTERMCP utility.

Administrator Assignments

With the Administrator Assignments function you can specify whether a user is to be a Natural Administrator or a Configuration Administrator.

A user who is defined as Configuration Administrator can modify the configuration files NATURAL.INI and NATCONF.CFG. Only Natural Administrators can be defined as Configuration Administrators.

Global Configuration File (NATCONF.CFG)

The global configuration file NATCONF.CFG contains the following assignments:

- DBMS Assignments
- System-File Assignment
- Dictionary-Server Assignment
- Printer Profiles
- Natural Version-Independent Operating System Files (NATOSDEP)

DBMS Assignments

Parameter	Function
DBID and DBMS Type	Specifies a value for each database ID.
DBMS Parameter	Establishes a connection to the database system. (Applies to SQL-type databases only).
SQL Date/Time Conversion	Specifies the date/time format for SQL databases.
Multifetch Disabling	Disables multifetch on an Adabas file and command-level basis.

If you select "DBMS Assignments", a dialog is displayed in which you can specify the DBID, DBMS Type and DBMS Parameter.

DBID and DBMS Type

Since the types of all databases which are to be accessed by Natural must be defined in the global configuration file, specify one of the following values for each database ID:

Value	Database Type
ADA	Adabas database server (this is the default).
ADX	Adabas single-user database (for PCs only).
OSQ	Any SQL database that can be accessed using Entire Access, Software AG's common interface to various SQL database systems. Note that in the user interface of the Natural Configuration utility the term OSQ is replaced with the term SQL .

Note:

You should define a database type for a DBID which has already been assigned to a system file; if you try to, an error message will be issued at Natural startup, indicating an inconsistency in the system file setting and an error when reading the database assignments.

A list of existing DBMS assignments is displayed in a list box.

DBMS Parameter

This field applies to SQL-type databases only.

In this field, you establish the connection to the database system that you want to work with. In addition, you can specify logon parameters specific to the database type.

See your Natural and Entire Access documentation for further information on how to access SQL-type database systems.

SQL Date/Time Conversion

As Natural has only one specific time format, the user has to decide how this format should be interpreted in the context of SQL database access.

If you select "SQL Date/Time Conversion", a dialog is displayed in which you can specify the conversion masks.

Mask

The value specifies the configuration for Entire Access. It also specifies the format used to retrieve SQL DATE/TIME/DATETIME information into Natural format A fields. The mask should match the RDBMS-specific configuration for the DATE, TIME, or DATETIME character string representation.

Date

This mask (usually a sub-string of the Mask value) specifies the character-string representation into which the Natural format D fields are converted during update or retrieval of SQL DATE columns.

Time

This mask (usually a sub-string of the Mask value) specifies the character-string representation into which the Natural format T fields are converted during update or retrieval of SQL TIME or DATETIME columns.

Remark

You can enter your remarks here, for example, to document how the SQL DATE/TIME character-string representation is configured on the database site.

See also the Entire Access documentation, Using Natural with Entire Access, Date/Time Conversion.

Multifetch Disabling

Natural uses Adabas command-level multifetch for Adabas L1, L2, L3 and L9 calls to minimize the number of database calls. For performance reasons it may be useful to disable multifetch on an Adabas file and command-level basis. This can be done with the Multifetch option. Before a multifetch call is issued, Natural checks if multifetch is disabled for the actual dbid/file/command code combination.

System-File Assignment

For each Natural system file, a database ID (DBID) and a file number (FNR) are specified in the Natural parameter file NATPARAM. To ascertain the location of each system file in the structure of directories, you use the System File Assignment function to assign a path name to the DBID/FNR combination. If you assign a path name that has already been assigned to another system file, all relevant internal information of the first assignment are copied automatically to the new assignment.

The path name must be a valid OpenVMS or UNIX directory path, indicating the physical path to the location of the system files on the disk.

Example:

All dictionary servers must first be defined by using the Natural RPC facility. The RPC parameter MAX-BUFF must be set to at least 8192 bytes.

Note: Currently, a server's node name must start with "FBKR" if you want it to be addressed by using a central mainframe broker.

Multiple logical server names can be defined; server assignments can be modified or deleted. A list of existing assignments can be displayed by pressing F4.

Specify the dictionary servers you want to use by setting the remote access parameter USEDIC to the corresponding logical server name; if USEDIC is set to blank, remote DDM access will not be possible.

Note: If you want to use this feature, Predict Version 3.3 or above must be installed; with Predict versions prior to 3.3, the Predict update tape PD2302 is required.

Printer Profiles

This function is used to define, modify or delete printer profiles.

Printer profiles are used for printing additional reports, for hardcopies and for batch output generation. They recognize particular Natural field attributes and insert the appropriate control sequences (see below) as defined in the profile.

With the ability to translate Natural field attributes into escape sequences, you can control your printer in various ways by using a specific profile name, and you can use the print features of a given device by using simple attributes in Natural programs.

To define a printer profile, you have to specify a profile name and leading and/or trailing commands (that is, printer control sequences) to be triggered at job, page or field level.

Each profile can be assigned to a Natural report number either statically by using the NATPARM Device/Report Assignments or dynamically by using the DEFINE PRINTER statement within a Natural program.

Triggering Events

A triggering event controls the level on which specified printer control sequences are to be applied. Available triggering events are: JOB, PAGE, AD and CD (a list of options can be obtained by pressing PF2).

- Specify JOB if you want your control sequences to apply to an entire print job; the specified control sequences will represent the job header and/or job trailer respectively.
- Specify PAGE if you want the control sequences to apply to each physical output page; the specified control sequences will then represent the page headers and/or page trailers respectively.
- Specify one of the Natural session parameters AD or CD along with appropriate field attributes if you want the control sequences to be applied at field level only; any field in a Natural program with corresponding attributes will then cause these control sequences to take effect. See the Natural Reference documentation for details on these session parameters.

Control Sequences

The leading control sequence is inserted immediately before the triggering event (for example, to define a job header or to set attributes for field representation). The trailing control sequence is inserted immediately after the triggering event (for example, to define a job trailer or to reset attributes previously set).

For each control sequence, a window appears, in which you can specify the control characters in either alphanumeric or hexadecimal format.

Example:

Alphanumeric format : **^1b(s1P**

Hexadecimal format : **^1b^28^73^31^50**

Note: The escape character must always be specified in hexadecimal format.

Natural Version-Independent Operating System Files (NATOSDEP)

This function is used to specify a directory common to all installed Natural versions, which is to contain operating system-specific Work-Files and temporary files used by all these Natural versions. These files are required for synchronization purposes and must not be deleted or modified by a Natural Administrator.

Note: If you use the system directory for temporary files, which is "/tmp" for UNIX and "SYSSCRATCH" for OpenVMS, ensure that no automatic procedures regularly delete the contents.

Work-Files

Work-Files are files where data can be written to and read from by Natural programs. They are used for intermediate storage of data and for data exchange between programs. Data can be transferred from or to a Work-File by using the Natural statements READ Work-File and WRITE Work-File, or UPLOAD and DOWNLOAD.

Defining Work-Files

To define a work-file, select Edit > Work-Files in NATPARM. In the Work-File Specification column, you can enter the name and extension for each of the up to 32 work-files.

If the specified work-file was previously defined, the current work-file definition is displayed, which can be overwritten. Enter the complete definition; that is, name *and* extension of your work-file. Press ENTER to save your settings, or press ESC to exit.

If the work-file was never defined, the Workfile Specification field is blank. Enter the work-file name including its path name.

For any numbered work-file, a complete work-file specification can be entered. This specification must contain the path and the work-file name.

If no work-file definition is specified, Natural automatically creates the file name and writes the work-file into the "tmp" directory specified in the local configuration file. The number consists of the specified work-file number and an arbitrary number assigned by the operating system.

Note for OpenVMS, AXP and VAX only:

The work-file name consists of the prefix "W", a number of up to seven digits and the extension "SAG".

Work-Files with Environment Variables

Work files can also be defined by using OpenVMS logical names or UNIX environment variables in the same way they are used with OpenVMS DCL commands or UNIX shell commands respectively.

Example:

Specify the following name for Work-File Number 4 in your Natural parameter module:

```
NATWORK04
```

Under OpenVMS, you use a logical name to define this work-file:

```
$ DEFINE NATWORK04 mydevice:[mydir]network04.dat
```

Under UNIX, you use an environment variable to define this work-file:

```
setenv NATWORK04 mydevice/mydir/network04.dat
```

Work-Files with Environment Variables (for UNIX Users)

Work files can also be defined by using them in the same way as work-file names that can be set without changing the NATPARM parameter file.

Example:

Specify the following name for a work-file:

```
$NATURAL/$myfile
```

and assume the following settings:

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

which will expand in the file name during Natural startup:

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

Work-Files with User Exit

Work files can also be defined by using the user exit USR1050N in the library SYSEXT.

Accessing Work-Files

Work-files can be accessed in two different ways: either locally or by using a data transfer with Entire Connection. Depending on the access method, a work-file must have a certain format.

Work-File Formats for Local Access

Three different work-file formats are available for local access. Natural recognizes the format by checking the work-file specification. This consists of a file name and an extension separated by a period:

file-name.extension

The three work-file formats are:

- a binary format specific to Software AG (to be preferred),
- ASCII format,
- a format which corresponds to the one used by Entire Connection.

For the work-file number to be used, the profile parameter "ENTIRE CONNECTION" in the NATPARM parameter file must be set to "N". A work-file specification is required for the appropriate format as described in the following sections.

Binary Format

The binary format specific to Software AG is defined by using any file name and either a period and the extension "SAG" or no period and extension at all.

Note: A two (2) byte length precedes each record written.

Examples:

xxxxxxx.SAG (any file name with a period and an "SAG" extension)

xxxxxxx (any file name without a period or extension)

The binary format can be used with all data types.

ASCII Format

To define an ASCII work-file format, enter a file name, a period and either any extension except "SAG" and "NCD" or no extension at all.

Examples:

xxxxxxxx.xxx (any file name with a period and any extension except "SAG" or "NCD")

xxxxxxxx . (any file name and a period)

Since each record written is terminated with the LF character sequence, this format is recommended for alphanumeric data only.

Entire Connection Format

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

The data file is defined by using any file name, a period and the extension "NCD"; the corresponding format file is automatically generated; it has the same name as the data file, but the extension ".NCF". Both extensions must be in upper case characters.

Examples:

xxxxxxxx.NCD (any file name with a period and an "NCD" extension for the data file)

xxxxxxxx.NCF (any file name with a period and an "NCF" extension for the format file)

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

Note: With Entire Connection, the RECORD option of the READ WORK FILE statement makes no sense and can therefore not be used.

Format for Work-Files to be Accessed by using a Data Transfer

Work files to be accessed by using a data transfer must be in Entire Connection format (see above).

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

For the work-file number to be used, you have to set the profile parameter ECPMOD to "Y" in NATPARM. No work-file specification is required in this case, because Entire Connection prompts you to enter a file name.

Work-File Types

Work-File Type PORTABLE

The work-file type PORTABLE performs an automatic endian conversion of a work-file when it 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 and F8. The floating point format is assumed to be IEEE. There are, however, slight differences in IEEE floating point representation by

different hardware systems. As far as we know, 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 computer-specific representation, so that a conversion is performed only if the file is read by a computer with different representation.

This keeps performance as fast as possible. There are no other conversions for the work-file type PORTABLE, but the file format makes it possible to add them in future releases.

When a READ WORK is done for a dynamic variable, it is resized to the length of the current record.

Work-File Type UNFORMATTED

The work-file type UNFORMATTED reads or writes a complete file with just one dynamic variable and just one record, e.g., to store a video which was read from a database. No formatting information is inserted, everything is written/read just as it is.

Record File Format for OpenVMS

Work files (1 to 32) with binary format are created with RMS file format variable length, whether all other files are created with RMS file format stream line feed and record attribute carriage return carriage control.

Special Considerations for Work-Files with Extension NCD

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.
- The maximum work-file record size and format file size that can be handled by Entire Connection is 1900 bytes.
- When an array is written to or read from an ".NCD" work-file, a maximum of 255 array elements can be written/read at a time.
- The VARIABLE option of the WRITE WORK FILE statement cannot be used for ".NCD" work-files.
- If you have "old" work-files whose names have ".NCD" extensions, and are not Entire Connection files, 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).
- Since a data transfer involves screen I/Os to send and receive the data, you should use a CLOSE WORK FILE statement after you finish writing the work-file and make sure that no Natural screen I/Os occur during the transfer.

After a read operation, the transfer is automatically closed when the end of the file has been detected; this enables screen I/Os, however, the work-file is still open in Natural and has to be closed by either using a CLOSE WORK FILE statement or returning to the command level.

Note: A work-file must also be explicitly closed in ON ERROR routines if these contain any screen I/Os.

- When transferring data by using Natural work-files to other platforms, remember that the range of possible values for floating point variables on a mainframe computer is different from that on other platforms.
- The possible value range for F4 and F8 variables on a mainframe is:

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

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

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

- for F8 variables:

$\pm 2.22 * 10^{-308}$ to $\pm 1.79 * 10^{308}$

- Data transfer is possible only for one work-file number at a time.
- A Natural error message is returned if DBMS calls are issued during an Entire Connection data transfer and their number exceeds the limit for DBMS calls permitted between screen I/Os (specified with the MADIO profile parameter).
- To circumvent this error, the user exit USR1068N in library SYSEXT is provided. USR1068 resets the database call counter to "0". It must be invoked each time a DBMS call is issued during data transfer.
- If files with the extension ".NCD" created by Entire Connection are read into Natural with the READ WORK FILE statement, you must ensure that Entire Connection does not remove trailing blanks from the records.
- To provide for that, Entire Connection must be set up as follows:
From the Host menu, select Session setup. Select the session, then click the Modify button. Select the tab File Transfer 1. Ensure that the Keep trailing blanks checkbox is checked.

Natural in Batch Mode

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

General Information

Natural can be used in two processing modes:

- Interactive Mode
- Batch Mode

In interactive mode, the input of commands and data comes from a terminal keyboard and the output is displayed on a terminal screen. In batch mode, input is read from a file and output is written to a file.

When Natural is run as a background batch job, no interaction between Natural and the person who submitted the batch job is necessary. The batch job consists of a set of programs such that each is completed before the next program is started. The programs are executed serially and receive sequential input data.

Batch mode is of particular interest for mass data processing and re-usable execution.

Detecting Batch Mode

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

in batch mode	*DEVICE is equal to "BATCH"
in interactive mode	*DEVICE is not equal to "BATCH" (in most of the cases *DEVICE is equal to "VIDEO")

Example:

```

IF *DEVICE = "BATCH" THEN
  /* Batch Mode Process */
  INPUT #NAME #AGE
ELSE
  /* Interactive Process */
  INPUT USING MAP ...
END-IF

```

Batch-Mode Simulation

This section describes how to simulate batch mode processing in an OpenVMS environment. With batch-mode simulation, the input and/or output is directed to a file instead to a terminal.

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

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

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

Example for OpenVMS to redirect the input channel:

```
$ DEFINE NATINPUT input-file-name
$ NATURAL BATCH
```

Example for UNIX to redirect the input channel:

```
Natural Batch < input-file-name
```

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

Example for OpenVMS to redirect the input and output channel:

```
$ DEFINE NATINPUT input-file-name
$ DEFINE NATOUTPUT output-file-name
$ NATURAL BATCH
```

Example for UNIX to redirect the input and output channel:

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

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

For OpenVMS:

```
$ DEFINE NATINPUT input-file-name
$ DEFINE NATOUTPUT nla0:
$ NATURAL BATCH
```

For UNIX:

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

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

Sample Input File for OpenVMS:

```
dlist program * /* select the direct command line and issue LIST command
. /* terminate the LIST command
fin /* terminate NATURAL
```

Sample Input File for UNIX:

```
dlist program **^M /*select the direct command line and issue LIST command
.^M /* terminate the LIST command
fin^M /* terminate NATURAL
```

The character "^M" is required for simulating the ENTER key.

The content of this input file has the same effect as the corresponding terminal input during an online session: after Natural has been started, the direct command line is selected, the LIST command is invoked and terminated, and Natural is terminated, too.

Real Batch Mode

To run Natural in real batch mode you have to specify the dynamic parameter BATCHMODE. In addition, some input and output channels have to be defined (as described below).

Advantages over Batch-Mode Simulation

It is recommended to use real batch mode instead of batch-mode simulation, because real batch mode has the following advantages:

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

Restrictions

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

- The terminal database SAGtermcap will be not supported. Therefore, the terminal capability TCS which is used for a different character set is not supported. To use a different character set, use environment variable NATCHARSET instead.
- No colors and video attributes (such as blinking, underlined and reverse video) are written to the batch output file CMPRINT
- Filler characters are not displayed within an INPUT statement
- No interactive input or output is possible

Input and Output Channels

The following input and output channels are of interest in batch mode:

- CMSYNIN
- CMOBJIN
- CMPRINT
- CMPRTnn
- CMWRKnn
- NATLOG

CMSYNIN (Batch Input File for Natural Commands and INPUT Data)

CMSYNIN is used for the input file that contains Natural commands, and (optionally) data to be read by INPUT statements during execution of Natural programs.

CMOBJIN (Batch Input File for Natural INPUT Data)

CMOBJIN is used for data intended to be read by Natural INPUT statements. This type of data can alternatively be placed in the CMSYNIN file immediately following the relevant RUN or EXECUTE command.

CMPRINT (Batch Output File)

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

CMPRTnn (Output File for Additional Reports)

CMPRTnn is used for additional reports referenced by any Natural program executed during the session. "nn" is two digit decimal number in the range 01 to 31 corresponding to the report number used in a DISPLAY, PRINT and WRITE statement.

In order to allow the user to specify variable print-file names, alpha-format system variables and numeric counter markers may be embedded in the filename specification for CMPRTnn. The supported alpha-format system variables are:

In order to allow the user to specify variable print-file names, alpha-format system variables and numeric counter markers may be embedded in the filename specification for CMPRTnn. The supported alpha-format system variables are:

*APPLIC-ID

*APPLIC-NAME

*DEVICE

*ETID

*INIT-USER

*LIBRARY-ID

*NET-USER

*PID

*PROGRAM

*USER

*USER-NAME

If any of these strings (in upper-case only) are encountered within the print file specification, they will be replaced at run time with the contents of the appropriate system variable. Additionally, a counter marker (#) may be used. This will be replaced by a 2-digit counter which will automatically be incremented for each print file.

Example:

The specification CMPRT01=abc_*PID_*ETID_*PROGRAM_#.dat in a Natural session with process-id 1234, ETID XYZ running a program with the name PRINT which produces print file output to file 01 would produce print-files with the following names (assuming the program runs 3 times):

```
abc_1234_XYZ_PRINT_01.dat
abc_1234_XYZ_PRINT_02.dat
abc_1234_XYZ_PRINT_03.dat
```

CMWRKnn (Batch Output File for Natural Work-Files)

CMWRKnn is used for Natural work-files referenced by any Natural program executed during the session. "nn" is a two digit decimal number in the range 01 to 32 corresponding to the number used in a READ WORK FILE or WRITE WORK FILE statement.

NATLOG (Natural Log File)

NATLOG is used to log messages that could not be written to the batch output file CMPRINT. It is recommended to enable NATLOG in batch mode.

Sample Session for Batch Mode

The following example demonstrates how to start Natural in batch mode. A simple Natural program is executed and three data items are taken from the batch input file. After the items are processed from the INPUT statement, a subsequent DISPLAY statement writes the data to the batch output file. Then, Natural terminates.

Natural Commands (CMSYNIN=batch.cmd):

```
LOGON SYSEXBAT
EXECUTE RECCONT
FIN
```

Natural Input Data (CMOBJIN=batch.inp):

```
Ben,%
Smith
```

Natural Program RECCONT in Library SYSEXBAT:

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

Natural Command Line:

```
NATURAL BATCHMODE CMSYNIN=batch.cmd CMOBJIN=batch.inp CMPRINT=batch.out
BMSIM=MF NATLOG=ALL
```

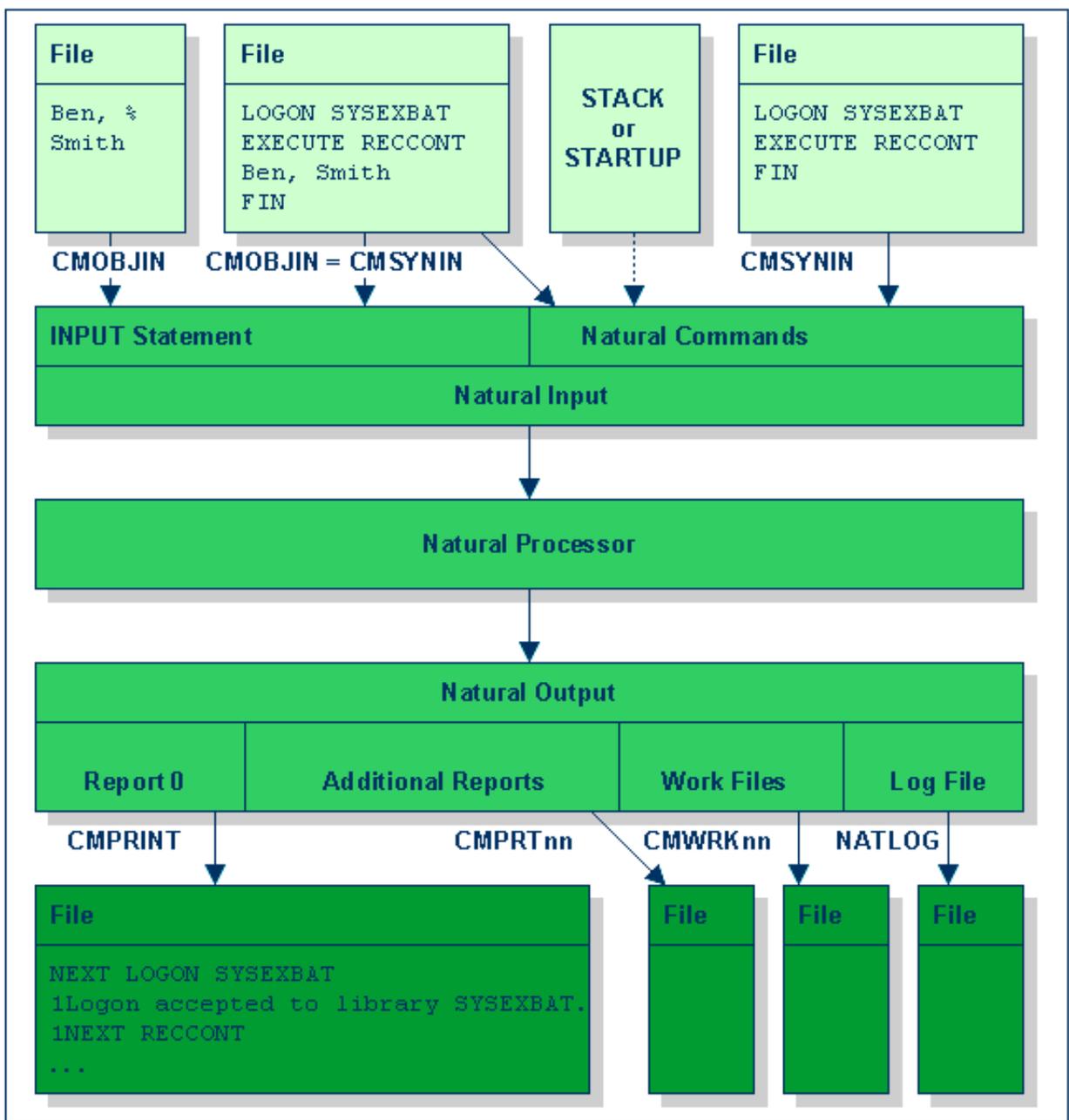
After Natural is invoked by using the command line described above, Natural produces the following output.

Contents of Batch Output File (CMPRINT=batch.out) after Execution:

```

_NEXT LOGON SYSEXBAT
1Logon accepted to library SYSEXBAT.
1NEXT EXECUTE RECCONT
1#firstname          #lastname
_DATA Ben,%
_DATA Smith
1Page      1                      01-10-13  10:55:30
0#FIRSTNAME #LASTNAME
-----
0Ben        Smith
1NEXT FIN
1NAT9995 NATURAL session terminated normally.
    
```

The following graphic illustrates how Natural reads input and writes output in batch mode.



Hints for Migration from Natural for OpenVMS Version 2.1 to Version 5.1

The following table shows the main batch environment differences between Natural for OpenVMS Version 2.1 and Version 5.1.

Action	Natural Version 2.1	Natural Version 5.1
Activate Batch Mode	-	Specify dynamic parameter BATCHMODE
Define channel for Commands	Define logical symbol NATURAL\$COMMAND	Set parameter CMSYNIN
Define channel for Input Data	Define logical symbol NATURAL\$INPUT	Set parameter CMOBJIN
Define channel for Output	Define logical symbol NATURAL\$OUTPUT	Set parameter CMPRINT

Optionally, you may set the parameter BMSIM to VM to generate as for Natural Version 2.1 output.

Natural Exit Codes

There are two types of Natural exit codes:

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

Special Considerations for Natural on OpenVMS

If Natural fails during startup or is terminated by the TERMINATE statement, you can query the exit code from within a DCL file. As opposed to Natural for UNIX, Natural for OpenVMS does not return low return code values such as "12" (startup error "Parameter module not found"). Instead, Natural for OpenVMS adds the value "268,435,456" to the Natural error code one would get using Natural for UNIX. If this value were not added, the image run-down handler would interpret the value "12" as a Natural internal access violation and a system access violation message would be printed.

Exception:

If Natural is terminated by using the TERMINATE statement with Value 0, Natural exits with Exit Code 1. This is because under OpenVMS the C-runtime makes no difference between an exit (0) and an exit (1) statement. In both cases, a 1 is returned.

You can retrieve this error code in a DCL command file by inquiring the status value \$STATUS. The following example shows how to retrieve the Natural exit code.

Recommendation:

The value of \$STATUS does not explain if this is a startup error or an error generated by the TERMINATE statement. For Natural version 5.1.1 there are about 60 startup errors defined. For that reason, use Exit Code 100 to 255 for TERMINATE, otherwise you could not clearly indicate the reason for the exit.

```

$ SET NOON
$ NAT := $NATBIN:NATURAL.EXE
$ DEFINE/USER NATINPUT SYS$INPUT
$ DEFINE/USER NATOUTPUT SYS$OUTPUT
$
$ nat parm=unknown
$
$! Note:
$! %X10000000 is hexadecimal and stands for the decimal number 268,435,456
$!
$ status_value = $STATUS .AND. .NOT. %X10000000
$
$ IF status_value .GE. 100
$ THEN
$ WRITE SYS$OUTPUT "NATURAL terminates by using TERMINATE statement"
$ ELSE
$ IF status_value .EQ. 1
$ THEN
$ WRITE SYS$OUTPUT "NATURAL terminates successfully"
$ ELSE
$ WRITE SYS$OUTPUT "NATURAL terminates with startup error " + "'status_value'"
$ ENDIF
$ ENDIF
$
$ EXIT

```


Natural Startup Errors

0	On UNIX: No error occurred.
1	On Open VMS: No error occurred.
2	Terminal Control String (TCS) capability specified in SAGtermcap or Environment Variable NATTCHARSET.
3	Failed to initialize character conversion table.
4	Failed to load character conversion table.
5	Error reading the DATABASE assignments in the global configuration file.
6	Unable to find FNAT or FUSER. Check your configuration files.
7	Cannot initialize LFILE table.
12	Unable to read specified parameter module. Please verify the parameter file.
13	Unable to read parameter module.
14	Storage manager initialization failed.
15	End of file (EOF) encountered while reading from STDIN.
16	Unable to open buffer pool; contact the Natural system administrator.
17	Error reading the buffer pool assignments in the local configuration 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.
27	Login cancelled. Natural terminates.
28	Security violation during start of Natural. Natural terminates.
29	Security violation during start of Natural. Login aborted due to too many login failures.
30	Natural system error message raised.
31	This is not a security nucleus.
32	Password check failed.
33	Lock manager cannot create/initialize semaphores.
34	Unable to build Natural Library structure. Check your System File assignment. User/group settings might not be correct. S-bit setting might be required. Path specifications in global configuration file might be corrupted. LIBDIR.SAG might be missing.
35	Internal wfc i/o terminal driver error.
36	Internal XVT error.

37	DCOM Startup error.
38	Creation of runtime context failed.
39	NATDIR and/or NATVERS not set.
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 NATPARM utility.
42	Batch mode driver error.
43	Screen window size is too small.
44	Exit from SQL signal handler.
45	Unable to load add-on product.
46	Unable to access FNAT library SYSLIB. Insufficient privilege or file protection violation.
47	Unable to read PARM_PATH.
48	Unable to read NATURAL.INI for CONFIG_NAME.
49	Unable to read NATURAL.INI for NATTCAP.
50	Unable to read NATURAL.INI for NATCONV.
51	Unable to read TMP_PATH.
52	Unable to read PROFILE_PATH.
53	Unable to open NATURAL.INI.
54	Unable to read NATCONF.CFG for NATOSDEP.
55	Unable to read NATURAL.INI for NATEXTLIB.
56	Unable to find NATDIR in SAG.INI.
57	Unable to find DEFAULT-VERSION in SAG.INI.
58	Unable to find NATINI in SAG.INI.
59	Invalid option specified.
60	Not enough memory to initialize internal tables.
61	Batch error occurred, but processing continued due to CC=ON parameter.
62	More than one Natural session with active repository not allowed
63	Natural session with active repository already running

Invoking Natural Subprograms from 3GL Programs

Natural subprograms can be invoked from a programming object written in a 3rd generation programming language (3GL). The invoking program can be written in any programming language that supports a standard CALL interface.

For this purpose, Natural provides the interface "ncxr_callnat". The 3GL program invokes this interface with a specification of the name of the desired subprogram.

Note: Natural must have been activated beforehand; that is, the invoking 3GL program must in turn have been invoked by a Natural object with a CALL statement.

The subprogram is executed as if it had been invoked from another Natural object with a CALLNAT statement.

When the processing of the subprogram stops (either with the END statement or with an ESCAPE ROUTINE statement), control will be returned to the 3GL program.

Passing Parameters from the 3GL Program to the Subprogram

Parameter fields can be passed from the invoking 3GL program to the Natural subprogram. For passing the parameter fields, the same rules apply as for passing parameter fields with a CALL statement.

The 3GL program invokes the Natural interface "ncxr_callnat" with four parameters:

Parameter	Contents
1	The name of the Natural subprogram to be invoked.
2	The number of parameter fields to be passed to the subprogram.
3	The address of the table that contains the addresses of the parameter fields to be passed to the subprogram.
4	The address of the table that contains the format/length specifications of the parameter fields to be passed to the subprogram.

The sequence, format and length of the parameter fields in the invoking program must match exactly the sequence, format and length of the fields in the DEFINE DATA PARAMETER statement of the subprogram. The names of the fields in the invoking program and the invoked subprogram can be different.

For an example of how to invoke a Natural subprogram from a 3GL program, refer to the samples "MY3GL.NSP" (for the main program), "MY3GLSUB.NSN" (for the subprogram) and "MYC3GL.C" (for the "C" function).

For UNIX:

```
$NATDIR/$NATVERS/samples/sysexuex
```

For OpenVMS:

```
NATDIR:[v5110,samples]
```

Natural TX Interface (for UNIX only)

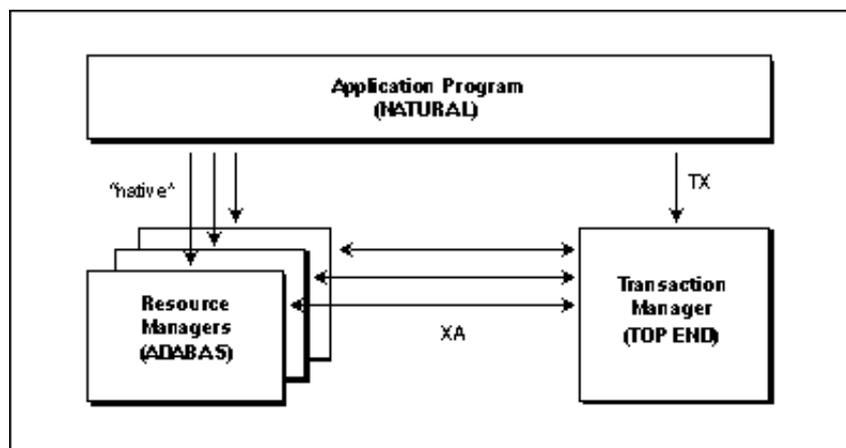
The Natural TX interface is an optional feature to enable Natural users to perform distributed transaction processing.

Overview

By supporting distributed transaction management systems, the Natural TX interface offers distributed transaction processing to the Natural user, based on the X/Open model for distributed transaction processing. This model comprises three basic elements:

- The application program; that is, Natural.
- The transaction manager; that is, for example, TOP END.
- The resource managers; that is, for example, several Adabas database management systems.

Note: Currently, the Natural TX interface only supports the database management system Adabas for UNIX Version 2.2.1 (or above) and the distributed transaction management system TOP END.



The Natural application program is the component that determines when a distributed transaction begins and ends. It uses resource (that is, database) managers to perform work that is to be part of the distributed transaction. Natural ends a transaction with an END TRANSACTION (commit) or BACKOUT TRANSACTION (rollback) statement and is not involved in the two-phase commit procedure.

The transaction manager processes the transaction begin and end statements from Natural and conveys this information to the database managers. It also coordinates the two-phase commit and transaction recovery processing.

A resource manager manages a certain part of the system's shared resources, for example, the databases.

The application programming interface (API) between the application program and the transaction manager is called TX interface; the system programming interface between the transaction manager and the resource managers is called XA interface; for information on the Adabas XA interface, refer to the Adabas for UNIX DBA Reference Documentation.

Operational Requirements

To be able to use the Natural TX interface, you have to relink the Natural nucleus. To do so, perform the following steps:

1. To make the Natural "build" directory the current directory, issue the command:

```
cd $NATDIR/$NATVERS/bin/build
```

2. To build a new Natural environment issue the command:

```
make Natural tp=yes ada=dyn
```

This "make" command generates a new "Natural" file which is linked to the TX interface.

3. To copy this new "Natural" file into the "bin" directory, issue the following command:

```
make install
```

To make the Natural TX interface operational, you have to modify your Natural parameter file as follows:

1. Invoke the NATPARM utility by entering the command "natparm" at the system prompt: The NATPARM menu is displayed.
2. Select the Edit option: A window appears with a list of parameter groups.
3. Select "DBMS": A second window is displayed with the individual parameters that belong to this group.
4. Select "XA Databases (XADB)"; see also the description of the XADB profile parameter: A further window is displayed, in which the DBIDs of all databases which will be used in distributed transaction processing must be marked.

Note: The selected databases must be equipped with an XA interface to the corresponding transaction manager.

Natural with TOP END

To make Natural known to TOP END, you have to define a new TOP END System and a new Application Component (for example "natural").

The name of the TOP END System should correspond to the user and group definitions specified for Natural (for example, "sag").

In addition, all relevant environment variables (TP_SYSTEM, TP_NAME, etc.) must be set at runtime.

Restrictions

When using the Natural TX interface, no transaction data can be stored with the END TRANSACTION statement or retrieved with the GET TRANSACTION DATA statement.

Using Versioning Software

If you want to use any third-party versioning software together with Natural, you have to write your own interface. This interface is called by Natural each time an object is modified. For more information on how to build this interface, refer to the example in directory "\$NATDIR/\$NATVERS/samples/natncvc".

Issuing OpenVMS System Commands from a Natural Program

The Natural user exit "SHCMD" can be used to issue a system command, call a DCL command procedure or execute an executable program on OpenVMS from within a Natural program.

Format:

```
CALL 'SHCMD' command [option]
```

Parameters:

Command	Command which is executed under control of the operating system command interpreter. Natural will wait until the command is completed and then Natural returns the control back to the Natural program.
Option This parameter is optional	Option describes how the command should be executed.

The following options are available:

SCREENIO	Used to refresh the Natural screen output after the command is completed
NOSCREENIO	Used to hide the output generated by the command. The hidden output is redirected to the null device.

Note: Option SCREENIO and NOSCREENIO may be not used at the same time.

Return Codes:

Natural will manipulate return codes from the control interpreter in the following way:

- Even numbers are interpreted as error codes and are retained unchanged.
- Odd numbers are interpreted as success codes and are converted to 0.

The following return code values are available:

0	Command successfully executed.
4	Illegal SHCMD parameter specified.
all others codes	Command has returned an error.

Examples:

Execute a DCL command procedure from within Natural:

```
CALL 'SHCMD' '@MYDCL.COM'
```

Execute an executable program from within Natural:

```
CALL 'SHCMD' 'MYPROGRAM.EXE'
```

Executing system command DIRECTORY on OpenVMS to view a directory:

```
CALL 'SHCMD' 'DIRECTORY'
```

After executing the DIRECTORY command, you will recognize that the output generated by the system command has changed the last Natural screen output. You have to press the refresh-screen key to clear the screen. To do this automatically, you can specify the 'SCREENIO' option.

```
CALL 'SHCMD' 'DIRECTORY' 'SCREENIO'
```

Retrieve the return code by using the RET function:

```
DEFINE DATA LOCAL
  1 rc (I4)
END-DEFINE
CALL 'SHCMD' 'DIRECTORY' 'SCREENIO'
ASSIGN rc = RET( 'SHCMD' )           /* retrieve return code
IF rc <> 0 THEN
  IF rc = 4 THEN
    WRITE NOTITLE 'Illegal option specified'
  ELSE
    WRITE NOTITLE 'Command not executed successfully (rc=' rc ') '
  END-IF
ELSE
  WRITE NOTITLE 'Command executed successfully'
END-IF
END
```

Execute the DCL command procedure MYSUCCESS.COM. Natural will recognize that the command procedure returns an odd return number and assumes success by mapping the exit code 15 to 0.

DCL command procedure MYSUCCESS.COM:

```
$ WRITE SYS$OUTPUT "DCL returns SUCCESS"
$ EXIT 15
```

Executing the DCL command procedure MYERROR.COM. Natural would recognize that the command procedure returns an even return number, assumes error and leaves the number unchanged.

DCL command procedure MYERROR.COM:

```
$ WRITE SYS$OUTPUT "DCL returns ERROR"
$ EXIT 14
```

Issuing UNIX System Commands from a Natural Program

The Natural user exit "SHCMD" can be used to issue a system command, call a shell script or execute an executable program on UNIX from within a Natural program.

Format:

```
CALL 'SHCMD' command [option]
```

Parameters:

command	Command which is executed under control of the operating system command shell. Natural will wait until the command is completed and then Natural returns control back to the Natural program.
option This parameter is optional	Option which describes how the command should be executed.

The following options are available:

SCREENIO	Used to refresh the Natural screen output after the command is completed.
NOSCREENIO This parameter is optional	Used to hide the output generated by the command. The hidden output is redirected to the null device.

Note:

Option SCREENIO and NOSCREENIO may be not used at the same time.

Return Codes:

The following return code values are available:

0	Command successfully executed.
4	Illegal SHCMD parameter specified.
All others codes This parameter is optional	Command shell return code.

Examples:

Execute a command shell from within Natural:

```
CALL 'SHCMD' 'myshell.sh'
```

Execute an executable program from within Natural:

```
CALL 'SHCMD' 'myprogram'
```

Execute system command ls on UNIX to list the contents of a directory:

```
CALL 'SHCMD' 'ls'
```

After executing the `ls` command, you will recognize that the output generated by the system command has changed the last Natural screen output. You have to press the refresh-screen key to clear the screen. To do this automatically, you can specify the `'SCREENIO'` option.

```
CALL 'SHCMD' 'ls' 'SCREENIO'
```

Retrieve the return code by using the `RET` function:

```
DEFINE DATA LOCAL
  1 rc (I4)
END-DEFINE
CALL 'SHCMD' 'ls' 'SCREENIO'
ASSIGN rc = RET( 'SHCMD' )           /* retrieve return code
IF rc <> 0 THEN
  IF rc = 4 THEN
    WRITE NOTITLE 'Illegal option specified'
  ELSE
    WRITE NOTITLE 'Command not executed successfully (rc=' rc ')'
  END-IF
ELSE
  WRITE NOTITLE 'Command executed successfully'
END-IF
END
```

Tuning SQL Database Access

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

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

SQLRELCMD

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

Possible values:

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

SQLMAXSTMT

This parameter determines the size of the statement table.

Possible values:

- 1 to 64 (default: 16)

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

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

Example

To set the above parameters dynamically, enter the following at the Natural command prompt:

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

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

User Exit for Computation of Sort Keys - `NATUSKnn`

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

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

You can write a NATUSKnn user exit in the C programming language using the CALL interface. The *character-string* specified with SORTKEY will be passed to the user exit. The user exit has to be programmed so that it converts "incorrectly sorted" characters in this string into corresponding "correctly sorted" characters. The converted character string is then used in the Natural program for further processing.

Note: A conversion table is currently not supplied.

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

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

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

A sample user exit, natusk01.c, is provided in source form: it applies to English and converts all English lower case letters in the character string to upper case. The sample is to be found in \$NATDIR/\$NATVERS/samples/sysexuex, where you can also find the other user exits.

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

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