Natural

Natural System Architecture

Version 8.2.6

April 2017

ADABAS & NATURAL

This document applies to Natural Version 8.2.6 and all subsequent releases.

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

Document ID: NATMF-NNATARC-826-20170418

Table of Contents

Preface

This documentation describes the basic system architecture of Natural for Mainframes and the client/server architecture of Natural's Single Point of Development (SPoD). SPoD allows centralized development for multiple platforms.

Natural Basic Architecture Overview	Major architectural components of Natural.
Natural SPoD Architecture	Major architectural components of Natural's Single Point of Development

I Natural Basic Architecture Overview

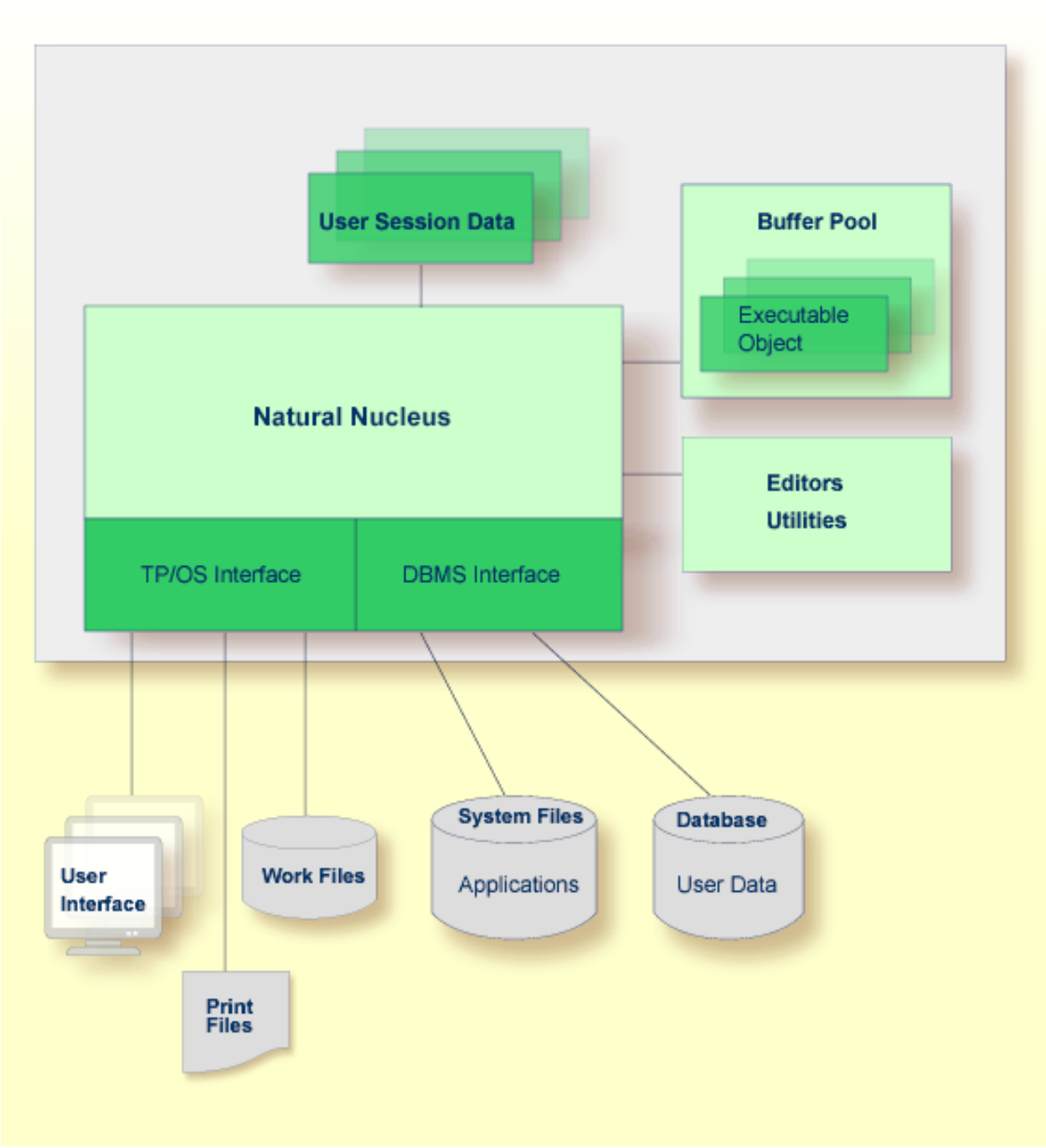
The architectural concept of Natural for Mainframes is primarily based on the principle of open architecture and is realized via the definition of interfaces. Information is exchanged between these interface and external components during both application development and program execution.

The functions of the Natural system are available on all platforms on which Natural runs and where they are applicable. Natural hides the complexity of the system environment (operating system, TP system, database system, user environment) from the application development environment.

Special system-dependent tables and routines embed the Natural components into the system environment and provide the internal functions with the required resources.

This section describes the major components of Natural and how they interact to provide Natural's functionality as a development tool.

For information on each component, click on an item in the diagram below or choose an item from the menu below.


User Session Data
Buffer Pool
Executable Object
Natural Nucleus
Editors
Utilities
TP/OS Interface
DBMS Interface
User Interface
Work Files
System Files
Applications
Database
User Data
Print Files

Natural Nucleus	Major components of the Natural nucleus: compiler, runtime system, command interpreter and configuration settings (Natural parameters).
User Session Data	Temporary data storage in user-specific work areas.
Natural Buffer Pool	Principles of Natural buffer pool operation and object loading.
Natural Editors and Utilities	Natural editors and utilities used for building and maintaining an application.
TP/OS Interface	TP monitor and OS (operating system) interfaces provided by Natural.
User Interface	User interfaces supported by Natural.
Print Files **Work Files**	Use of print files and work files.
Natural System Files	Storage of object modules in Natural system files.
DBMS Interface	Database management system (DBMS) interfaces provided by Natural.
Database Access	Natural access to data stored in databases.

1 Natural Nucleus

The Natural nucleus consists of two functional parts: the environment-dependent nucleus and the environment-independent nucleus.

The environment-dependent nucleus contains components that depend on the mainframe operating system or TP system (online interface) in use.

The environment-independent nucleus can be shared by different mainframe operating and TP systems and used simultaneously by multiple users. The environment-independent nucleus comprises the kernel of Natural which supplies important Natural functions such as command interpretation, object compilation and object execution.

This section describes the main components of the environment-independent nucleus.

Natural Runtime System

The Natural runtime system can be considered a virtual machine that provides the environment necessary for executing Natural objects within an application. The Natural runtime system interprets and executes Natural internal object code (binary meta code).

The section below contains information on:

- Object Execution
- Object Starter and Object Executor

Object Execution

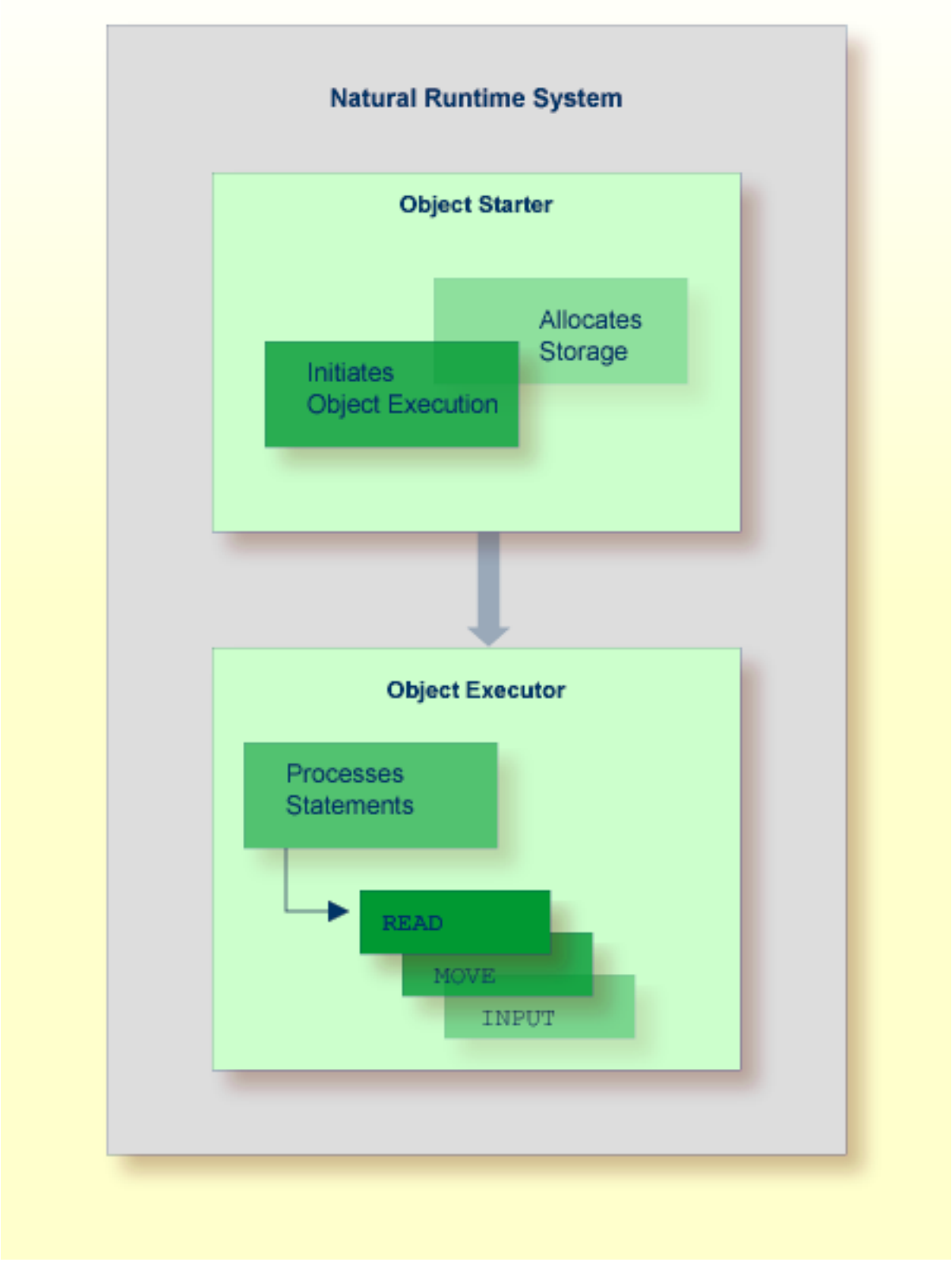
Internal Natural object code is executed when a Natural object is requested for execution.

Object execution is requested either directly by a user, or indirectly if the object currently being executed issues a Natural statement that requests execution of another object. The Natural system command `EXECUTE`, for example, directly executes the user-written program specified with the command. The Natural `CALLNAT` statement specified in a Natural program, for example, requests execution of a subprogram.

The ***Example of Object Loading*** in the section *Natural Buffer Pool* illustrates the process flow when executing a Natural program.

Object Starter and Object Executor

The diagram below is a schematic illustration of object execution by the Natural runtime system:


Natural Runtime System
Object Starter
Allocates Storage
Initiates Object Execution
Object Executor
Processes Statements
READ
MOVE
INPUT

The Natural runtime system consists of two major components: the object starter and the object executor.

The object starter locates the object to be executed in the **Natural buffer pool** and allocates storage for variable data processed by the object as **user session data**. Finally, the object starter passes control to the object executor.

The object executor interprets the Natural statements in the object and executes them one after the other. In the diagram above, for example, the object executor first processes the `READ` statement by calling the database and retrieving the records requested and then continues with the `MOVE` statement.

Related Topics:

- ***User Session Data***
- ***Natural Buffer Pool***
- ***Example of Object Loading*** - *Natural Buffer Pool*

Natural Compiler

The Natural compiler generates the executable form of Natural source code. Natural source code is human-readable programming code that consists of a sequence of Natural statements. (For information on Natural statements and programming advice, see the *Statements* documentation and the *Programming Guide*.)

The Natural compiler reads the source code from the source area. The source code in the source area is part of the user session data. Source code is read into the source area by using the Natural system command `READ` or `EDIT`. The compiler checks the syntax of the source code and, if successful, generates Natural internal object code that can be interpreted and executed by the **Natural runtime system**.

When using the appropriate Natural system command, source code can be either compiled and executed, or compiled and stored. The section below describes the types of object module available for storage and the Natural system commands used for object compilation and execution:

- Cataloged Object
- Source Object
- Commands for Compilation

- Example of Compilation

Cataloged Object

A cataloged object is the executable (compiled) form of a Natural object. It is created by the Natural compiler and is stored as an object module in a Natural system file. Compiling source code and creating a cataloged object is referred to as cataloging an object. A cataloged object is created by using the Natural system command `CATALOG` or `STOW`.

At execution time, the cataloged object is loaded into the Natural buffer pool and executed by the Natural runtime system. Natural objects can only be executed or reference one another if they have been stored as cataloged objects in a Natural system file.

A cataloged object cannot be modified or decompiled.

Source Object

A source object (or a saved object) contains the human-readable form of Natural source code. Source code is saved as a source object in a Natural system file by using the Natural system command `SAVE` or `STOW`.

To execute source code contained in a source object, you need to compile the source code in order to create generated object code that can be interpreted and executed by the Natural runtime system.

Commands for Compilation

Natural provides different system commands for compiling source code. Depending on the Natural system command used, compilation is combined with any of the following actions:

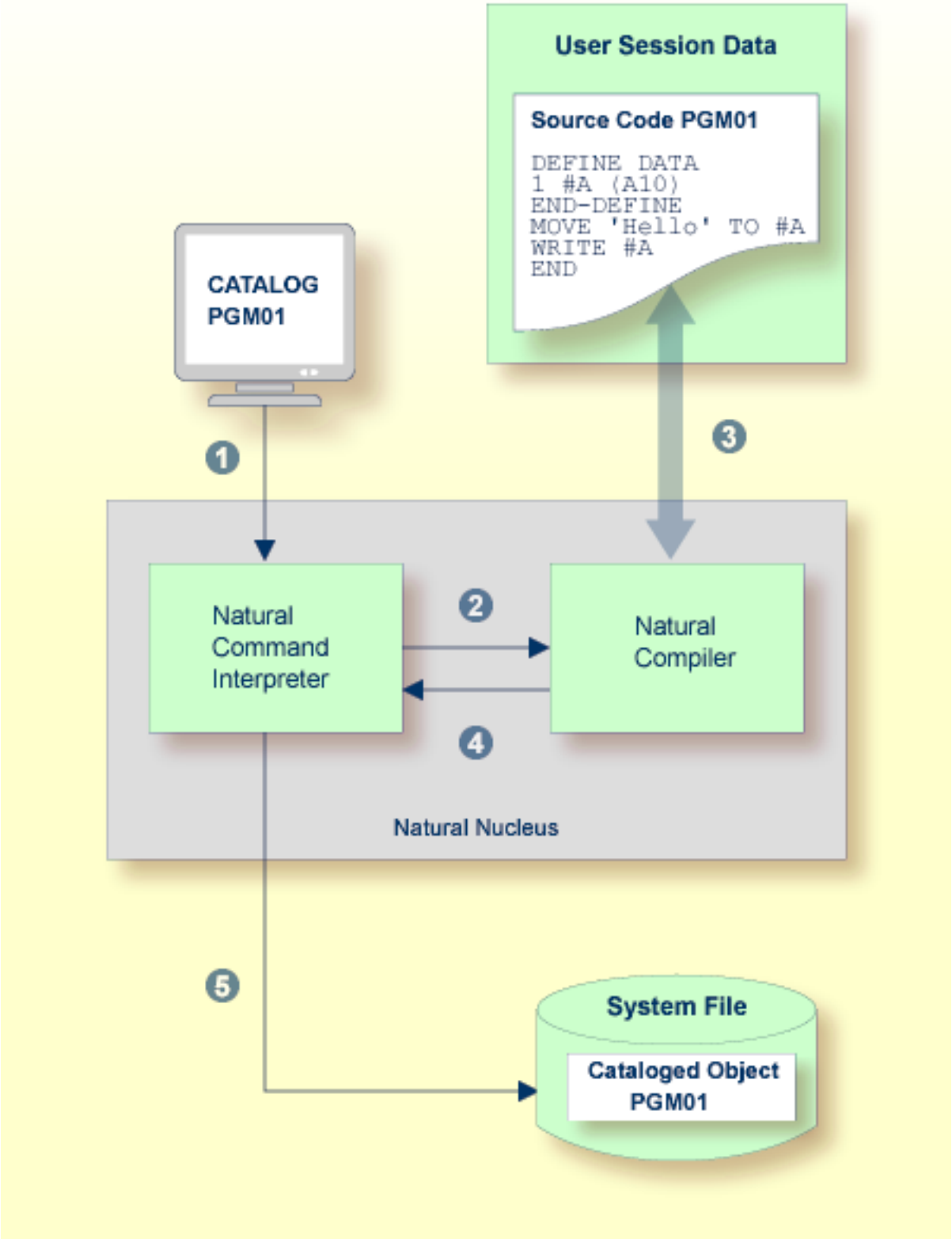
Action	System Command
Compile source code. Perform a syntax check and generate object code. The generated object code is not stored as an object module in a Natural system file.	`CHECK`
Compile source code. If successful, store the generated object code as a **cataloged object** in a Natural system file.	`CATALOG`
Compile source code. If successful, store the generated object code as a **cataloged object** in a Natural system file. Additionally, store the original source code as a separate **source object** in a Natural system file.	`STOW`
Compile source code of a Natural object of the type program. If successful, immediately execute the generated object code. The generated object code is not stored as an object module in a Natural system file.	`RUN`

Related Topics:

- *System Commands* documentation
- *Objects for Natural Application Management - Programming Guide*

Example of Compilation

The diagram below illustrates the process flow when compiling source code with the Natural system command `CATALOG`:


User Session Data
Source Code PGM01
DEFINE DATA
1 #A (A10)
END-DEFINE
MOVE 'Hello' TO #A
WRITE #A
END
CATALOG
PGM01
1
3
Natural
Command
Interpreter
2
Natural
Compiler
4
Natural Nucleus
5
System File
Cataloged Object
PGM01

Legend

1. A user issues the Natural system command `CATALOG PGM01` requesting compilation of the source code currently contained in the source area and storage of the generated object code as a **cataloged object** with the name PGM01.
2. The Natural command interpreter interprets the command `CATALOG` and passes on the request to the Natural compiler.
3. The Natural compiler reads the Natural statements currently contained in the source area, checks the syntax and generates object code.
4. The Natural compiler returns control to the Natural command interpreter.
5. The Natural command interpreter stores the generated object code as a **cataloged object** with the name PGM01 in the current Natural system file.

Natural Command Interpreter

The Natural command interpreter checks and executes user command input from a Natural command prompt.

If Natural Security is installed, command execution can be restricted to a single user or a group of users.

Related Topics:

- *System Commands* documentation
- *Natural Security* documentation

Configuration Settings

Natural parameters manage the configuration of a Natural environment.

Natural parameters are used to standardize and automate development and production processes or adapt standard settings to the needs of individual users. A Natural parameter, for example, is used to set defaults for report creation, define the size of a report or define the size of storage area required such as the source area of an editor.

Most of the characteristics of a Natural environment are predefined by Software AG. The Natural administrator can configure different default environment settings valid for all Natural users. A user can adapt the settings to his needs by overriding default environment settings with a dynamic profile parameter or session parameter.

The section below contains information on:

Profile Parameters

Profile parameters are specified statically or dynamically.

Static parameters are specified in the Natural parameter module, during the installation of Natural. They are used as the default for each Natural session.

Dynamic parameters are specified at the start of a Natural session. You can predefine a set of dynamic parameters with the Natural SYSPARM utility.

Related Topics:

- *Profile Parameters - Parameter Reference* documentation
- *Profile Parameter Usage - Operations* documentation
- *Building a Natural Parameter Module - Operations* documentation
- *SYSPARM Utility - Utilities* documentation

Session Parameters

Session parameters are specified within an active Natural session and/or within a Natural object. The main purpose of session parameters is to control the execution of Natural programs.

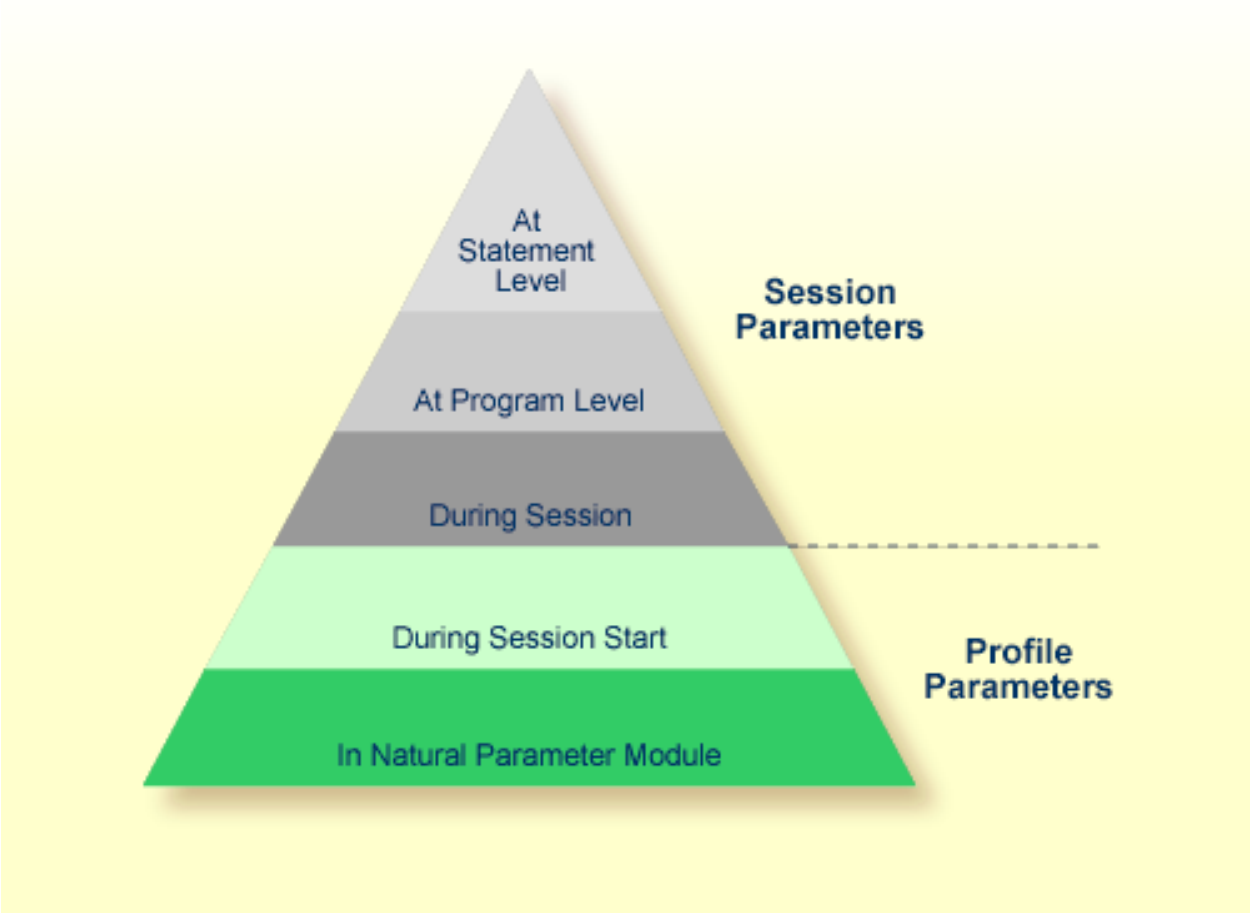
Related Topic:

- *Session Parameters - Parameter Reference* documentation

Parameterization Levels

There is a hierarchical structure of the levels at which Natural parameters can be set. A parameter value set on a higher level overrides the value defined on a lower level. For example, when you specify a parameter dynamically, the new parameter value overrides the static specification as set for the corresponding parameter in the Natural parameter module.

The diagram below illustrates when a parameter can be set and the Natural parameter hierarchy from the lowest level at the base of the pyramid to the highest level at the apex:



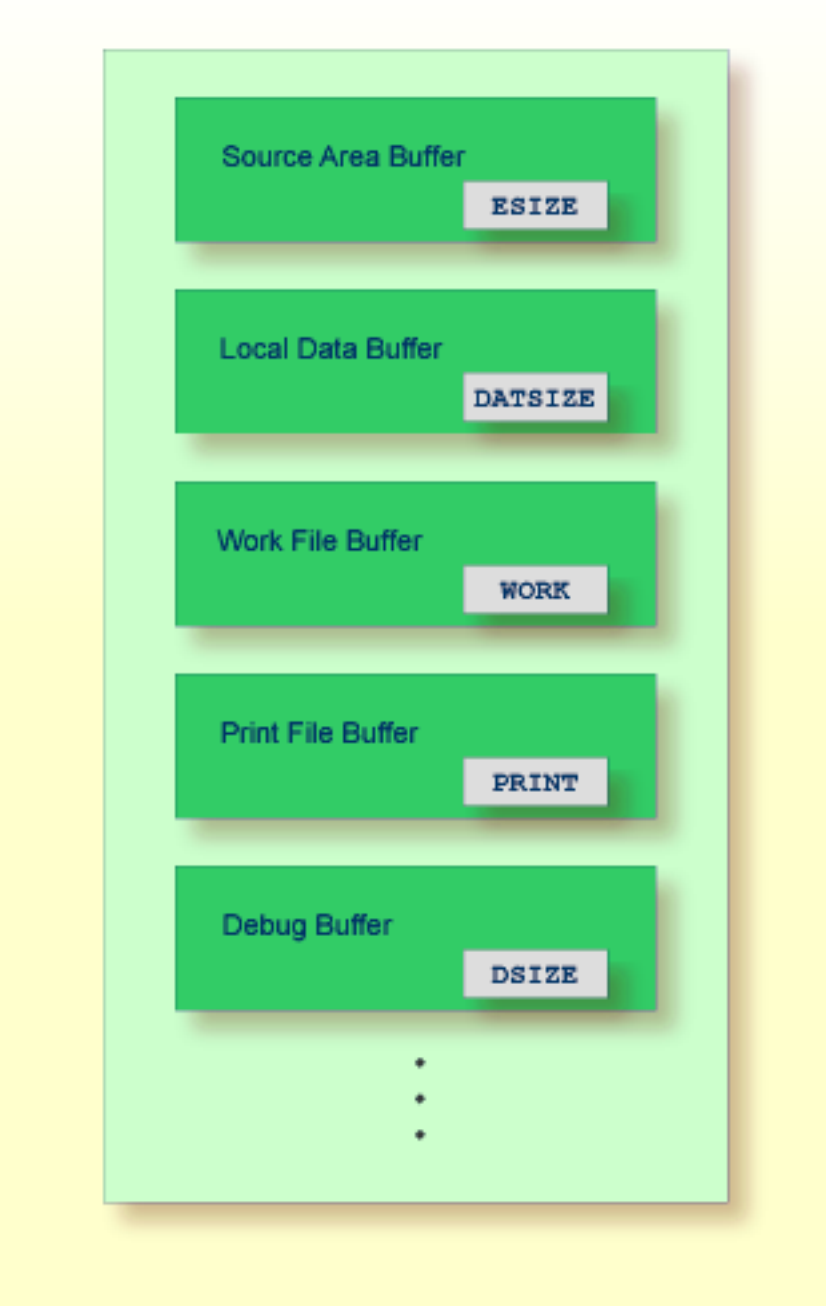
Related Topic:

- *Natural Parameter Hierarchy - Operations* documentation

2 User Session Data

Each Natural user session (online or batch) is assigned a separate work area that contains the data of this session. User session data is saved for the duration of the online or batch session.

The work area comprises a set of Natural buffers. A buffer is a storage unit that holds a particular block of data such as variable data referenced in a Natural program.



As illustrated in the diagram above, the size of a buffer can be specified with a Natural parameter, except for internal or variable Natural buffers. For example, the Natural profile parameter `ESIZE` determines the size of the source area assigned to a Natural editor for editing source code. The profile parameter `DATSIZE` determines the size assigned for holding local data from a Natural program (and subordinate objects if invoked by the program) at execution time.

Natural provides buffer usage statistics with information on the buffers allocated for the current session, their sizes, and the actual buffer space being used; see the relevant documentation sections in *Related Topics* below.

Related Topics:

- *Parameter Reference* documentation
- *Buffer Usage Statistics - BUS*, *General SYSTP Functions*, *SYSTP Utility*, *Utilities* documentation
- *BUS - System Commands* documentation

3 Natural Buffer Pool

The shared-memory Natural buffer pool serves as a temporary storage area into which Natural **cataloged object(s)** (for example, programs) are loaded from a Natural system file for subsequent execution by the Natural runtime system and/or object compilation by the Natural compiler.

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

This section describes the basic principles of buffer pool operation and depicts the object loading procedure.

Object Loading

Object loading into the buffer pool is initialized when an executable Natural object is requested for execution.

When object execution is requested, the respective **cataloged object(s)** are read from the appropriate **Natural system file** and loaded into the buffer pool where they can be executed by the **Natural runtime system**.

Apart from executable Natural objects, non-executable objects of the type data area (local, global, parameter) are also loaded into the buffer pool if a Natural object that references such a data area is cataloged or recataloged (compiled).

Related Topic:

- ***Object Execution*** *- Natural Runtime System, Natural Nucleus*

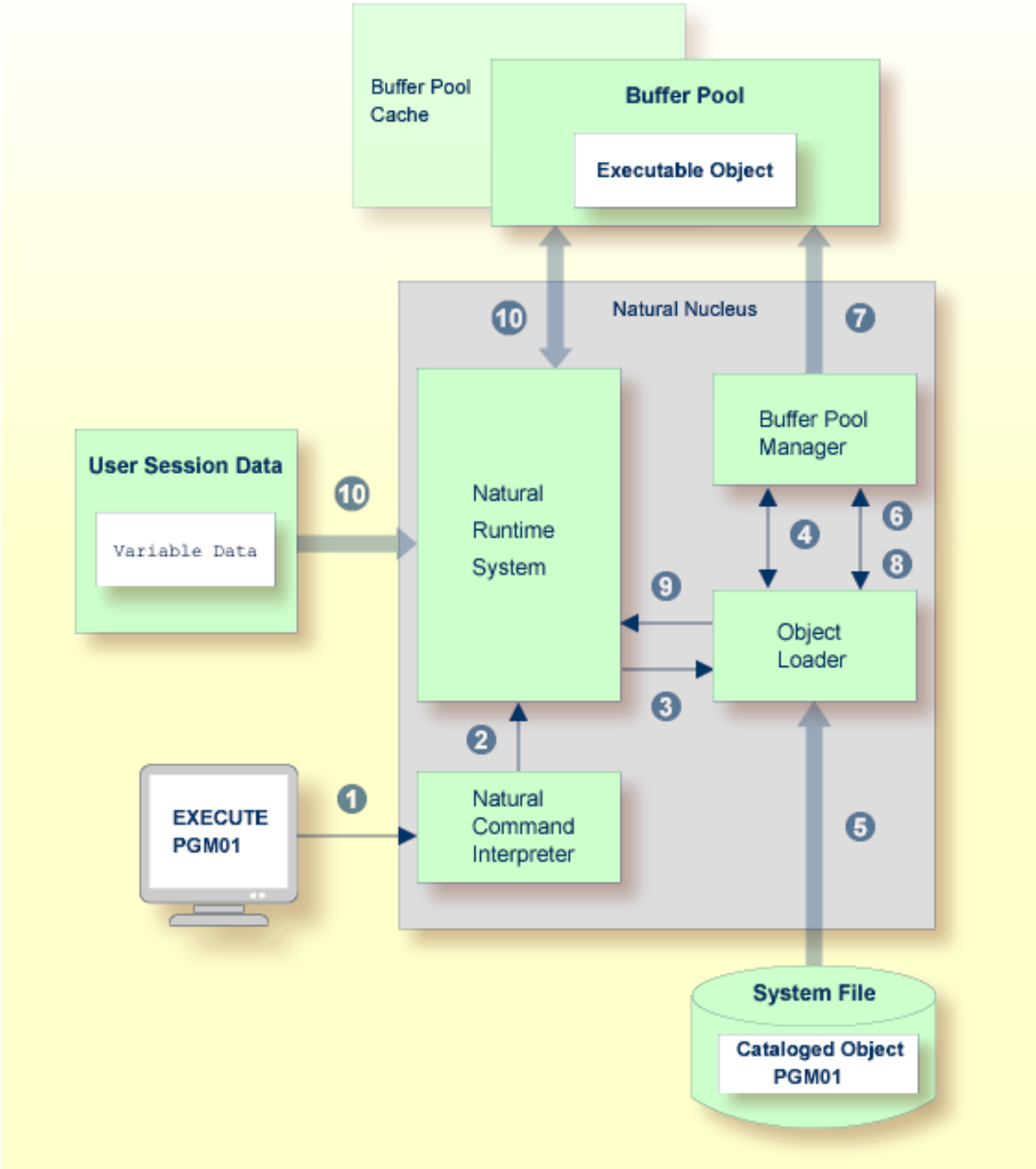
Object Removal

Objects are removed from the buffer pool when they are no longer referenced by a user and space is required for loading new objects. This guarantees maximum use of available storage space in the buffer pool.

If a buffer pool cache exists, unused objects are swapped from the buffer pool into the buffer pool cache. If an unused object is also removed from the buffer pool cache or if no buffer pool cache exists, the respective object is reloaded from the appropriate Natural system file when a user references it again.

Example of Object Loading and Execution

The diagram below illustrates the process flow when loading and executing a Natural program:


Buffer Pool Cache
Buffer Pool
Executable Object
Natural Nucleus
10
7
User Session Data
Variable Data
10
Natural Runtime System
Buffer Pool Manager
4
6
8
9
Object Loader
3
2
EXECUTE PGM01
1
Natural Command Interpreter
5
System File
Cataloged Object PGM01

Legend

1. A user issues the Natural system command `EXECUTE`, requesting execution of a Natural object of the type program with the name PGM01.
2. The Natural command interpreter interprets the command and passes the request to the Natural runtime system.
3. The Natural runtime system passes the request to the object loader.
4. The object loader instructs the buffer pool manager to search the buffer pool (and the buffer pool cache, if available) for PGM01 and determine the address at which the object is stored in the buffer pool.

   If the buffer pool manager finds the buffer pool address of PGM01, the buffer pool passes the address to the object loader and execution continues with 8.
   If the buffer pool manager does not find the buffer pool address of PGM01, the buffer pool manager returns the negative search result to the object loader and object execution continues with 5.
5. The object loader retrieves the cataloged object of PGM01 from the appropriate Natural system file.
6. The object loader passes the cataloged object of PGM01 to the buffer pool manager.
7. The buffer pool manager loads PGM01 into the buffer pool.
8. The buffer pool manager returns the buffer pool address of PGM01 to the object loader.
9. The object loader passes the buffer pool address of PGM01 to the Natural runtime system.
10. The Natural runtime system interprets and executes the compiled code of the cataloged object of PGM01.

Related Topics

The following topics in the *System Architecture* documentation refer to the Natural buffer pool:

- ***Natural System Files***
- ***Natural Runtime System*** - *Natural Nucleus*
- ***Cataloged Objects*** - *Natural Compiler, Natural Nucleus*

The following topics in other Natural documentation refer to the Natural buffer pool:

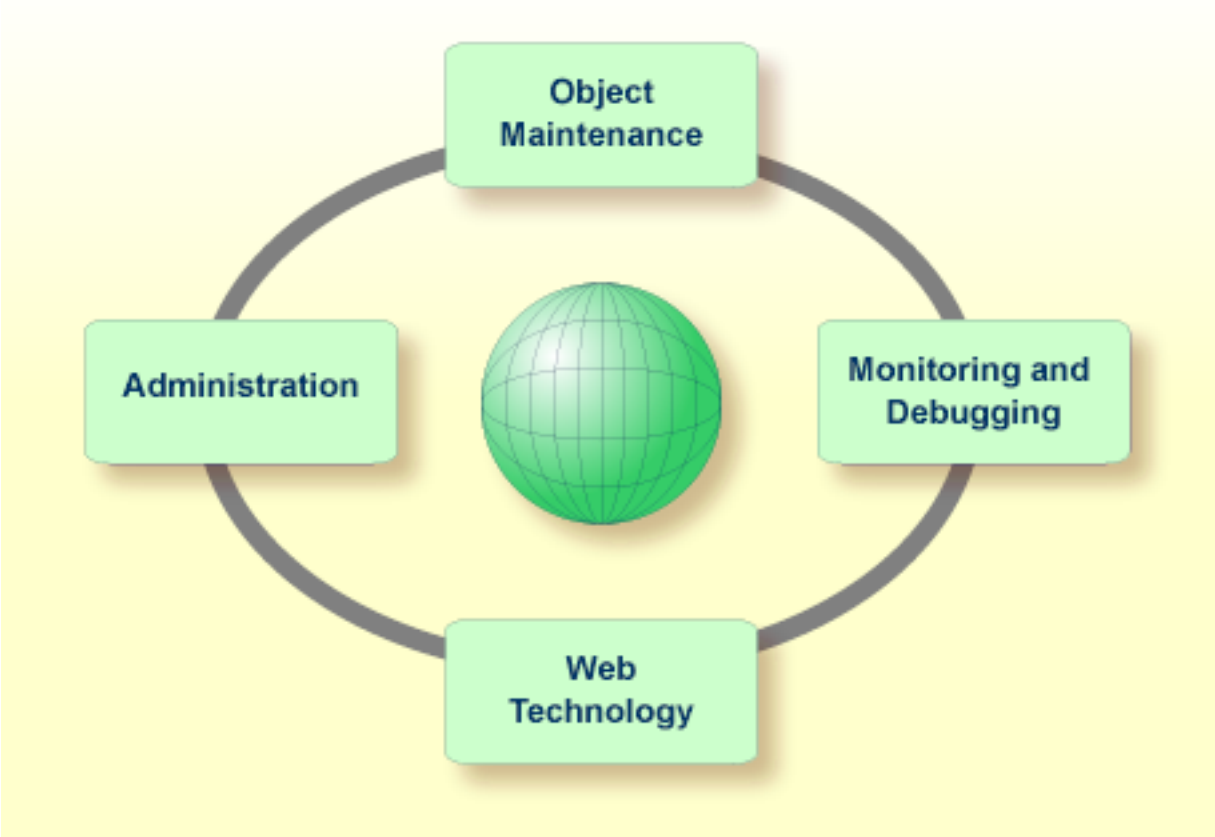
- *Natural Buffer Pool - Operations*

- *Natural Storage Management - Operations*

4 Natural Editors and Utilities

Natural utilities are tools that provide a set of functions, which are used in order to maintain an application development, administration or operations environment.

This diagram below shows the major functional purposes into which Natural editors and utilities can be grouped:



A Natural editor or utility can be used for the following purposes:

- To handle Natural objects (or non-Natural objects) such as programs, subprograms and data definition modules;
- To perform administration functions such as transferring applications from one hardware platform to another, controlling remote procedure calls, generating reports and maintaining application-specific error messages;
- To monitor and debug applications;
- To use web technology such as accessing a web server or processing XML documents.

Related Topics:

- *Objects for Natural Application Management - Programming Guide*
- *Editors* documentation
- *Utilities* documentation
- *Natural Web Interface - Web Technology* documentation
- *XML Toolkit - Web Technology* documentation

5 TP/OS Interface

Natural provides interfaces that allow the Natural nucleus to access a TP monitor for online transaction processing and an operating system (OS) for batch processing.

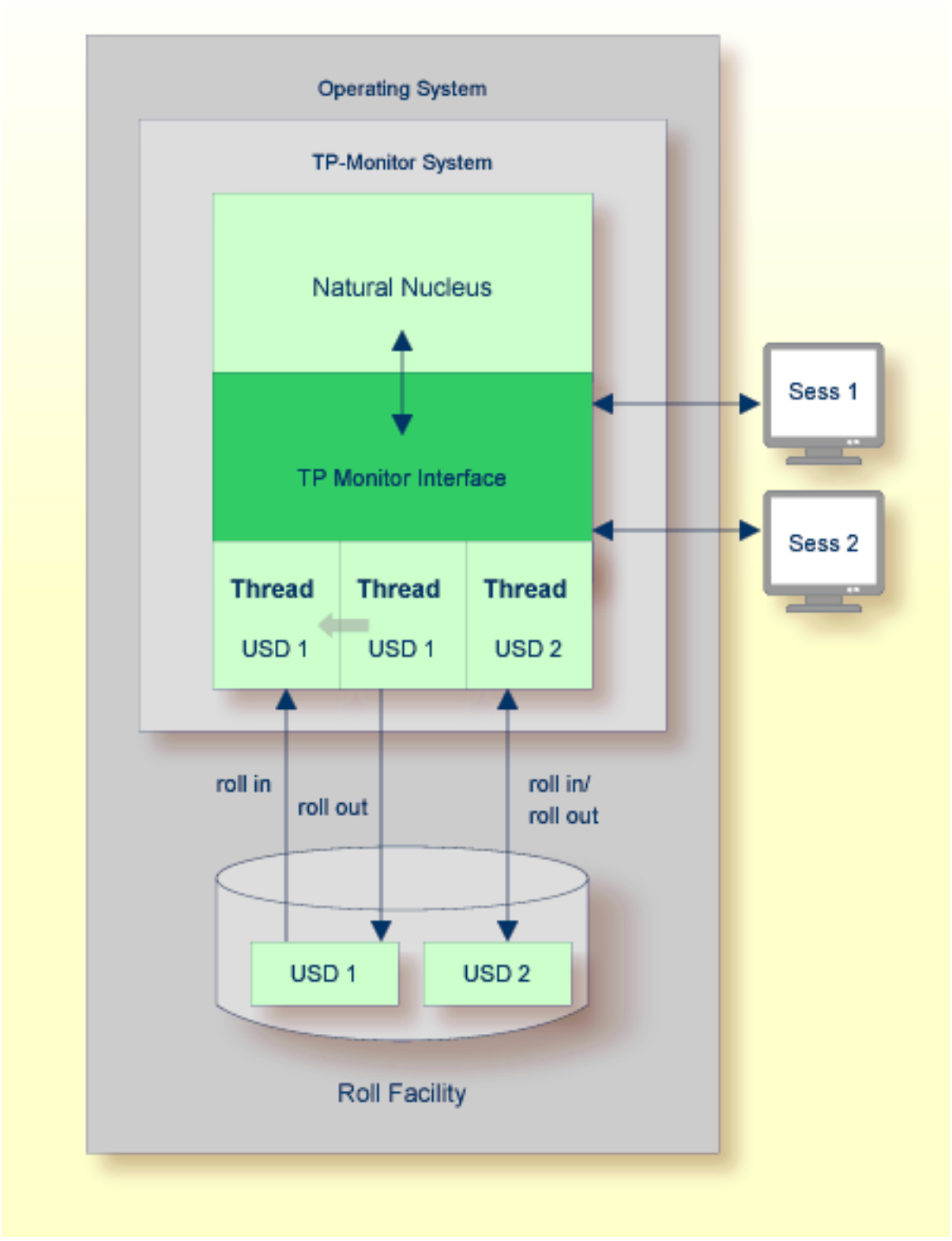
Online Processing

In a TP monitor environment, Natural operates as a standard TP program. It follows the rules that apply to programs executing under the control of this TP monitor.

In a TP monitor environment, the work area that contains **user session data** is referred to as the Natural thread. It is maintained for the duration of the user session.

A Natural thread can be rolled out into a roll facility while a session is waiting for terminal input from the user. When rolling out threads, storage is freed for other user sessions. The Natural thread is rolled in when the user provides terminal input, for example, when he presses ENTER. A thread can be relocated to another storage area as indicated in the diagram below by USD1 in the thread that contains user session data (USD) of the user session Sess 1.

The diagram below is an example of how a TP monitor (here: Com-plete) manages storage allocation by rolling Natural threads into or out of a roll facility:

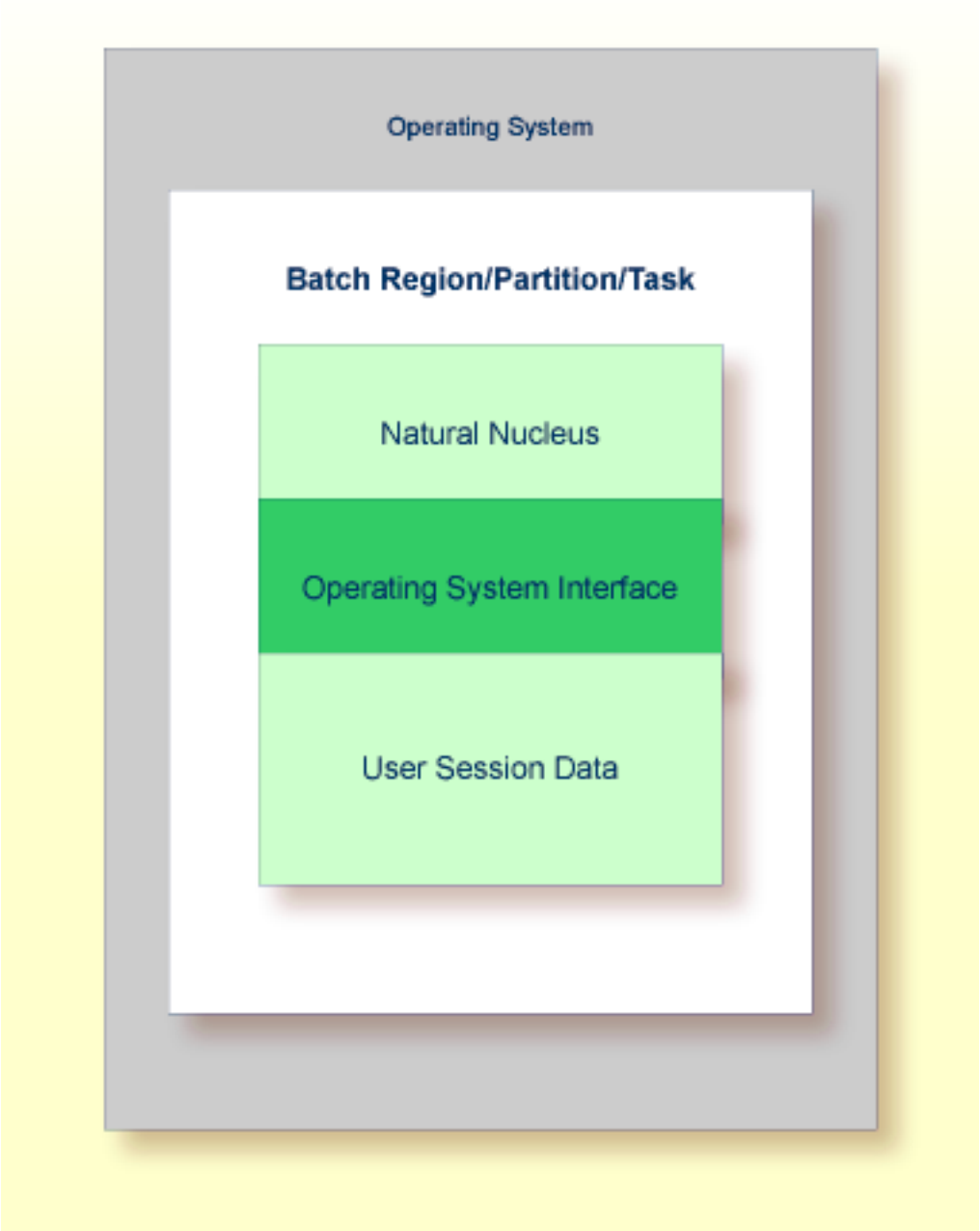

Operating System
TP-Monitor System
Natural Nucleus
TP Monitor Interface
Thread
Thread
Thread
USD 1
USD 1
USD 2
Sess 1
Sess 2
roll in
roll out
roll in/
roll out
USD 1
USD 2
Roll Facility

Batch Processing

In batch mode, Natural processes a session that has been initialized by a batch job. No interaction between the computer and the user who submitted the batch job is necessary.

A batch job consists of programs that are executed serially and that receive sequential input data. Input data is read from a file and output is written to a file. Depending on the operating system, the work area established for a batch job is contained in a batch region (under z/OS), a partition (under z/VSE) or a task (under BS2000/OSD). The work area contains user session data that is maintained for the duration of the batch user session.

The schematic diagram below illustrates a batch processing environment:


Operating System
Batch Region/Partition/Task
Natural Nucleus
Operating System Interface
User Session Data

Natural TP/OS Interfaces Supplied

For information on the TP monitor interface supplied by Natural and using Natural with a TP monitor, refer to the appropriate sections in the *TP Monitor Interfaces* documentation:

- *Using Natural with TP Monitors*
- *Natural under CICS*
- *Natural under Com-plete/SMARTS*
- *Natural under IMS TM*
- *Natural under TIAM*
- *Natural under TSO*
- *Natural under openUTM*

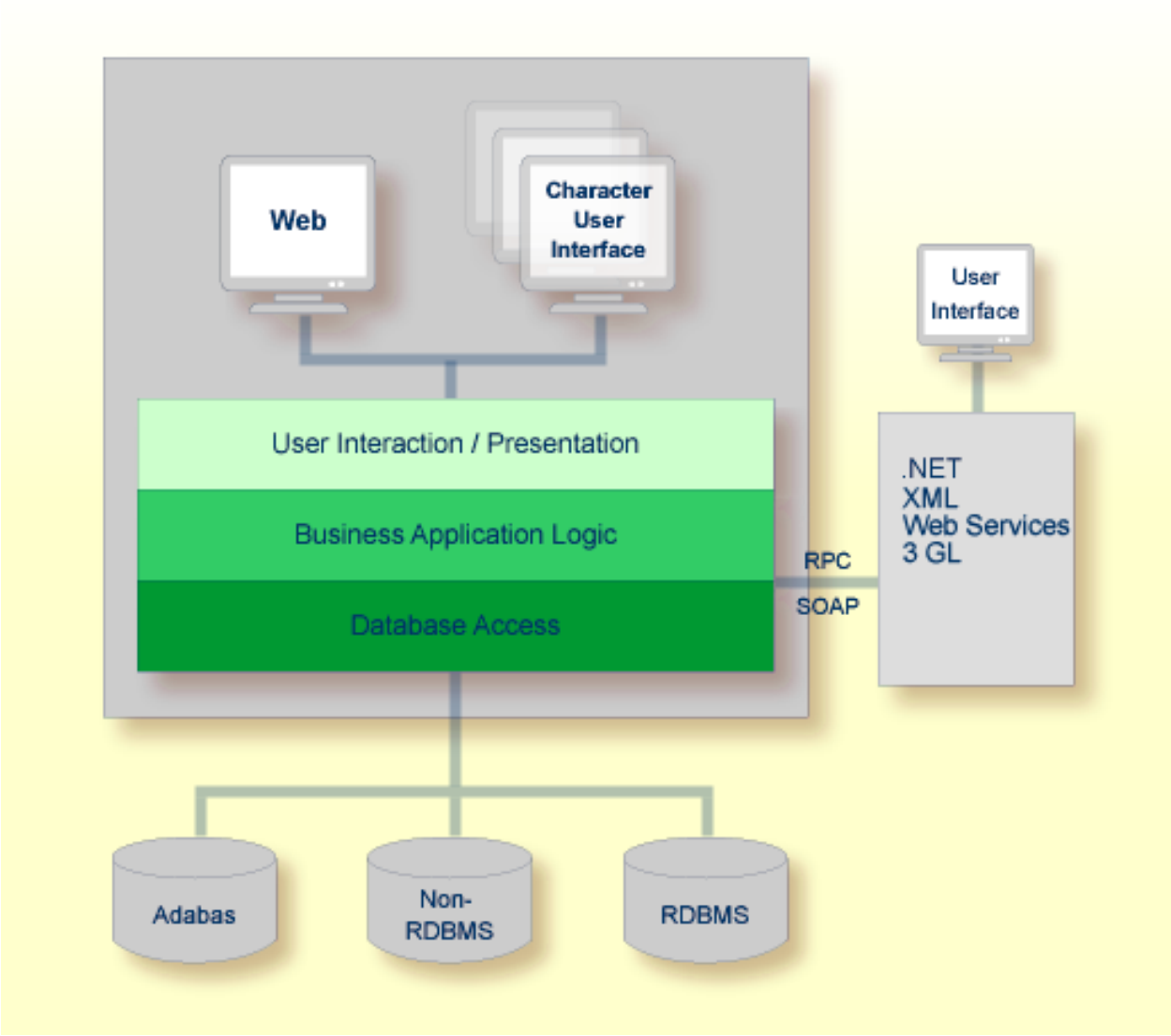
For information on running Natural in batch mode, refer to the section *Natural in Batch Mode* in the *Operations* documentation.

6 User Interface

Natural supports distributable applications with multiple-layer architecture: presentation, business application logic and database access. A Natural application is modular and provides the appropriate types of object required for each application layer.

Natural allows programming of a variety of user interfaces: web-based interfaces (for example, HTML or XML), process-driven applications with character user interfaces (CUIs) and event-driven graphical user interfaces (GUIs).

In addition, a Natural application can interact with user interfaces that are connected to non-Natural environments. Connections between Natural and non-Natural environments are established, for example, by using a remote procedure call (RPC) or a SOAP request that allows programs on a client computer to run a Natural object of the type subprogram on the Natural RPC server.


Web
Character User Interface
User Interface
User Interaction / Presentation
Business Application Logic
Database Access
RPC
SOAP
.NET
XML
Web Services
3 GL
Adabas
Non-RDBMS
RDBMS

7 Print Files - Work Files

Print files and work files are data containers for permanent or intermediate storage of data processed in Natural objects. These data containers are, for example, physical sequential files, members of partitioned data sets or tape data sets provided by either the operating system or the TP monitor. Natural accesses print files and work files sequentially.

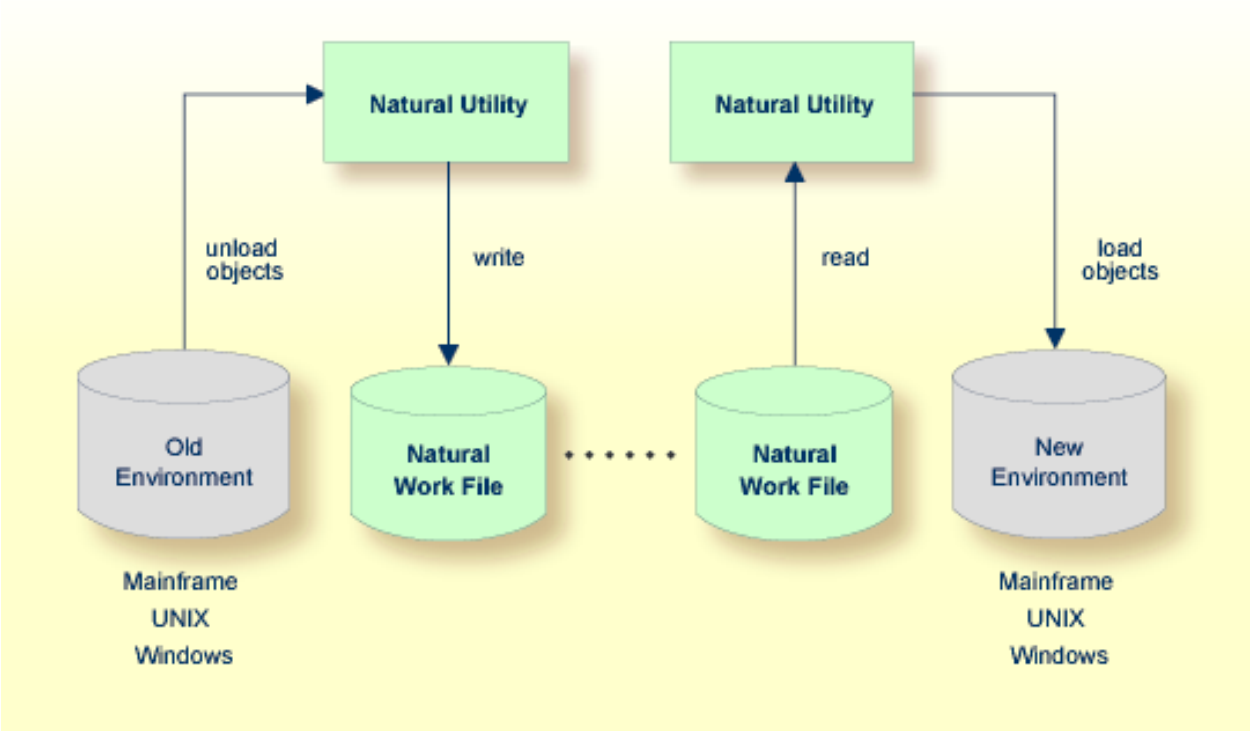
A print file is used to store report data and printer control characters required to output a report on a printer.

A work file is used to store data that can be exchanged between Natural objects or between a Natural object and an object written in another programming language such as COBOL.

Print files and work files can be used for batch or online processing following the conventions of the operating system or TP monitor installed.

Transferring Objects with Work Files

The diagram below illustrates how data contained in a work file can be used to transfer Natural objects between different Natural environments on different platforms. This is done by using a Natural utility (for example, the Object Handler) that unloads objects in the source environment into a work file and loads them from the work file into the target environment. If required, an application protocol such as FTP is used for transferring work files from source to target environments.



Defining and Accessing Print and Work Files

Print files and work files are logically defined for a Natural environment and can be physically assigned to a file or a printer by using a Natural parameter and/or control statements of the underlying operating system or TP monitor. Assignments can be changed for the current session during runtime.

Data is written to and read from a work file or written to a print file by using the appropriate Natural statements.

8 Natural System Files

A Natural system file stores Natural objects and non-Natural objects that belong to either user-defined applications or Natural system applications (for example, utilities) supplied by Software AG.

Natural objects stored in a system file include **cataloged objects** and **source objects** as described in Natural Compiler in the section Natural Nucleus. Natural system files are stored in database file or storage systems such as Adabas and VSAM. Depending on the system environment, Natural system files can reside in different database file or storage systems.

A Natural system file is accessed when a user reads or modifies a Natural object. In addition, a system file is accessed when an object is loaded into the buffer pool for subsequent execution (see the section ***Natural Buffer Pool***).

Types of System File

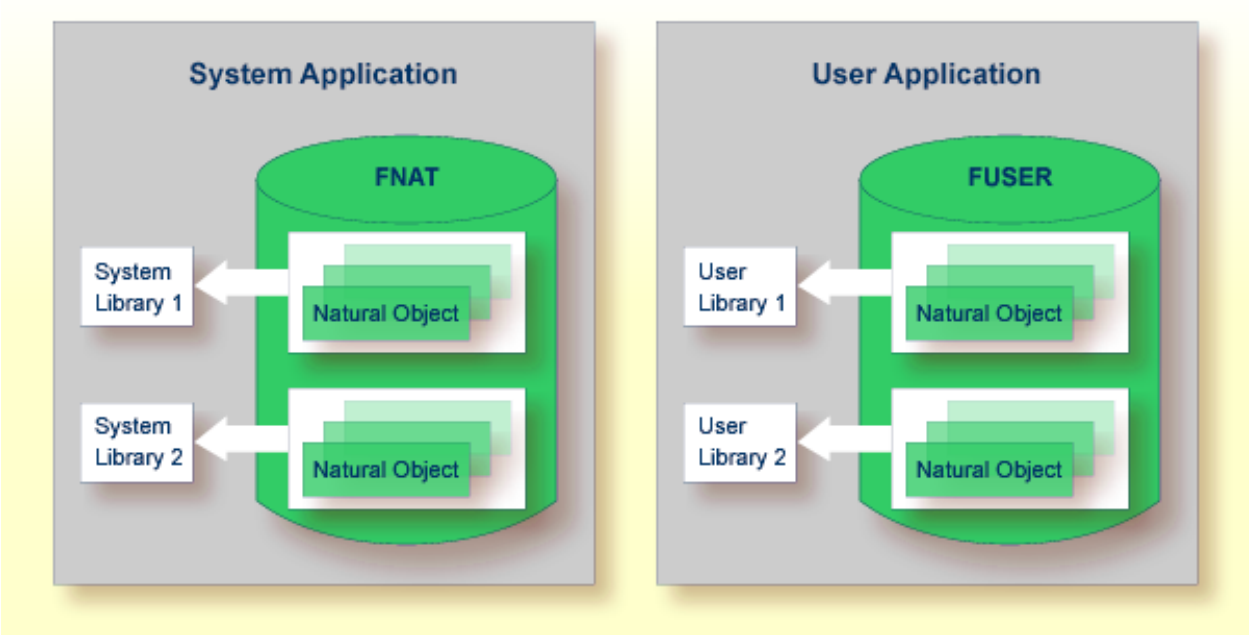
The table below lists and describes the Natural system files that are usually available in a Natural environment. The availability of the system files and the data contained in the files depends on the Software AG products installed in addition to base Natural.

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

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

Libraries in System Files

Natural objects contained in the Natural system files FNAT and FUSER are grouped into logical constructs called libraries as illustrated in the diagram below:



Related Topics:

- ***Cataloged Object*** and ***Source Object*** - *Natural Compiler, Natural Nucleus*
- *Natural Security* documentation
- *Natural Advanced Facilities* documentation
- ***Natural Data Definition Modules*** - *DBMS Interface - Database Access*
- *Predict* documentation
- *Natural Development Server* documentation

9 DBMS Interface - Database Access

Natural supplies DBMS (database management system) interfaces to access data stored in an Adabas database, a relational database management system (RDBMS) and/or a non-RDBMS.

Natural applications can access user data in multiple DBMSs concurrently. Data stored in a DBMS can be accessed randomly as opposed to **print files and work files** (see the relevant section), which are accessed sequentially.

Supported DBMSs include DB2, DL/I and VSAM data sets.

This section provides information on the DBMSs supported by Natural and the principles of accessing data in a database from within a Natural application.

Database Management Systems Supported

Natural supplies interfaces for the following database management systems (DBMSs):

- **Adabas**
- **DL/I**
- **DB2**
- **VSAM**

Adabas

The Natural for Adabas interface provides access to data stored in an Adabas database.

Adabas is Software AG's database management system. Adabas supports relational as well as nested relational data structures.

The Natural for Adabas interface is an integrated part of Natural that allows access to Adabas databases on local computers. For remote access, additional routing and communication software is required such as Software AG's Entire Net-Work. In any case, the type of the host machine running the Adabas database is always transparent to Natural.

Related Topics:

- *Accessing Data in an Adabas Database - Programming Guide*
- *Adabas* documentation

DL/I

The Natural for DL/I interface provides access to data stored in an IMS DB database.

DL/I is IBM's data manipulation language of IMS DB. IMS is a database and transaction management system from IBM.

Related Topic:

- *Natural for DL/I* documentation

DB2

The Natural for DB2 interface provides access to data stored in IBM's DB2 UDB for z/OS operating systems.

Related Topic:

- *Natural for DB2* documentation

VSAM

The Natural for VSAM interface provides access to data stored in VSAM data sets.

VSAM is an IBM access method to maintain records of different data set organizations: key-sequenced data sets, entry-sequenced data sets or relative-record data sets.

Related Topic:

- *Natural for VSAM* documentation

Natural Data Manipulation Language

Natural data manipulation language (DML) provides a common data access syntax across all DBMSs supported by Natural. Natural DML allows Natural objects to access different DBMSs by using the same language statements (DML statements) such as `FIND`, `READ`, `STORE` and `DELETE`.

Natural determines the DBMS from its configuration files and translates the DML statements into database-specific commands; that is, Natural generates direct commands for Adabas databases, SQL statement strings and host variable structures for RDBMSs using SQL.

A Natural object using a DML statement that does *not* contain a database-specific clause can be executed against different DBMSs. A DML statement that contains a database-specific clause must be changed before it can be used for a different DBMS. Database-specific considerations are described in the *Statements* documentation.

Related Topic:

- *Database Access and Update - Statements* documentation

Special SQL Statements

In addition to DML statements, Natural provides a special set of SQL statements for exclusive use in conjunction with RDBMSs. SQL statements include `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `COMMIT` and `ROLLBACK`. SQL-specific statements are described in the *Statements* documentation.

Flexible SQL and facilities for working with stored procedures complete the set of special SQL statements. Flexible SQL allows arbitrary SQL syntax.

Related Topics:

- *SQL Statements - Statements* documentation
- *Database Access and Update - Statements* documentation

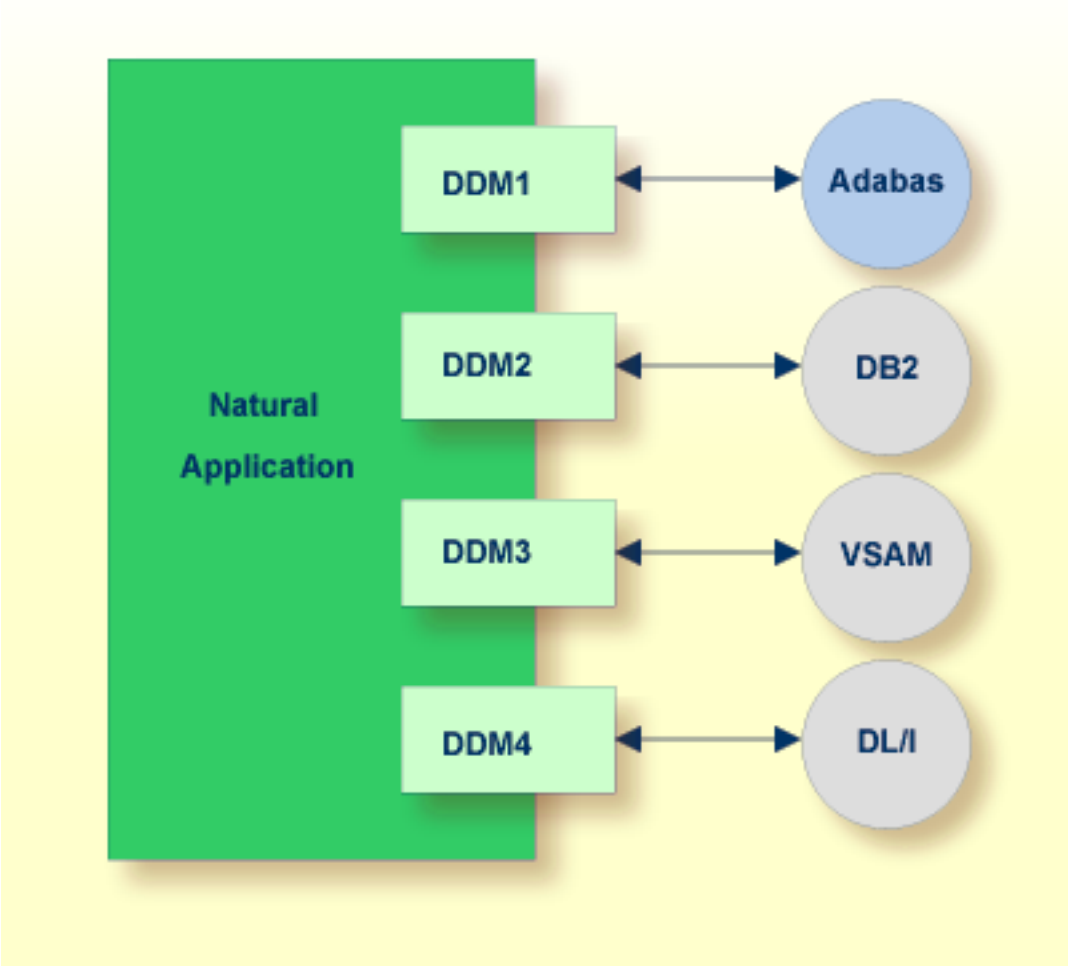
Natural Data Definition Modules

Natural provides an object called data definition module (DDM) that allows convenient and transparent access to database files of different DBMSs.

A DDM constitutes a logical view of a physical database file. The DDM contains information on the individual fields of the file - information which is relevant for the use of these fields in a Natural object.

A DDM establishes the connection between Natural data structures and the data structures in the DBMS to be used. Such a database structure might be a table in an RDBMS, a file in an Adabas database or a VSAM data set. Hence, the DDM hides the real structure of the accessed database from the Natural application.

The diagram below shows how Natural can access multiple databases from within a single application by using references to DDMs that represent the specific data structures in the specific DBMS:


Natural
Application
DDM1
DDM2
DDM3
DDM4
Adabas
DB2
VSAM
DL/I

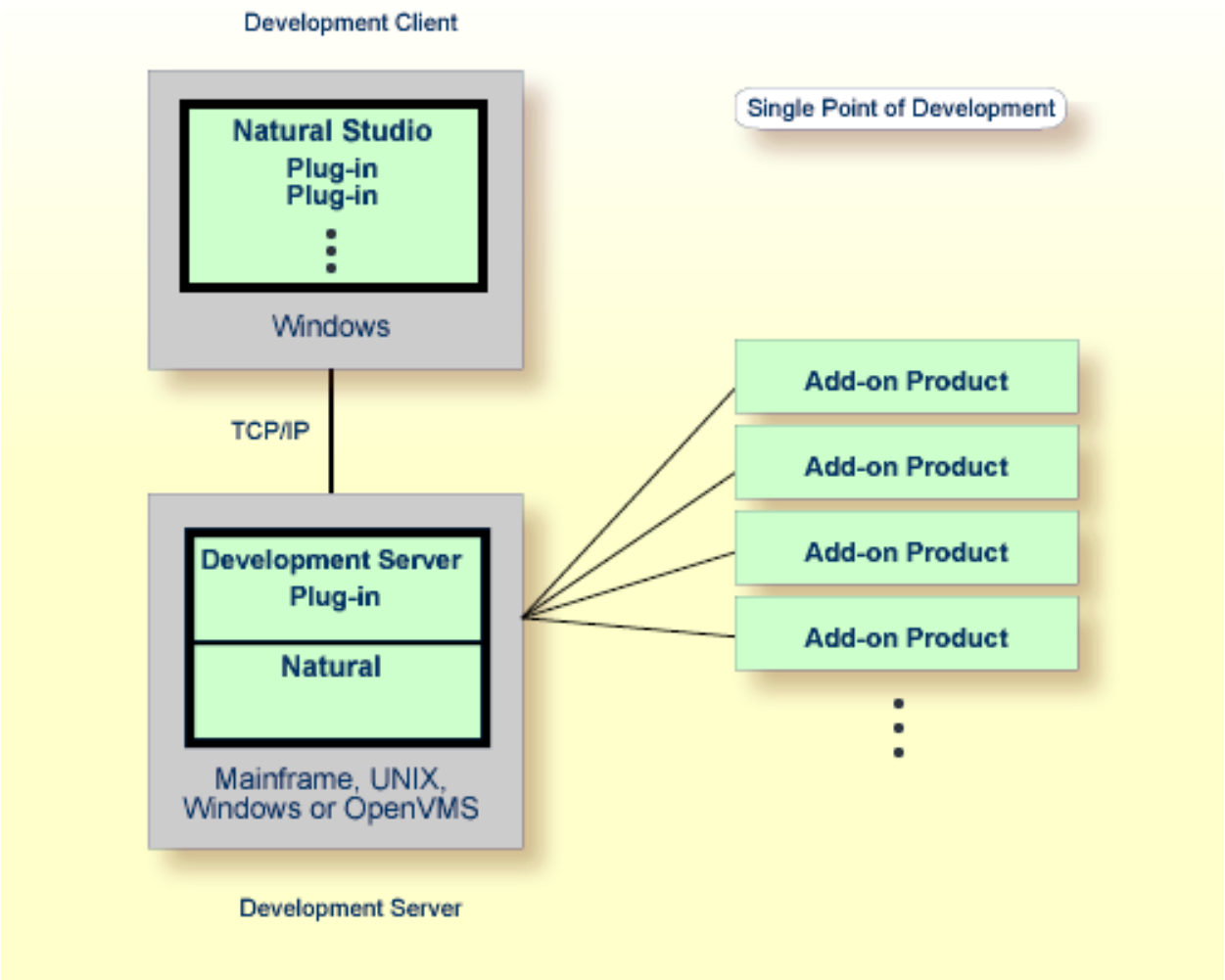
II Natural SPoD Architecture

10 Natural SPoD Architecture

This section describes the system architecture of Natural's Single Point of Development (SPoD).

SPoD allows centralized application development from a single Windows environment. You can use the features of the Natural Studio (provided with Natural for Windows) to develop and test Natural applications in a remote environment located on a mainframe, a UNIX or an OpenVMS platform.

SPoD is based on a client/server concept that allows one single development environment for all platforms. The diagram below illustrates this concept and the major components of the SPoD architecture:



Development Client

Natural for Windows serves as the remote development desktop client for the target environment on a mainframe, a UNIX or an OpenVMS platform. The desktop client includes Natural Studio, which is the central workstation where users design applications. All Natural-related functions required for remote development can be performed from within Natural Studio.

Development Server

Natural Development Server plug-in allows remote development for the Natural installation in the target environment on a mainframe, a UNIX, a Windows or an OpenVMS platform. Natural on the target platform plus Natural Development Server plug-in constitute the development server.

Add-on Product

Natural Studio provides plug-ins that can be used to integrate one or more Natural add-on products (for example, Predict) into a SPoD environment. The installation of the respective add-on product(s) in the development server environment is a prerequisite for Natural Studio plug-ins.

Security

You can use Natural Security to protect the Natural Development Server environment, and Natural base applications and compound applications. For further information, refer to the *Natural Security* documentation.

Related Topic:

- *Natural Single Point of Development* documentation