Natural

Unicode and Code Page Support

Version 9.3.2

April 2026

ADABAS & NATURAL

This document applies to Natural Version 9.3.2 and all subsequent releases.

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

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

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

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

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

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

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

Document ID: NATWIN-NNATUNICODE-932-20260408

Table of Contents

Preface v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Introduction to Unicode and Code Page Support 5
About Code Pages and Unicode 6
About Unicode and Code Page Support in Natural 7
3 Enabling Unicode and Code Page Support 9
ICU Library 10
4 Configuration and Administration of the Unicode and Code Page Environment 11
Profile Parameters 12
Encoding Information 13
Deploying Natural Objects with Encoding Information 13
5 Development Environment 15
Development Environment for Applications 16
Customizing Your Environment 17
Editors in the SPoD Environment 18
6 Unicode and Code Page Support in the Natural Programming Language 21
Natural Data Format U for Unicode-Based Data 22
Statements 23
Logical Condition Criteria 27
System Variables 28
Large and Dynamic Variables 28
Session Parameters 28
Sample Programs 31
7 Unicode Input and Output Handling in Natural Applications 33
Displaying and Entering Unicode Data 34
Natural Web I/O Interface Client 35
8 Bidirectional Language Support 39
General Information 40
Screen Direction 40
Field Direction 41
Maps and Dialogs 43
Print Methods 43
Terminal Capabilities 44
9 Double-Byte Character Support 45
10 Unicode Data Storage 47
Unicode Data and Parameter Access 48
Database Management System Interfaces 48
Work Files and Print Files 49
11 Platform Differences 53
General Information 54

Windows 54
Linux 55
12 Migrating Existing Applications 57
Impact of Unicode on Existing Applications 58
Migrating Existing Objects 58
Adding Unicode Support to Existing Applications 59
Migrating Natural Remote Procedure Calls (RPC) 60
13 Special Considerations and Limitations 61
14 Help and Troubleshooting 63
Receiving the Startup Error "Invalid Code Page Specified" 64
The Default Code Page 64
Picking the Right Format When Saving Your Natural Sources 64
Handling UTF-8 Encoding with Natural Code 64
Incorrectly Displayed Characters 65
Receiving an Error When Editing a Natural Source 65
Receiving an Error When Saving a Natural Source 65
Finding out the Encoding of a Natural Source 65
Changing the Encoding of a Natural Source 66
Converting an Existing Natural Source into UTF-8 Format 66
Substitution Characters Used When a Character Cannot Be Converted 66
Using UTF-8 Sources with Previous Natural Versions 66
Receiving a Conversion Error When Cataloging a Source Which Has UTF-8 Format 67
Receiving Junk on Linux When Displaying U Format by a Terminal Emulation 67
Working with a Current SPoD Client and an Older SPoD Server 67
Working with a Current SPoD Server and an Older SPoD Client 67

Preface

This documentation describes how Natural supports Unicode and code pages on Windows and Linux platforms. It also describes how Natural supports bidirectional languages and double-byte characters.

This documentation is organized under the following headings:

Introduction	General information on code pages and the Unicode Standard, and on how Unicode and code pages are supported in Natural.
Enabling Unicode and Code Page Support	Information on the ICU library.
Configuration and Administration of the Unicode/Code Page Environment	Information on profile parameters which provide Unicode and code page support, and on the encoding of code page data.
Development Environment	How to customize your environment and how Unicode is handled by the Natural editors.
Unicode and Code Page Support in the Natural Programming Language	Information on the U format and on statements, logical condition criteria, system variables, large and dynamic variables, and session parameters which provide Unicode and code page support.
Unicode Input/Output Handling in Natural Applications	How to display and enter Unicode data. Information on the Natural Web I/O Interface client which is used in SPoD and runtime environments.
Bidirectional Language Support	How Natural supports bidirectional languages.
Double-Byte Character Support	How Natural supports double-byte characters.
Unicode Data Storage	Information on database access, and on the work file types and print files which provide Unicode and code page support.
Platform Differences	Handling differences on Windows and Linux platforms.
Migrating Existing Applications	About the impact of Unicode on existing applications. How to migrate existing objects, add Unicode support to existing applications, and how to migrate Natural remote procedure calls (RPC).
Special Considerations and Limitations	Important information and restrictions.
Help and Troubleshooting	Answers to frequently asked questions and help regarding frequently received errors.

1 About this Documentation

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

Document Conventions

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

Online Information and Support

Product Documentation

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

Product Training

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

Tech Community

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

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

Product Support

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

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

Data Protection

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

2 Introduction to Unicode and Code Page Support

- About Code Pages and Unicode 6
- About Unicode and Code Page Support in Natural 7

About Code Pages and Unicode

A traditional code page is a list of selected character codes, arranged in a certain order, that support specific languages or groups of languages that share common scripts. A code page can contain a maximum of 256 character codes. For character sets which contain more than 256 characters (for example, Chinese or Japanese), double-byte code unit handling (DBCS) is used: DBCS code pages are actually multi-byte encodings, a mix of 1-byte and 2-byte code points.

Code pages have the inherent disadvantage of not being able to be used to store different languages in the same data stream. Unicode was designed to remove this restriction by providing a standard encoding for all character sets which is independent of the platform, program, or language used to access the data. With Unicode, a unique number is provided for every character.

A single number is assigned to each code element defined by the Unicode Standard. Each of these numbers is called a “code point” and, when referred to in text, is listed in hexadecimal form following the prefix "U". For example, the code point "U+0041" is the hexadecimal number "0041" (equal to the decimal number "65"). It represents the character "A" in the Unicode Standard which is named “LATIN CAPITAL LETTER A”.

The Unicode Standard defines three encoding forms that allow the same data to be transmitted in a byte, word or double word oriented format. A “code unit” is the minimal bit combination that can represent a character in a specific encoding. The Unicode Standard uses 8-bit code units in the UTF-8 encoding form, 16-bit code units in the UTF-16 encoding form, and 32-bit code units in the UTF-32 encoding form. All three encoding forms encode the *same* common character repertoire and can be efficiently transformed into one another without loss of data.

In the context of Natural, we are concerned with two of these encoding forms: UTF-16 and UTF-8. Natural uses UTF-16 for the coding of Unicode strings at runtime and UTF-8 for the coding of Unicode data in files. UTF-16 is an endian-dependent 2-byte encoding; the endian format that will be used depends on the platform. UTF-8 is a variable-length encoding.

For a complete description of Unicode, see the Unicode consortium web site at ***http://www.unicode.org/***.

Note: For obtaining information on Unicode code points, you can use the `SYSCP` utility which is available with Natural for Windows.

About Unicode and Code Page Support in Natural

For Unicode support, the Natural data format U and specific statements, parameters and system variables are used. For details, see the remainder of this documentation.

Most existing data is available in code page format. When converting this data to Unicode, it is required that the correct code page is used. Natural provides the possibility to define the correct code page on several levels:

- The system code page is used if a default code page is not defined in Natural.

  On the platforms supported by Natural for Linux, check whether the detected system code page meets your expectations. For more information, see the description of the Natural parameter `CP`.
- The default code page is used when the Natural parameter `CP` is defined; this overwrites the operating system's code page.
- The object code page which is defined, for example, for a source overwrites the default code page for this object.

When using Unicode strings and code page strings in one application, Natural performs implicit conversions where necessary (for example, when moving or comparing data). Explicit conversions can be performed with the statement `MOVE ENCODED`.

In most cases, existing applications which do not require Unicode support, will run unchanged. Changes can be necessary if the existing sources are encoded in different code pages. For more information, see *Migrating Existing Applications* later in this documentation.

It is not possible to run an existing application and also support Unicode data without any changes to the application. The Natural data format U has to be introduced in the application and it will most probably not suffice to simply replace the A format definitions with U format definitions. All code which assumes a specific memory layout of strings (for example, `REDEFINE` from alpha-numeric to numeric format) has to be adapted.

Unicode characters are not permitted within variable names, object names and library names.

Unicode-based data are supported for Adabas.

Natural uses the International Components for Unicode (ICU) library for Unicode collation and conversion. For more information, see *http://userguide.icu-project.org/*. See also *ICU Library* later in this documentation.

3 Enabling Unicode and Code Page Support

- ICU Library 10

ICU Library

The ICU libraries are always installed with the full set of ICU conversion and collation data. The settings in the configuration file *NATCONV.INI* apply to the A format. For the U format, the corresponding checks (for example, when a character is translated to upper case) are done via the ICU library.

Note: For obtaining information on the ICU version and the supported code pages, you can use the `SYSCP` utility which is available with Natural for Windows.

4 Configuration and Administration of the Unicode and Code Page Environment

- Profile Parameters 12
- Encoding Information 13
- Deploying Natural Objects with Encoding Information 13

Notation *vr*:

When used in this document, the notation *vr* represents the 2-digit ICU version number.

Profile Parameters

This section lists the profile parameters which are used in conjunction with Unicode and code page support.

Parameter	Description
CP	Defines the default code page for Natural. This code page is used for the runtime and development environment if not superposed with a code page defined for a single object (for example, for a Natural source). Only platform-suitable code pages can be used. This means, for example, that no EBCDIC code page can be defined for a Windows or Linux platform.
CPCVERR	Specifies whether a conversion error that occurs when converting from Unicode to code page or from code page to Unicode or from one code page to another code page results in a Natural error or not. This parameter is not regarded for the conversion of Natural sources when loading them into the source area or when cataloging them.
CPOBJIN	Specifies the code page in which the batch input file for data is encoded. This file is defined with the Natural profile parameter CMOBJIN.
CPPRINT	Specifies the code page in which the batch output file shall be encoded. This file is defined with the Natural profile parameter CMPRINT.
CPSYNIN	Specifies the code page in which the batch input file for commands is encoded. This file is defined with the Natural profile parameter CMSYNIN (Windows and Linux).
SRETAIN	Specifies that all existing sources have to be saved in their original encoding format. See also *Customizing Your Environment*.
SUTF8	Specifies the default format to be used when Natural sources are saved. **Note:** On Linux, this parameter can only be used in a SPoD environment.
SUBCHAR	Specifies the substitution character for the conversion from Unicode to the default code page. If SUBCHAR is OFF, the default substitution character defined by ICU will be used. **Note:** SUBCHAR does not influence conversions from code page to Unicode or from Unicode to code pages which differ from the default code page.
WEBIO	Specifies whether the Natural Web I/O Interface client (which supports Unicode) or the terminal emulation window (which is not Unicode-enabled) is used for input and output. In a local Windows environment, the output window (which is Unicode-enabled) is used.

Parameter	Description
	In a remote Windows environment, the Natural Web I/O Interface client is always used, regardless of the setting of this parameter.

See also:

- *Code Pages for the Input and Output Files* in the section *Natural in Batch Mode* of the *Operations* documentation
- For valid code pages, see ***http://www.iana.org/assignments/character-sets***.

Encoding Information

The encoding of code page data can be specified on different levels.

Level 1 - Default Code Page

The default code page can be defined with the `CP` parameter. It overwrites the system code page and is valid for all code page data.

Level 2 - Code Page for a Single Object

A code page can be defined for Natural sources, batch input (`CPOBJIN`, `CPSYNIN`) and output files (`CPPRINT`).

In addition, a code page can be defined for work files of type ASCII, ASCII compressed, Unformatted and CSV; see *Work File Assignments* in the *Configuration Utility* documentation.

If a code page is defined at object level, this overwrites the default code page.

Important: It is important that the correct code page is defined for every object. For more information, see ***Migrating Existing Applications***.

Deploying Natural Objects with Encoding Information

If you want to deploy Natural objects for which encoding information has already been defined, you have to keep in mind that the encoding information is stored in the file *FILEDIR.SAG* and that it is lost if you deploy only the object file from outside of Natural.

When deploying Natural objects, you have the following possibilities for keeping the encoding information:

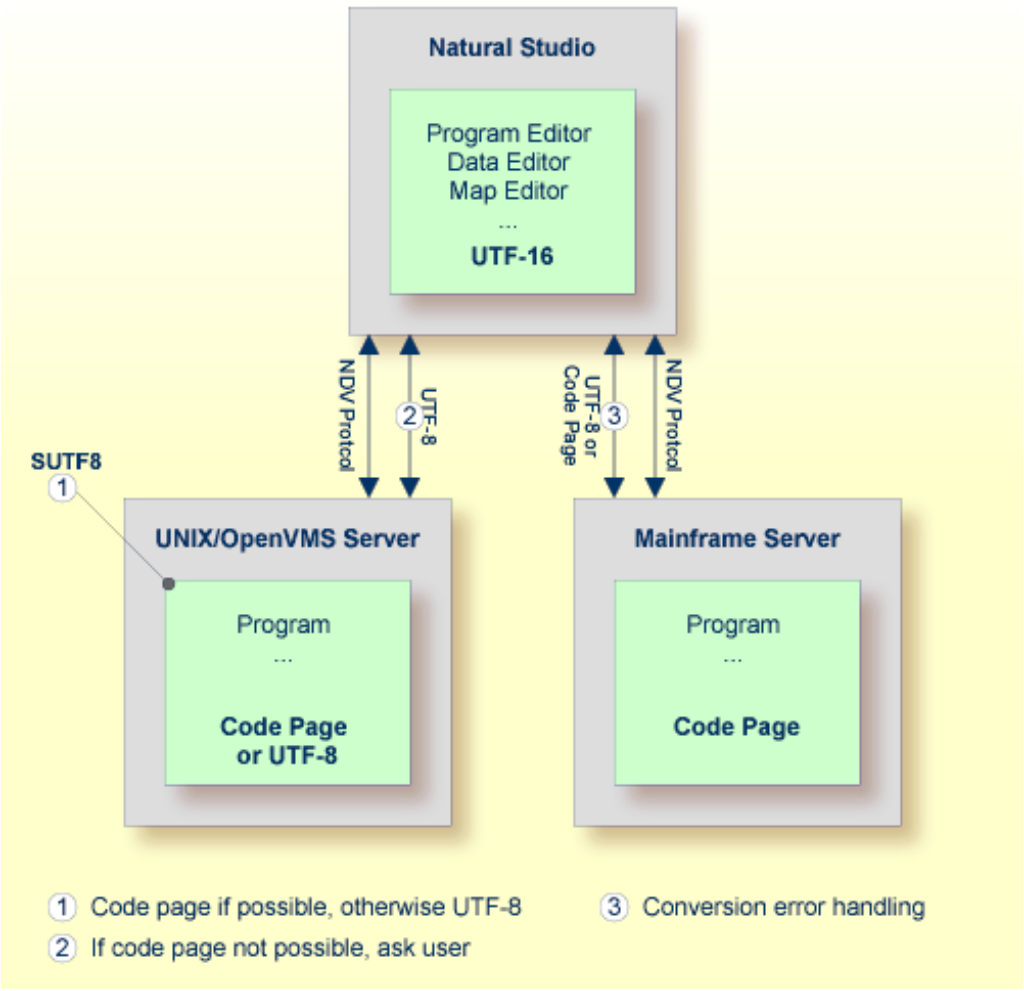
- You can copy the entire library. The copy of the library can then be distributed to all Windows and Linux platforms. In this case, the original code page is kept. If a library is copied from Windows to Linux, you have to keep in mind that it may be possible that the objects cannot be opened with a native Natural for Linux editor because these editors can only open objects with the default code page.
- You can use the Object Handler which keeps the encoding information. In this case, the original code page is kept. If a Windows library is unloaded on Linux, you have to keep in mind that it may be possible that the objects cannot be opened with a native Natural for Linux editor because these editors can only open objects with the default code page.
- You can copy and paste objects with Natural Studio. In a SPoD environment, if the target environment is located on a platform different from the source environment, Natural tries to save the object with the default code page of the target environment. If this is not possible, the object is stored in UTF-8 format. For Linux targets, this assures that the object can be opened with the native Natural for Linux editor, if all characters of the source are available in the default code page of the Linux server.

5 Development Environment

- Development Environment for Applications 16
- Customizing Your Environment 17
- Editors in the SPoD Environment 18

Development Environment for Applications

The development environment for Unicode applications is Natural Single Point of Development (SPoD).



In a SPoD environment, the Natural objects of a Unicode application which are located on a Natural Development Server (NDV) can be modified using Natural Studio. If supported by the server, the sources are exchanged between client and server in UTF-8 format.

On NDV servers for Linux, the setting of the profile parameter `SUTF8` determines the format that is used when storing the Natural object on the server. This is handled just like the local Windows case.

On NDV servers for mainframes, the objects are stored with the default or their original encoding, depending on the setting of the profile parameter `SRETAIN`.

Customizing Your Environment

It is important that you define the correct default code page for your environment before changing any Natural code. For more information, see *Migrating Existing Applications*.

If you want to store characters from different languages in your sources, you have to save the sources in UTF-8 format, or you have to use hexadecimal UH constants in the sources. With the profile parameters `SUTF8` and `SRETAIN` you can control in which format sources are saved. The following table lists some situations and the recommended settings.

Note: On Linux, the parameter `SUTF8` can only be used in a SPoD environment.

Situation	Settings	Effect
Sources are located on Windows; U constants are needed.	`SUTF8=ON, SRETAIN=OFF`	All sources are saved in UTF-8 format when saving them with Natural 6.2 or above. New sources are created in UTF-8 format. All characters can be stored in a source.
Sources are located on Windows and Linux; U constants are needed and SPoD is used for development.	`SUTF8=ON, SRETAIN=ON`	All sources are saved in UTF-8 format when a conversion to the original code page is no longer possible; if it is possible, the code page of a source will not be changed. New sources are created in UTF-8 format. All characters can be stored in a source. A source with UTF-8 format can only be changed with SPoD; it can no longer be handled with the Natural for Linux editor.
Sources are located on Windows and Linux; no U constants are needed.	`SUTF8=OFF, SRETAIN=ON`	All sources are saved with the original code page. New sources are saved with the default code page (of server). Only characters from the source code page can be stored in a source. The sources can further be handled with the Natural for Linux editor.
Sources are located on Windows and Linux; U constants are needed and SPoD is used for development.	`SUTF8=OFF, SRETAIN=ON`	All sources are saved with the original code page. New sources are saved with the default code page (of the server). Only characters from the source code page can be stored in a source. The sources can further be handled with the editors for Natural for Linux and Natural for Mainframes. All Unicode constants have to be defined as hexadecimal constants (UH).

If the parameter `SUTF8` is set to `OFF` and you stow a source which contains characters from different character sets, but which was not yet saved in UTF-8 format, it is possible that the generated program is created, but that the source cannot be saved and thus remains unchanged. This happens if characters from different character sets are used in a comment or in a U constant. For this reason, it is recommended that you set the parameter `SUTF8` to `ON` if you want to create sources with characters from different character sets and if your sources do not need to be distributed to mainframe platforms.

If the parameter SRETAIN is set to OFF, all sources are saved with the default code page. You have to be careful with this setting because it may lead to improper code page information if you have sources which were created with an earlier Natural version. In this case, the encoding information of the source is unassigned and the source is always opened with the default code page (value of the system variable *CODEPAGE). This will often work even if the default code page is not the correct encoding of the source. Some language-specific characters will be displayed incorrectly in this case. If such a source is opened with the wrong code page and is saved with SRETAIN being set to ON, no encoding will be stored for the source; the source can later be opened correctly if Natural is started with the correct default code page. However, once you have saved the source with SRETAIN being set to OFF, the default code page will be saved as the encoding of the source; from this time on, the source will only be opened with this code page. For this reason, you should use this setting only if you are certain that all of your Natural sources are encoded in the default code page.

See also: *Regional Settings* in the *Configuration Utility* documentation.

Editors in the SPoD Environment

The Natural for Windows editors are fully Unicode-enabled. Via SPoD they can also be used for mainframe and Linux sources. The editors provided with Natural for Mainframes and Natural for Linux are not Unicode-enabled.

When a source is opened with an editor in Natural Studio (Natural for Windows), the content of the source will be converted from the corresponding code page to Unicode before it is loaded into the editor. This will guarantee that all characters can be displayed correctly even if the source contains characters which are not included in the system code page. If the conversion from the source's code page to Unicode fails, an error will be displayed and the editor is not opened. In this case, the user has to define the correct encoding of the source. The source encoding can be changed in the **Properties** dialog box (see *Properties for the Nodes* in the *Using Natural Studio* documentation).

For Windows and Linux sources, the Natural for Windows editors allow saving sources which contain characters from different languages in UTF-8 format. On mainframes, it will not be possible to save UTF-8 sources.

Note: If you save a Linux source in UTF-8 format or with a code page which differs from the default code page, the source can no longer be opened with the native Natural for Linux editor. Mainframe sources can be saved with a different code page and can be edited with the native Natural for Mainframes editors.

Even if you do not want to use Unicode strings in your programs and sources, the Unicode-enabled editors have the advantage that you can write sources in all code pages, no matter which system code page is installed. For example, if you have installed the "windows-1252" (Latin 1) code page, you can write a program containing Cyrillic characters and save this program with the "windows-

1251" (Cyrillic) code page. You only have to select code page "windows-1251" in the **Save As** dialog box (see *Saving an Object with a New Name* in the *Using Natural Studio* documentation).

Using the Natural for Windows program editor, you can convert text constants into their hexadecimal Unicode representations (see *Converting to Hexadecimal Format* in the *Program Editor* section of the Natural for Windows *Editors* documentation). If you are developing for a platform where UTF-8 sources are not preferred, you can thus enter all characters for a Unicode constant, select all the characters of the constant, convert them to their hexadecimal representation and then add the "UH" prefix for Unicode hexadecimal constants. Furthermore, when you hover the mouse pointer over a character or a selected character range of a text constant, a tool tip shows the corresponding hexadecimal Unicode representation.

A byte order mark (BOM) consists of the character code "U+FEFF" at the beginning of a data stream where it can be used as a signature defining the byte order and encoding form, primarily of unmarked plain-text files. On Windows, a byte order mark is used by some editors (for example, Notepad) to mark UTF-8 files. The Natural for Windows editors will recognize an UTF-8 byte order mark when reading an object. If the object has no other encoding defined so far, Natural will interpret it as UTF-8 and when the object is saved, UTF-8 will be stored as the encoding for the object. The byte order mark is removed in this case.

6 Unicode and Code Page Support in the Natural Programming Language

- Natural Data Format U for Unicode-Based Data 22
- Statements 23
- Logical Condition Criteria 27
- System Variables 28
- Large and Dynamic Variables 28
- Session Parameters 28
- Sample Programs 31

Natural Data Format U for Unicode-Based Data

In Natural, you can specify Unicode strings with the format U and U constants.

- **Format U**
  With format U, you can define data which holds Unicode strings. The Natural data format U is internally UTF-16.

  See also *Format and Length of User-Defined Variables* in the *Programming Guide*.

- **U Constants**
  You can define Unicode constants with the prefix "U". For example:

  ```
  U'Äpfel'
  ```

  The prefix "UH" can be used for defining Unicode constants in hexadecimal format. Four hexadecimal digits represent one UTF-16 code unit as defined by the Unicode Standard. So the overall length must be a multiple of four. For example, if you need the hexadecimal form of

  ```
  U'Äpfel'
  ```

  you need the UTF-16 code units for "Ä", "p", "f", "e" and "l" (which are "U+00C4", "U+0070", "U+0066", "U+0065" and "U+006C") and you have to combine them to the following hexadecimal string:

  ```
  UH'00C40070006600650006C'
  ```

  See also *Unicode Constants* in the *Programming Guide*.

The data format U is endian-dependent. This has to be considered when moving between the formats B and U.

U versus A

The advantage of the U format (as compared with the A format) is, that it can hold any combinations of characters from different languages and that it does not depend on the default code page (value of the system variable `*CODEPAGE`). Moreover, the U format makes it easier to share data between different platforms; no more conversions (for example, from EBCDIC to ASCII) are necessary.

On the other hand, U format data often consumes more memory than A format data. For languages in which most strings can be represented by single-byte encoding, U format will result in strings occupying twice the space that was previously required. However, for East Asian languages, the memory consumption will often not be higher.

Statements

Basically, U format can be used in most statements which allow A format. However, if a Natural object name is given as an operand of a statement (for example, in the `CALLNAT` statement), U cannot be used because Natural object names have A format. For information on a specific statement, see the *Statements* documentation.

Basically, A and U format can be used together in one statement, however, it is recommended that you use only one format within one statement, either A or U. If both formats are used together, all variables have to be converted to a uniform format; this may lead to conversion errors.

The following statements are particularly affected when using Unicode:

- MOVE NORMALIZED
- MOVE ENCODED
- EXAMINE
- PARSE JSON
- PARSE XML
- REQUEST DOCUMENT
- CALLNAT (RPC)

MOVE NORMALIZED

Normalization in Unicode: A process of removing alternate representations of equivalent sequences from textual data in order to convert the data into a form that can be binary-compared for equivalence. The Unicode Standard defines different normalization forms. The normalization form that results from the canonical decomposition of a Unicode string, followed by the replacement of all decomposed sequences by primary composites where possible, is called “Normalization Form Composed” (NFC).

Natural assumes that all Unicode data is in NFC format to assure that string operations can be performed without partial truncation of a Unicode character. Natural conversion operations assure that the resulting Unicode string is in NFC. If Unicode data is received from outside of Natural and it is not guaranteed that the data has NFC format, the `MOVE NORMALIZED` statement can be applied.

Example:

Character Sequence	NFC
ê (U+00EA)	ê (U+00EA)
e (U+0065) + ^ (U+0302)	ê (U+00EA)

Note: Concatenating two or more strings in NFC format can result in not-NFC format.

MOVE ENCODED

An implicit conversion between Unicode and the default code page (value of the system variable `*CODEPAGE`) is performed when moving strings from U to A or vice versa with the `MOVE` statement.

Furthermore, the `MOVE ENCODED` statement can be used for conversion between different code pages or from any available code page to Unicode and vice versa. This can be helpful if data is coming from outside of Natural and this data is coded in a code page which differs from the default code page. But even for conversions between the default code page and Unicode, this statement can be used if you want to obtain a potential conversion error with the `GIVING` clause; if `CPCVERR` is set to `ON`, the `MOVE` statement will stop with a runtime error in this case.

If a character cannot be converted, it depends on the setting of the `CPCVERR` parameter whether a substitution character is used for this character or whether the conversion fails. The default substitution character (defined by ICU) for the conversion from Unicode to the default code page (`CP`) can be changed with the profile parameter `SUBCHAR`.

This statement can also be used for conversion from U data into UTF-8 format.

Note: If you convert data to a code page which differs from the default code page, it is recommended not to use this data in I/O. I/O is only meaningful with the default code page.

EXAMINE

A “grapheme” is what a user normally thinks of as a character. In most cases, a Unicode code point is a grapheme, however, a grapheme can also consist of several Unicode code points. For example, a sequence of one base character and one or more combining characters is a grapheme.

Example: "a" (U+0061) + "." (U+0323) + "^" (U+0302) defines one grapheme which is displayed as follows:

ậ

Note: If a base/combining character sequence is normalized, this does not mean that the sequence is always replaced by a pre-composed character, because not all characters are available in a pre-composed format.

A “supplementary code point” is a Unicode code point between "U+10000" and "U+10FFFF". A supplementary code point is in UTF-16, represented by a surrogate pair which consists of two

code units where the first value of the pair is a “high-surrogate code unit”, and the second is a “low-surrogate code unit”. Such characters are generally rare, but some are used, for example, as part of Chinese and Japanese personal names, and therefore support for these characters is commonly required for government applications in East Asian countries.

The string handling statements such as EXAMINE and its SUBSTRING option work on UTF-16 code units. It is the user's responsibility that the code does not separate graphemes or surrogate pairs.

However, the clauses CHARPOSITION and CHARLENGTH of the EXAMINE statement (see *Syntax 3 - EXAMINE for Unicode Graphemes*) can be used to ask for the start and length (in UTF-16 code units) of graphemes. The result values can be used for SUBSTRING calls. With these clauses, it is possible to scan a string grapheme by grapheme.

Example:

```
DEFINE DATA LOCAL
1 #UNICODE-STRING    (U15)
1 #CODE-UNIT-INDEX   (N4)
1 #CODE-UNIT-LEN     (N4)
1 #GRAPHEME-NUMBER   (N4)
END-DEFINE

MOVE U'aปcอี่bนัcยี่d' TO #UNICODE-STRING

#GRAPHEME-NUMBER := 1

REPEAT
EXAMINE
        FULL VALUE OF #UNICODE-STRING
        FOR CHARPOSITION #GRAPHEME-NUMBER
        GIVING POSITION IN #CODE-UNIT-INDEX
        GIVING LENGTH IN #CODE-UNIT-LEN

        DISPLAY #UNICODE-STRING #GRAPHEME-NUMBER #CODE-UNIT-INDEX #CODE-UNIT-LEN

        #GRAPHEME-NUMBER := #GRAPHEME-NUMBER + 1
WHILE #CODE-UNIT-INDEX NE 0
END-REPEAT

END
```

The above example program provides the following output:

```
Page     1                                                    05-12-15  09:33:49

#UNICODE-STRING #GRAPHEME-NUMBER #CODE-UNIT-INDEX #CODE-UNIT-LEN
--------------- ---------------- ---------------- --------------

aปิcอิ่bนัcยื่d           1                1                1
aปิcอิ่bนัcยื่d           2                2                2
aปิcอิ่bนัcยื่d           3                4                1
aปิcอิ่bนัcยื่d           4                5                3
aปิcอิ่bนัcยื่d           5                8                1
aปิcอิ่bนัcยื่d           6                9                3
aปิcอิ่bนัcยื่d           7               12                1
aปิcอิ่bนัcยื่d           8               13                3
aปิcอิ่bนัcยื่d           9                0                0
```

PARSE JSON

The document to be parsed is always internally converted to UTF-8 if the document is not already encoded in UTF-8.

See the description of the `PARSE JSON` statement for further information.

See also *Statements for Internet Access and Parsing* in the *Programming Guide*.

PARSE XML

XML documents can contain information within the XML document header about the encoding of the document (for example, `<?xml version="1.0" encoding="UTF-8" ?>`). If an XML document contains this information, the parsing of the XML document on Windows and Linux platforms always includes a conversion of the code page given within the XML document header to the default code page of Natural (value of the system variable `*CODEPAGE`), if the receiving field is not of format U.

See the description of the `PARSE XML` statement for further information.

See also *Statements for Internet Access and Parsing* in the *Programming Guide*.

REQUEST DOCUMENT

Data transfer with the `REQUEST DOCUMENT` statement normally does not involve any code page conversion. If you want to have the outgoing and/or incoming data encoded in a specific code page, you can use the `DATA ALL` clause and/or the `RETURN PAGE` clause of the `REQUEST DOCUMENT` statement to specify this.

See the description of the `REQUEST DOCUMENT` statement for further information.

See also *Statements for Internet Access and Parsing* in the *Programming Guide*.

CALLNAT (RPC)

Data exchange in Unicode format via RPC is supported. See the description of the `CALLNAT` statement.

If U data is sent from a platform with big endian encoding to a platform with little endian encoding or vice versa, the encoding is adapted so that it conforms with the encoding on the receiving platform. For example, when U data in little endian encoding arrives on a big endian platform, this data is converted to big endian encoding before it is handed over to the program. When this data is sent back, it is converted back to little endian encoding.

Logical Condition Criteria

In a logical condition criterion, Unicode operands can be used together with alphanumeric and binary operands. If not all operands are Unicode operands (format U), the second and all following operands are converted to the format of the first operand. If a binary operand (format B) is specified as the second or a following operand, the length of the binary operand must be even; the binary operand is assumed to contain Unicode code points.

If the first operand is a Unicode operand (format U) and the comparison is therefore performed as a Unicode comparison, the ICU collation algorithm is used. The ICU algorithm does not perform a plain binary comparison. For example,

- some code points such as "U+0000" are ignored during the comparison process,
- combined characters are considered as being equal to the equivalent single code point (for example, the German character "ä" represented by "U+00E4" and the combination of the code points "U+0061" and "U+0308" are considered as being equal by ICU).

Note: Comparing an alphanumeric and a Unicode operand can deliver different results, depending on the sequence of the fields.

See also *Logical Condition Criteria* in the *Programming Guide*.

System Variables

*CODEPAGE

The system variable `*CODEPAGE` is used to return the IANA name of the default code page, that is, the code page used for conversions between Unicode and code page format.

*LOCALE

The system variable `*LOCALE` contains the language and country of the current locale.

Large and Dynamic Variables

U format can be used for large and dynamic variables. For dynamic U variables, `*LENGTH` returns the number of UTF-16 code units.

See also *Introduction to Dynamic Variables and Fields* in the *Programming Guide*.

Session Parameters

The following session parameters are available:

Parameter	Description
`DL`	Specifies the display length for a field of format A or U. See also *Display Length for Output - DL Parameter* in the *Programming Guide*.
`EMU`	Edit mask in Unicode.
`ICU`	Insertion character in Unicode.
`LCU`	Leading characters in Unicode.
`TCU`	Trailing characters in Unicode.

DL versus AL

As long as Natural was not Unicode-enabled, the length of an alphanumeric field was always identical to the number of columns needed for displaying the field (called number of display columns). This was even true for the East Asian languages which use DBCS code pages: an A format field can hold only half the characters (for example, A10 results in A5).

Example:

```
DEFINE DATA LOCAL
1  #A8 (A8)
END-DEFINE
#A8 := 'computer'
WRITE #A8
#A8 := '電腦系統'
WRITE #A8
END
```

The above code results in the following output:

```
Page      1 ...

computer
電腦系統
```

With U format fields, the length of a field and the number of display columns is no longer identical. U characters can have narrow width (for example, Latin characters), wide width (for example, Chinese characters) or no width (for example, combining characters). Therefore, it is totally unknown how many display columns a U field needs; this depends on the contents of the field. Natural cannot automatically decide how many columns are to be reserved on the screen: if the maximum size is assumed, Latin output will have large gaps, and if the minimum size is assumed, Chinese output cannot be displayed totally. Therefore, the Natural programmer has to define the display width of a field; this is done with the `DL` parameter. The `AL` parameter cannot be used for this purpose, because it cuts away the part of the field which exceeds the defined length. But we do not want to cut any characters from the U field; we only want to define the start position of the following field.

Example:

```
DEFINE DATA LOCAL
1  #U8 (U8)
1  #U4 (U4)
END-DEFINE
#U8 := 'computer'
WRITE #U8
#U4 := U'電腦系統'
WRITE #U4 (DL=8)
END
```

The above code results in the same output as above:

```
Page      1 ...

computer
電腦系統
```

On Windows, either locally with the output window or in a remote development environment with the Natural Web I/O Interface client, it is possible to scroll in a field where the defined value for the DL parameter is smaller than the real display width of the field.

EMU, ICU, LCU, TCU versus EM, IC, LC, TC

The parameters EMU, ICU, LCU and TCU allow using characters which are not included in the default code page. They are stored in Unicode format in the generated program. These parameters can be used with all field formats.

The parameters EM, IC, LC and TC can also be used with U format fields. These parameters may also be useful if characters which are contained in the default code page have different encodings in other code pages. For example, the Euro sign (€) has the code point "0x80" in the "windows-1252" (Latin 1) code page, but the code point "0x88" in the "windows-1251" (Cyrillic) code page. Thus, using a Unicode parameter (EMU, ICU, LCU or TCU) will assure that the Euro sign is always displayed correctly, no matter what code page is installed on the PC.

Example for EMU:

```
DEFINE DATA
LOCAL
 01 EMPLOYEES-VIEW VIEW OF EMPLOYEES
   02 FIRST-NAME
   02 NAME
   02 SALARY (1)
END-DEFINE
*
  READ (6) EMPLOYEES-VIEW
    DISPLAY NAME FIRST-NAME SALARY(1) (EMU=999,999€)
  END-READ
*
END
```

The above code results in the following output:

```
Page     1                                                  05-12-15  11:45:36

        NAME              FIRST-NAME        ANNUAL
                                            SALARY
-------------------- -------------------- --------

ADAM                 SIMONE               159,980€
MORENO               HUMBERTO             165,810€
BLOND                ALEXANDRE            172,000€
MAIZIERE             ELISABETH            166,900€
CAOUDAL              ALBERT               167,350€
VERDIE               BERNARD              170,100€
```

Sample Programs

The library SYSEXPG contains sample programs for Unicode and code page support in Natural:

- UNICOX01 lists all Unicode characters.
- UNICOX02 converts Unicode characters to code points and vice versa.
- CODEPX01 lists all code pages, whether the code page is supported in Natural and which encoding it uses. For all supported code pages, it offers services to list the characters of the code page and to convert a string from the code page into its hexadecimal representation and vice versa.
- CODEPXL1 lists all characters of any 1-byte code page.
- CODEPXL2 lists all characters of any 2-byte code page.
- CODEPXC1 converts a string from any code page into its hexadecimal representation and vice versa.

7 Unicode Input and Output Handling in Natural Applications

- Displaying and Entering Unicode Data 34
- Natural Web I/O Interface Client 35

Displaying and Entering Unicode Data

If you want to display or enter Unicode data, the following possibilities exist:

- When working in the local development environment with Natural for Windows, all Unicode characters can be displayed and entered in the Natural output window.
- When working in a remote development environment with Natural for Windows (SPoD), the Natural Web I/O Interface client (see **below**) is necessary for displaying and entering all Unicode characters.
- When running applications with Natural for Linux, Natural for Mainframes or Natural for Windows, see *Natural Web I/O Interface Client* below.

Notes:

1. Even if you are working with a Unicode-enabled output interface on Windows, you will see only the Unicode characters which are supported by the currently selected font.
2. If you are working with the Unicode-enabled output window on Windows, characters which are not contained in the current code page will be ignored when entering data in A format fields if the parameter `CPCVERR` is `ON`.
3. Unicode data cannot be displayed on 3270 terminals.

If you run Natural via a terminal emulation or a mainframe terminal such as IBM 3270/3279, the page will be converted to the default code page (value of the system variable `*CODEPAGE`) before displaying it, so that all characters which are not contained in the default code page are replaced with the substitution character. Equally, input is only possible in code page format and will be converted to Unicode format before assigning it to a U format field. You have to regard that the substitution character is defined by the ICU conversion tables. Depending on this character, it is possible that garbage is displayed with a terminal emulation. On Linux platforms, you can change this substitution character by setting the profile parameter `SUBCHAR`. However, it is strongly recommended that you use the Natural Web I/O Interface when displaying characters not contained in the default code page. When running a remote Windows session, the Natural Web I/O Interface will be used in any case.

On code page oriented mainframe terminals, it is important to select the suitable code page. The default code page of Natural, the code page of the terminal and even the font used by the terminal determine the capability of displaying certain characters correctly.

Natural Web I/O Interface Client

The Natural Web I/O Interface client is used to display non-GUI information which contains Unicode characters. It can be used in the following environments:

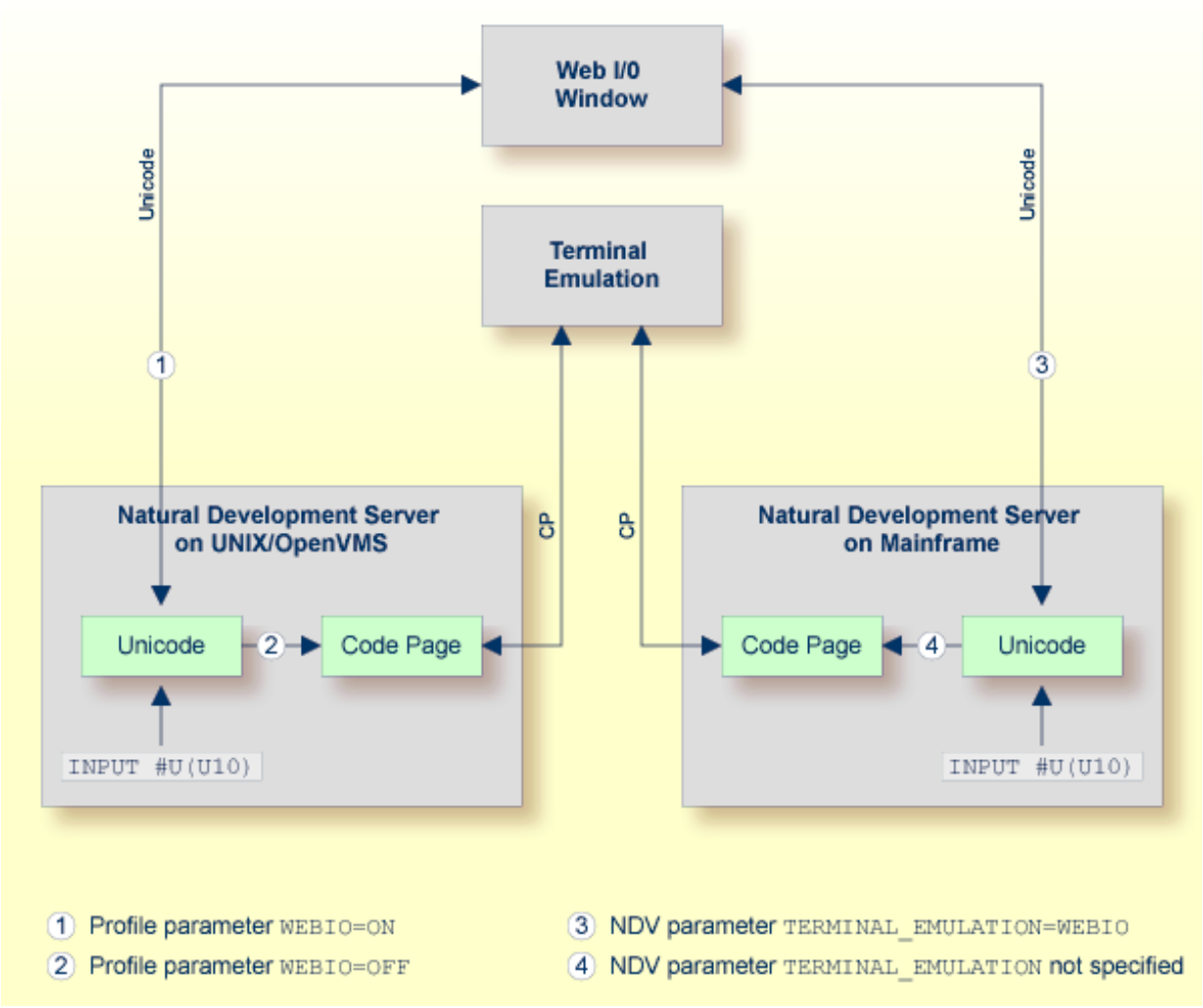
- SPoD Environment
- Runtime Environment

SPoD Environment

The Natural Web I/O Interface client can be invoked when you use Natural for Windows and you are working with Natural Studio in a remote development environment (SPoD); see *Natural Web I/O Interface Client* in *Remote Development Using SPoD* which is part of the Natural for Windows documentation.

When the Natural Web I/O Interface client is used, the Web I/O window appears instead of the terminal emulation window which is not Unicode-enabled in remote Linux or mainframe environments, or instead of the output window in remote Windows environments.

The following graphic shows the SPoD environment for Unicode applications with Natural Development Servers (NDV) on Linux and mainframes:



So that the Natural Web I/O Interface client can be invoked, the Natural Development Server has to be configured as follows:

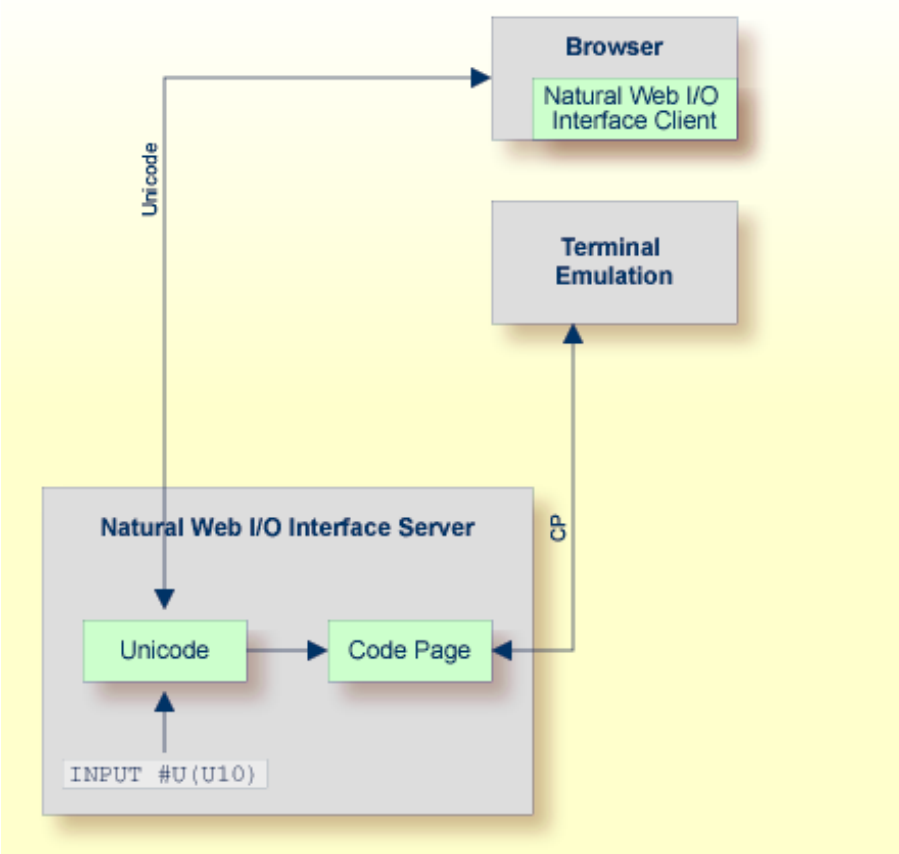
- **Linux**
  If you want to use the Natural Web I/O Interface client in a remote Linux environment, the profile parameter `WEBIO` must be set to `ON` on the NDV server. See *Configuration Utility* in the *Natural for Linux and Cloud* documentation.
- **Mainframe**
  If you want to use the Natural Web I/O Interface client in a remote mainframe environment, the NDV configuration parameter `TERMINAL_EMULATION` must be set to `WEBIO` on the NDV server. See *NDV Configuration Parameters* in the *Natural Development Server* documentation. The Natural profile parameter `TMODEL` can be used to determine the user screen size.

- **Windows**
  In a remote Windows environment, the Natural Web I/O Interface client is always used, regardless of the setting of the profile parameter `WEBIO`.

Runtime Environment

The Natural Web I/O Interface client appears when running applications with Natural. It runs in a web/application server.

The following graphic shows the runtime environment for Unicode applications:



Natural recognizes automatically whether the session has been started from the Natural Web I/O Interface client or from the terminal emulation.

Prerequisites for using the Natural Web I/O Interface client:

- **Natural for Linux**
  It is required that the Natural Web I/O Interface server (which is implemented as a daemon) has been installed and activated. See *Natural Web I/O Interface* in the *Natural for Linux and Cloud* documentation.

- **Natural for Windows**
  It is required that the Natural Web I/O Interface server (which is implemented as a service) has been installed and activated. See *Natural Web I/O Interface* in the *Natural for Windows* documentation.

8 Bidirectional Language Support

- General Information 40
- Screen Direction 40
- Field Direction 41
- Maps and Dialogs 43
- Print Methods 43
- Terminal Capabilities 44

General Information

Some languages, for example Arabic and Hebrew, are written from right-to-left (RTL), whereas the majority of the languages, for example English and German, are written from left-to-right (LTR). Text which contains both left-to-right and right-to-left characters is called bidirectional text.

Natural provides a basic support for bidirectional languages. On Windows, this support is activated when both the Natural default code page and the Windows system code page are defined as bidirectional code pages. If Natural does not define a specific code page, it is sufficient when a bidirectional Windows system code page has been defined. On Linux, the support for bidirectional languages is activated when the Natural default code page is a bidirectional code page.

The output of Natural programs can be controlled using the profile parameter `PM`, the terminal command `%V`, and the session parameter `PM`.

On Linux, the profile parameter `DO` (Display Order) is additionally used to support applications that have been originally written for terminals which support inverse (right-to-left) print mode, but no bidirectional data. These applications create the display order of bidirectional data in the application code. With the parameter `DO`, these applications are enabled to run compatibly also with I/O devices that support bidirectional data. This is for instance the case if an application runs in a browser with the Natural Web I/O Interface.

Screen Direction

The profile parameter `PM` defines the default screen direction. When `PM` is set to `R` (reset), the default screen direction is left-to-right. When `PM` is set to `I` (inverse), the default screen direction is right-to-left. All non-alphanumeric fields and system variables are automatically inverted by Natural so that they are displayed correctly from right-to-left if the screen direction is right-to-left. PF key lines (Linux) are not inverted; they are always shown from left-to-right.

The terminal command `%V` can be used to change the screen direction. If the screen direction is right-to-left, the layout of the current window is mirrored, which means that the origin of all window components or fields is the upper right corner. The screen direction is changed to right-to-left using `%VON` and is reverted to left-to-right using `%VOFF`.

Field Direction

The session parameter PM reverses the direction of a field. The effect of "reversing the direction of a field" depends on the statement in which the PM parameter is used and the platform. If the PM parameter is used in a MOVE statement, the content of the field is simply reversed (that is, the first character will become the last character, and so on); the result does not depend on the characters of the field. Trailing blanks are removed before the field is reversed.

For example, the following program

```
DEFINE DATA LOCAL
1  TEST1  (A10)
1  TEST2  (A10)
END-DEFINE
TEST1 := 'program'

MOVE TEST1 (PM=I) TO TEST2
INPUT TEST1 (AD=O) TEST2 (AD=O)

END
```

produces the following output:

```
TEST1 program    TEST2 margorp
```

where "margorp" is the reversed version of "program".

When the PM parameter is used for IO statements such as INPUT or DISPLAY, its effect is even more complex. In this case, the field direction is based on the screen direction:

- If the screen direction is left-to-right and PM=I is applied to a field, the field direction changes to right-to-left.
- If the screen direction is right-to-left and PM=I is applied to a field, the field direction changes to left-to-right.

On Windows and browser terminals (Natural Web I/O Interface), "reversing the field direction" does not mean that the characters of the field are simply reversed. Instead, the complex bidirectional algorithm is applied (for more information, see the Microsoft Windows documentation). On character-oriented terminals, however, the characters of a field are not resorted; they are simply reversed.

In the following example, the characters assigned to the variable TEST have been entered in the following sequence:

```
a b c  שנב  1 2 3
```

The following is an example program for Windows. The characters of the constant are already resorted when entering them in the program editor.

```
DEFINE DATA LOCAL
1  TEST  (A20)
END-DEFINE
TEST := 'abc 123 בנש'

SET CONTROL 'voff'

INPUT TEST (AD=O)  /
      TEST (AD=O PM=I)

SET CONTROL 'von'

INPUT TEST (AD=O)  /
      TEST (AD=O PM=I)
END
```

This program produces the following two screens on Windows:

```
TEST abc 123 בנש
TEST          123 בנש  abc
```

and

```
                                        123 בנש  abc TEST
                                  abc 123 בנש           TEST
```

The following is an example program for Linux. If the characters are entered in the sequence as described above, the program is displayed in the following way, because the characters are simply displayed in the keying sequence.

```
DEFINE DATA LOCAL
1  TEST  (A20)
END-DEFINE
TEST := 'abc שנב 123'

SET CONTROL 'voff'

INPUT TEST (AD=O)  /
      TEST (AD=O PM=I)

SET CONTROL 'von'

INPUT TEST (AD=O)  /
      TEST (AD=O PM=I)
END
```

On Linux, this program produces the following two screens:

```
TEST abc אבג 123
TEST          321 גבא  cba
```

and

```
                                        321 גבא  cba TSET
                                  abc אבג 123           TSET
```

Maps and Dialogs

On Windows and Linux, the map editor simplifies the handling of maps with bidirectional fields by offering the **Reverse Map** command. This command changes the display direction of the current map. The position of the fields is not changed; only the view is changed. On Windows, this command applies only to the current map. On Linux, a flag is set so that all following maps are displayed reversed; a following **Reverse Map** command will restore the original situation.

On Windows, the output of dialogs can be controlled in a similar way: both the dialog itself and most of the dialog controls offer an `RTL` attribute. If the `RTL` attribute of the dialog is checked, the screen direction of the dialog is right-to-left. If the `RTL` attribute of other controls is checked, the direction of these controls is right-to-left.

The profile parameter `PM` defines the default setting of the `RTL` attribute for new dialogs. When `PM` is set to `R` (reset), the `RTL` attribute is unchecked by default. When `PM` is set to `I` (inverse), the `RTL` attribute is checked by default. The default setting of the `RTL` attribute for newly created controls of a dialog is derived from the `RTL` attribute setting of the dialog.

If the `RTL` attribute of a dialog is changed when the dialog already contains controls, a dialog appears asking whether the `RTL` attributes of the controls should also be changed.

Print Methods

When working with bidirectional languages on Windows, "GUI" is the preferred print method. With the print method "GUI", the printed page will show the same layout as the window displayed on the screen. The sorting of the field characters is identical.

If the print method "TTY" is used, the printed layout will most probably differ from the layout of the screen window because the field characters are printed in logical sequence. For fields with right-to-left direction, all characters are simply reversed (that is, the first character will become the last character, and so on).

Terminal Capabilities

On Linux, some special terminal capabilities for bidirectional support can be defined with the Natural Termcap utility.

The key which is defined by the RTLF capability can be used to toggle the input direction of a field at runtime.

With the RTLM and LTRM capabilities, it is possible to switch automatically between right-to-left and left-to-right input mode - provided that the terminal emulation supports this functionality. The RTLM escape sequence will be inserted in front of right-to-left fields, and the LTRM escape sequence will be inserted in front of left-to-right fields.

9 Double-Byte Character Support

In most East Asian languages, language-specific characters in code page strings (that is, Natural format A) are represented by 2 bytes (the so-called double-byte characters) and ASCII characters are represented by 1 byte. Thus, a code pages string consists of characters with different lengths: some have 1 byte and others have 2 bytes.

Natural provides a basic support for double-byte characters. On Windows, this support is activated when both the Natural default code page and the Windows system code page are defined as double-byte code pages. If Natural does not define a specific code page, it is sufficient when a double-byte Windows system code page has been defined. On Linux, the support for double-byte characters is activated when the Natural default code page is a double-byte code page.

When double-byte character support is enabled, Natural assures for all string manipulations that a double-byte character is treated as a unit. This is essential for keeping the meaning of a string.

If a single leading or trailing byte of a double-byte character is left over after the manipulation of a variable of format A (for example, after extracting a substring with the `SUBSTRING` option), this byte is replaced with a blank character.

For the example below, the code page Shift_JIS is selected. Variable `#A` contains a string which consists of four characters. The first and last character is the double-byte character "FULL WIDTH LATIN SMALL LETTER B" which is represented in code page Shift_JIS by the byte sequence `H'8282'`. The second and third character is the single byte character "LATIN SMALL LETTER A" which is represented by one byte `H'61'`. Thus, the hexadecimal representation of the full string is `H'828261618282'`.

```
DEFINE DATA LOCAL
 1   #A   (A10)
END-DEFINE

#A := 'ｂaaｂ'

WRITE #A #A (EM=H(6))
EXAMINE #A FOR PATTERN 'Ｂ' REPLACE 'a'
WRITE #A #A (EM=H(6))

END
```

Without double-byte character support the output of the above program is as follows:

```
Page         1                                07-02-07    17:22:09

ｂaaｂ     828261618282
Ｂaｂ      826161828220
```

This is the result of not having treated the character "ｂ" (`H'8282'` in code page Shift_JIS) as one unit. The trailing byte of this character and the following character "a" (`H'61'`) are falsely interpreted as the double-byte character "Ｂ" (`H'8261'` in code page Shift_JIS).

With double-byte character support, the output of the program is as expected:

```
Page         1                                07-02-07    17:22:09

ｂaaｂ     828261618282
ｂaaｂ     828261618282
```

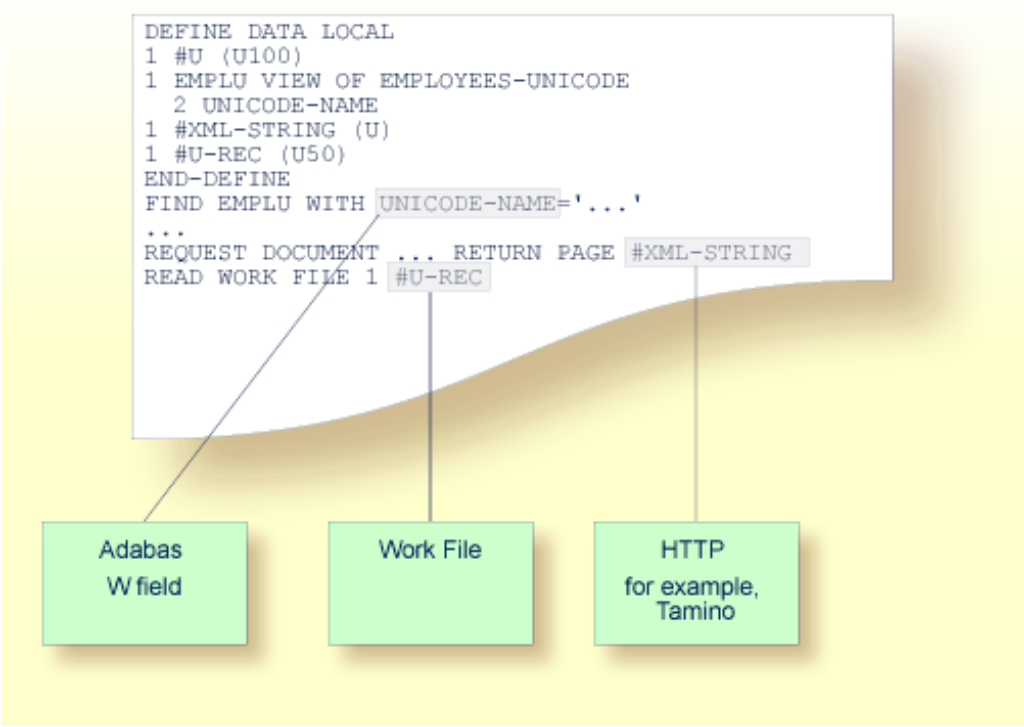
Note: On Windows, the Natural output window has been Unicode-enabled which means that all fields have Unicode format now. In case of A format fields containing double-byte characters, the behavior of the Natural output window has changed slightly. For A format input fields it is now possible to enter “Unicode-string-length” characters in the field. When leaving the field and the default code page is a double-byte code page, all characters which do not fit into the target A format field are removed. For example, an A10 field can hold 5 double-byte characters. In the output window, this field is represented by a Unicode field of length 10 with display length 5. So the user can enter 10 double-byte characters in the input field. When the user moves the cursor to another field on the page or leaves the page by pressing ENTER, the content of the field is converted to code page format so that only the first 5 double-byte characters remain.

10 Unicode Data Storage

- Unicode Data and Parameter Access 48
- Database Management System Interfaces 48
- Work Files and Print Files 49

Unicode Data and Parameter Access

The following graphic shows how Unicode data and parameters are accessed.



Database Management System Interfaces

Accessing Unicode Data in an Adabas Database

Natural enables users to access wide-character fields (format W) in an Adabas database.

Data Definition Module

Adabas wide-character fields (W) are mapped to the Natural data format U (Unicode).

Access Configuration

Natural receives data from Adabas and sends data to Adabas using UTF-16 as common encoding.

Before accesing unicode data in an Adabas database, you must set the correct `OPRB` parameter in `natparm`. The current `OPRB` parameter values are sent to Adabas with the open request. The values are used for wide-character fields and apply to the entire Adabas user session.

For detailed information, see *Unicode Data* in the *Accessing Data in an Adabas Database* part of the *Programming Guide*.

Work Files and Print Files

The following topics are covered below:

- WRITE WORK FILE
- READ WORK FILE
- Special Considerations for Work File Type Transfer
- Print Files

WRITE WORK FILE

The information below applies for the statement `WRITE WORK FILE`. See the *Statements* documentation for detailed information on this statement.

Code Page Data

The following work file types write code page data:

- ASCII and ASCII compressed
- Unformatted
- CSV
- Entire Connection

The work file type and the code page must be defined in the Configuration Utility. For further information, see *Work Files* in the *Configuration Utility* documentation.

All Natural data defined with the operands A (alphanumeric) and U (Unicode) are converted to the specified code page. If a code page has not been specified, all data are converted to the default code page which is defined with the `CP` parameter.

Note: In the work file, all written A and U operand data are in code page format.

If U operand data are to be written into these work files and afterwards read from these work files without loss of data, you have to define UTF-8 as the code page (in the Configuration Utility). In this case, all A and U operand data are written in UTF-8 format. A subsequent `READ WORK FILE` statement where the work file is also configured using code page UTF-8 reads the operand U data without loss of data.

Notes:

1. Work file data which have been written in UTF-8 format can be read by text editors which support UTF-8 (for example, Notepad on the Windows platform).
2. Natural data defined with the operand B (binary) are not converted to the code page which has been specified in the Configuration Utility. These data are written as they are stored in Natural, without any code page conversion.

If one of the above-mentioned work file types is specified and the code page UTF-8 is defined for the work file, the work file attributes `BOM` (write byte order mark) and `NOBOM` (do not write byte order mark) take effect. These attributes can be specified in the **Work Files** category of the Configuration Utility and with the `DEFINE WORK FILE` statement. If the code page UTF-8 is defined for the work file and the work file attribute `BOM` is specified, the UTF-8 byte order mark (hexadecimal representation: `H'EFBBBF'`) is written at the beginning of the work file, in front of the work file data.

If a work file type other than the above-mentioned work file types is used for writing the work file, or if a code page other than UTF-8 is defined for the work file, the specification of the attribute `BOM` is ignored during runtime. The following table shows the runtime behavior during the processing of the statements `WRITE WORK FILE` and `READ WORK FILE`:

Code Page and Attribute Setting	`WRITE WORK FILE`	`READ WORK FILE`
The code page UTF-8 is not specified for the work file (default). The work file attributes `BOM` and `NOBOM` have no effect.	No UTF-8 byte order mark is written. No conversion to UTF-8.	No check for UTF-8 byte order mark. No conversion from UTF-8.
The code page UTF-8 is specified for the work file. The work file attribute `BOM` is specified.	UTF-8 byte order mark is written. A and U fields are converted to UTF-8.	Check for UTF-8 byte order mark. If an UTF-8 byte order mark is found, it is removed from the work file data. A fields are converted from UTF-8 to the default code page. U fields are converted from UTF-8 to the Natural internal runtime representation UTF-16.
The code page UTF-8 is specified for the work file. The work file attribute `NOBOM` (default) is specified.	No UTF-8 byte order mark is written. A and U fields are converted to UTF-8.	Check for UTF-8 byte order mark. If an UTF-8 byte order mark is found, it is removed from the work file data. A fields are converted from UTF-8 to the default code page. U fields are converted from UTF-8 to the Natural internal runtime representation UTF-16.

Binary Data

The following work file types write binary data (for example, UTF-16 for operand format U):

- SAG
- Portable

Natural data defined with the operands A and U are not converted to code page. These data are written to the work file in binary format. For U operand data, this is done in UTF-16.

READ WORK FILE

The information below applies for the statement `READ WORK FILE`. See the *Statements* documentation for detailed information on this statement. Take note of the restrictions that are listed for the `RECORD` option.

Code Page Data

When the following work file types are used, the work file data that are read into Natural U (Unicode) operands are converted from the specified code page to UTF-16.

- ASCII and ASCII compressed
- Unformatted
- CSV
- Entire Connection

Data that are read into A (alphanumeric) operands are converted, if required, from the specified code page to the default code page which has been defined with the parameter `CP`.

If one of the above-mentioned work file types is specified and the code page UTF-8 is defined for the work file, the `READ WORK FILE` statement automatically checks the work file for an UTF-8 byte order mark. If an UTF-8 byte order mark is found at the beginning of the work file, it is removed. The data that are read from the work file are converted from UTF-8 to the default code page.

If data are read from another work file type, the check for a byte order mark is not performed and a byte order mark is therefore not removed.

For information on the runtime behavior during the processing of the statements `WRITE WORK FILE` and `READ WORK FILE`, see the table in the **previous** section.

Binary Data

When the following work file types are used, the work file data are read into the Natural operands A and U without conversion (that is: they are read in binary format):

- SAG
- Portable

The work file type Portable supports endian conversion for data of operand format U.

Special Considerations for Work File Type Transfer

Operand format U is generally supported for the work file type Transfer. If Entire Connection is not able to read or write Unicode for the selected file type, a runtime error message is displayed.

Print Files

The handling for Unicode data in print files depends on the selected logical device's (LPT1 to LPT31) print method, currently either GUI (Windows only) or TTY.

Regardless of the print method, data are passed to the Natural printing services in UTF-16 format. That is, any format A field data will already have been converted to Unicode.

GUI Print Method

With this Windows-only print method, the data are passed to the Windows printer driver in Unicode (UTF-16) format. Because this is the standard method for printing data in Windows, the driver invariably handles this data appropriately. This is therefore the recommended print method under Windows if any characters that are not within the system code page are being used.

TTY Print Method

With this print method, the data are, by default, converted from the internal (UTF-16) format into the system code page. However, by using a printer profile, it is possible to specify that the data should instead either be converted into UTF-8 format, or be subjected to an additional conversion to an arbitrary external code page. For more information on these alternatives, see *Printer Profiles* in the *Configuration Utility* documentation.

The rationale behind the default behavior of converting the data into the system code page is based on the current lack of printers capable of directly accepting raw text files in UTF-8 format.

11 Platform Differences

- General Information 54
- Windows 54
- Linux 55

General Information

On Windows and Linux platforms, Natural has internally been Unicode-enabled. This means that many structures containing strings have Unicode format now. For example, the Natural source area has now Unicode format. For this reason, Unicode data can be handled at runtime in the Natural I/O as well as in the Natural development environment when writing and cataloging Natural code.

Even if Natural is Unicode-enabled internally, all existing data currently has code page format. As a consequence, all this data is converted from code page format to Unicode format when used in Natural Version 6.2 or above. For example, if a source is opened with the program editor, a conversion from the code page file format to the Unicode source area format is performed. Even if you do not use the U format, this is of advantage: you can now see all language-specific characters, no matter which system code page is installed. However, the user is responsible for defining the correct code page information. See *Migrating Existing Applications* for more details.

When cataloging Natural objects, all constants which are not defined with the U prefix are converted to the code page of the corresponding source. If the source has UTF-8 format, these constants are converted to the default code page.

Notes:

1. In most cases, Unicode data requires more memory space than code page data. Therefore, the Natural parameter `USIZE` may need to be increased with Natural Version 6.2 or above.
2. Natural dialogs (editor and runtime) are Unicode-enabled as of Natural Version 6.3.

Windows

Unicode is fully supported in the local Natural for Windows environment.

The editors are Unicode-enabled and it is possible to enter all possible characters. When saving the source, Natural first tries to convert the source to the original code page. If this fails because the source contains characters which are not found in this code page, further processing depends on the setting of the parameter `SUTF8`. If `SUTF8` is `ON`, the source will be saved in UTF-8 format. If `SUTF8` is `OFF`, the user will be asked whether to save the source in the original code page or to cancel the current save. If the user decides to save the source in the original code page, the characters which are not found will be replaced with substitution characters. In addition, it is possible to select a code page explicitly in the **Save As** dialog box.

The program editor has been enhanced in order to support the Unicode bidirectional algorithm.

The output window is also Unicode-enabled. When characters are entered via the keyboard, A format fields accept only the characters which are available in the default code page.

Linux

Full Unicode support is only available with SPoD and the Natural Web I/O Interface. SPoD is necessary for entering Unicode input in Natural sources; the same applies as described above for the local Natural for Windows environment. The Natural Web I/O Interface is necessary for Unicode I/O from Natural applications.

If Natural is used via a terminal emulation, all output will be converted from Unicode to the default code page before displaying it. Characters which are not available in the default code page will be replaced with the substitution character of the default code page. Similar input is only possible on base of the default code page.

Note: Natural sources which have UTF-8 format can no longer be opened with the native Natural for Linux editor.

12 Migrating Existing Applications

- Impact of Unicode on Existing Applications 58
- Migrating Existing Objects 58
- Adding Unicode Support to Existing Applications 59
- Migrating Natural Remote Procedure Calls (RPC) 60

Impact of Unicode on Existing Applications

On Windows and Linux platforms, Natural has internally been Unicode-enabled which means that many structures containing strings have Unicode format now. For example, the Natural source area has now Unicode format. For this reason, data which is only available in code page format is internally converted to Unicode format. This applies, for example, to the Natural sources and to the Natural library names and object names. However, a conversion from code page to Unicode and vice versa can only be performed successfully if the correct code page is used for conversion. Even if an application is not changed but only re-cataloged, the code page information is important because for cataloging an object is loaded into the Natural source area. If all objects are coded in the system code page, no changes are necessary. If the objects are not coded in the system code page, see *Migrating Existing Objects on Windows and Linux Platforms* for further information.

On Windows, the Natural output window has been Unicode-enabled which means that all fields have Unicode format now. In case of A format fields where the code page string length differs from the Unicode string length, the behavior of the Natural output window has changed slightly. This is especially relevant for double-byte code pages where the code page string length is normally twice as long as the Unicode string length. For A format input fields, it is now possible to enter "Unicode-string-length" characters in the field. When leaving the field and the default code page is a double-byte code page, all characters which do not fit into the target A format field are removed.

The internal Unicode structure will most probably need more memory. If you have defined a low value for the profile parameter `USIZE`, it may be necessary to increase this value.

Migrating Existing Objects

Natural has been extended so that the code page information can be defined on several levels:

- The Natural profile parameter `CP` defines the default Natural code page.
- For several objects (Natural sources, Natural batch input/output files, work files of type ASCII, ASCII compressed, Unformatted and CSV) an object-specific code page can be defined.

If neither an object-specific code page nor a default code page is defined, Natural will use the operating system's code page.

Since it is not possible to identify the correct code page automatically, it is important that you define the required code page information yourself. The following scenarios are possible:

Status	Effort	Action
All data is available in the operating system's code page.	No effort	No action.
All data is stored with one code page, but this code page differs from the operating system's code page.	Easy	The Natural profile parameter CP has to be set to the correct code page.
The data is available in different code pages.	Depends on the number of sources and code pages	The correct code page has to be defined for every Natural object: ■ **Sources** If only few objects are affected, change the code page via the **Properties** dialog box. If several objects (for example, an entire library) are affected, use the FTOUCH utility for changing the code page. ■ **Batch Files** Set the Natural profile parameters CPOBJIN, CPSYNIN and CPPRINT to the correct code page. ■ **Work Files** Set correct code page for the work files in the Configuration Utility.
Different code pages are mixed in one object (for example, in a source)	High	The object has to be rewritten in UTF-8 format.

Adding Unicode Support to Existing Applications

It is easy to extend existing applications with new source code based on the U format. The following rules have to be regarded for the U format (as compared with the A format):

- A REDEFINE of U to a format other than U should be avoided because this may result in split characters.
- U format is endian-dependent. This has to be considered when moving between the formats B and U.
- Align U in DEFINE DATA for performance reasons (better performance on Linux).
- Keep in mind that characters may be lost when moving U to A.

If you want to change existing fields from A format to U format, the following rules have to be regarded:

- Code which assumes a specific encoding of strings has to be changed (for example, comparison with a B field).

- All REDEFINE statements of the field have to be checked for their validity.
- A REDEFINE to N is not possible (that is: you will not get the expected result).
- The database field has to be migrated to Unicode (provided that this is supported by your database).
- You may have to change the length of the field: if the A field contains DBCS characters, half the length is required for the U field.

Migrating Natural Remote Procedure Calls (RPC)

The profile parameter CP has been renamed to CPRPC. In earlier Natural versions, CP was used to specify the name of the code page used by the Entire Conversion Service (ECS) and applied only to the Natural RPC (Remote Procedure Call) when the transport protocol ACI (that is, EntireX Broker) was used.

A new CP parameter is available which defines the default code page for Natural data. When you are working with Natural RPC and have previously used the CP parameter dynamically, you have to change this parameter to CPRPC.

When you use parameter files from a previous version, you need not change anything; the Configuration Utility automatically migrates CP to CPRPC.

13 Special Considerations and Limitations

- The dialog editor, which is provided with Natural for Windows, and dialog-based runtime is not Unicode-enabled.
- The editors provided with Natural for Linux are not Unicode-enabled.
- If the `DL` parameter is specified for a field which is longer than 250 characters, a maximum of 250 characters will be displayed in the field.
- A Natural source line may not be longer than 250 bytes. The program editor, which works on Unicode format, checks only that the number of UTF-16 code units is not greater than 250. However, depending on the encoding of the source, the line length may increase when converting the encoding from UTF-16 to the source encoding. For example, the UTF-8 encoding requires up to 4 bytes for a Chinese character; an error will be displayed in this case and the changes will not be saved.
- For Linux, Unicode is only supported at runtime with the Natural Web I/O Interface. If an application is run in the terminal emulation or xterm and Unicode strings are displayed, strange effects may occur.
- Compared with previous Natural versions, the performance is degraded since several conversions between code page and Unicode have to be performed.

14 Help and Troubleshooting

- Receiving the Startup Error "Invalid Code Page Specified" 64
- The Default Code Page 64
- Picking the Right Format When Saving Your Natural Sources 64
- Handling UTF-8 Encoding with Natural Code 64
- Incorrectly Displayed Characters 65
- Receiving an Error When Editing a Natural Source 65
- Receiving an Error When Saving a Natural Source 65
- Finding out the Encoding of a Natural Source 65
- Changing the Encoding of a Natural Source 66
- Converting an Existing Natural Source into UTF-8 Format 66
- Substitution Characters Used When a Character Cannot Be Converted 66
- Using UTF-8 Sources with Previous Natural Versions 66
- Receiving a Conversion Error When Cataloging a Source Which Has UTF-8 Format 67
- Receiving Junk on Linux When Displaying U Format by a Terminal Emulation 67
- Working with a Current SPoD Client and an Older SPoD Server 67
- Working with a Current SPoD Server and an Older SPoD Client 67

Receiving the Startup Error "Invalid Code Page Specified"

The code page you have defined with the profile parameter `CP` does either not exist (see *http://demo.icu-project.org/icu-bin/convexp* for valid ICU code pages and *http://www.iana.org/assignments/character-sets* for the appropriate IANA names) or is an invalid default code page for the platform (for example, an EBCDIC code page cannot be used on a Windows or Linux platform).

The Default Code Page

The default code page is the code page which is the result of the evaluation of the profile parameter `CP`. If `CP` is not filled, it is the current operating system code page.

On the platforms supported by Natural for Linux, you should always define the `CP` parameter, because the ICU default could be defined differently for different Linux platforms and this definition can as well change for a specific platform with newer ICU versions.

The default code page which is used by Natural for conversions between code page and Unicode and vice versa can be detected by displaying the content of the system variable `*CODEPAGE`.

Picking the Right Format When Saving Your Natural Sources

Should you save all Natural sources in UTF-8 format depends on the characters you want to use and on the platforms on which your sources are located. If you want to use Unicode constants, UTF-8 is the only possibility to store all combinations of characters. However, you can define hexadecimal UH constants which can also be stored in code page sources. The disadvantage of hexadecimal constants is that you have to know the UTF-16 encoding for every character of the constant. On mainframes, UTF-8 format for sources is not possible at all. On Linux, UTF-8 sources can only be handled via SPoD; they cannot be handled locally on Linux.

Handling UTF-8 Encoding with Natural Code

Use the `MOVE ENCODED` statement for conversion from UTF-8 to UTF-16: the code page "UTF-8" has to be used for the A format variable.

Incorrectly Displayed Characters

Check if you are using the correct code page. If the code page is correct, check if the selected font supports the characters you want to display.

Receiving an Error When Editing a Natural Source

The code page which is defined for the source is not correct. When converting the contents of the source to Unicode, a conversion error occurs. Change the encoding of the source so that the conversion to Unicode is successful.

Receiving an Error When Saving a Natural Source

You have entered characters in the source which cannot be converted to the code page which was used to read the source. Check if you have entered these characters by mistake or if you really want to save the characters in the source. In the first case, remove the faulty characters and save the source. In the second case, save the source in UTF-8 format or, if the characters are contained in U constants, use UH constants instead.

If you have not entered any characters which are not contained in the code page of the source, check whether the profile parameter `SRETAIN` has been set to `OFF`. In this case, the source will be saved with the default code page. If the concerned source was previously saved with a different code page, a conversion error may occur.

Finding out the Encoding of a Natural Source

To find out the encoding of a Natural source, in Natural Studio, invoke the **Properties** dialog box for the source node. The **General** page shows the encoding of the source. If the **Encoding** text box is empty, no specific encoding is stored for the source. This means that the default encoding is used when reading the source.

The list view windows of Natural Studio also show the encodings of all listed objects.

Changing the Encoding of a Natural Source

In Natural Studio, invoke the **Properties** dialog box for the source node. The **General** page shows the encoding of the source. If this is not the correct encoding, you can change it by choosing the **Change** button: a list of available code pages is shown and you can select the correct encoding for the source.

Converting an Existing Natural Source into UTF-8 Format

Open the source in the Natural editor with the correct code page. Save the source with **Save As** and in the **Save As** dialog box, select UTF-8 as the encoding.

Substitution Characters Used When a Character Cannot Be Converted

Which substitution character is used if a character cannot be converted depends on the direction of the conversion: if a code page character cannot be converted to Unicode, the Unicode substitution character "U+FFFD" is used. If a Unicode character cannot be converted to a code page, the substitution character which is defined by ICU for this code page is used.

For the conversion from Unicode to the default code page, the substitution character can be changed by setting the profile parameter `SUBCHAR`.

Using UTF-8 Sources with Previous Natural Versions

You cannot use UTF-8 sources with previous Natural versions. Previous Natural versions do not know any code page information; a UTF-8 source will be interpreted as the current system code page.

Receiving a Conversion Error When Cataloging a Source Which Has UTF-8 Format

A Natural source with UTF-8 format cannot be cataloged because a code point cannot be converted.

All A constants in a source with UTF-8 format are converted to the default code page when storing them in the generated program. Either remove the characters which are not contained in the default code page from the A constants or use U constants instead of A constants.

Receiving Junk on Linux When Displaying U Format by a Terminal Emulation

All characters which are not contained in the default code page will be replaced with the substitution character of the code page before displaying the output on a terminal emulation. For an ASCII code page, the substitution character defined by the ICU conversion table is often "0x1A", which could be a control character on Linux terminals. It is strongly recommended to use the Natural Web I/O Interface when using U format in I/O statements. If using a terminal emulation is essential, the substitution character (`SUBCHAR`) can be changed to a printable character (for example, "?").

Working with a Current SPoD Client and an Older SPoD Server

You can work with a current SPoD client and an older SPoD server, but you should set the code page of the SPoD client to the code page of the server sources.

See also *Prerequisites for Natural Single Point of Development* at ***http://documentation.software-ag.com/natural/spod_prereq/prereq.htm***.

Working with a Current SPoD Server and an Older SPoD Client

You can work with a current SPoD server and an older SPoD client, but this is not recommended if you have defined encodings for sources.

See also *Prerequisites for Natural Single Point of Development* at ***http://documentation.software-ag.com/natural/spod_prereq/prereq.htm***.