

# Natural

## Unicode and Code Page Support

Version 9.1.2

October 2020

This document applies to Natural Version 9.1.2 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 © 1979-2020 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. 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 AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://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 AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://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 AG.

**Document ID: NATMF-NNATUNICODE-912-20220220**

## Table of Contents

Preface .....	v
1 About this Documentation .....	1
Document Conventions .....	2
Online Information and Support .....	2
Data Protection .....	3
2 Introduction .....	5
About Code Pages and Unicode .....	6
About Unicode and Code Page Support in Natural .....	7
3 Enabling Unicode and Code Page Support .....	9
International Components for Unicode for Software AG (ICS) .....	10
ICS Module SAGICU .....	10
Alternative ICS Modules for Support of Architecture Levels .....	12
Load Modules with Minimum Collation Data .....	13
ICU Data Libraries .....	13
ICU Data Items .....	14
Unicode and Code Page Support for Adabas .....	15
Translation Tables .....	15
Support of Multi-Byte Code Pages .....	16
4 ICS Transition Version 222 .....	17
ICS 222 - Available Functionality .....	18
ICU Data Customization as of ICU 64 .....	18
5 ICS 311 .....	19
Data Scope and Data Handling .....	20
The CFICU STEPLIB Parameter .....	20
Validating the CFICU STEPLIB Parameter .....	21
6 Configuration and Administration of the Unicode/Code Page Environment .....	23
Customizing the ICU Data Library .....	24
Profile Parameters and Macros .....	27
Encoding Information .....	32
7 Development Environment .....	33
Development Environment .....	34
Customizing Your Environment .....	35
Editors in the SPoD Environment .....	35
Code Page Support for Editors, System Commands and Utilities .....	36
Code Page Support for Natural Source Objects .....	38
8 Unicode and Code Page Support in the Natural Programming Language .....	41
Natural Data Format U for Unicode-Based Data .....	42
Statements .....	43
Logical Condition Criteria .....	47
System Variables .....	48
Large and Dynamic Variables .....	48
Session Parameters .....	48
Sample Programs .....	51

9 Unicode Input/Output Handling in Natural Applications .....	53
Displaying and Entering Unicode Data .....	54
Natural Web I/O Interface Client .....	54
10 Bidirectional Language Support .....	59
General Information .....	60
Screen Direction .....	60
Field Direction .....	60
Arabic Shaping .....	62
11 Unicode Data Storage .....	65
Unicode Data/Parameter Access .....	66
Database Management System Interfaces .....	66
Work Files and Print Files .....	67
12 Migrating Existing Applications .....	69
Impact of Unicode on Existing Applications .....	70
Migrating Existing Objects .....	70
Adding Unicode Support to Existing Applications .....	72
Migrating Natural Remote Procedure Calls (RPC) .....	72
13 Frequently Asked Questions .....	73
Why do I get the startup error "Invalid code page specified"? .....	74
What is the "default code page"? .....	74
What default code page is used? .....	74
How can I display all relevant Natural code page settings? .....	74
How can I handle UTF-8 encoding with Natural code? .....	74
Why are some characters not displayed correctly? .....	75
Why do I get an error when I want to edit a Natural source? .....	75
Why do I get an error when I want to save a Natural source? .....	75
How can I find out the encoding of a Natural source? .....	75
How can I change the encoding of a Natural source? .....	76
Which substitution character is used if a character cannot be converted? .....	76
Can I use Natural sources with previous Natural versions that are not code page enabled? .....	76
Index .....	77

---

# Preface

---

This documentation describes how Natural supports Unicode and code pages on mainframe platforms. It also describes how Natural supports bidirectional languages.

This documentation is organized under the following headings:

<b>Introduction</b>	General information on code pages and the Unicode Standard, and on how Unicode and code pages are supported in Natural.
<b>Enabling Unicode and Code Page Support</b>	Information on International Components for Unicode for Software AG (ICS), ICU data libraries, support of code pages, and on transaction tables.
<b>ICS Transition Version 222</b>	Information on the ICS version based on the last service pack of the previously released ICS 221.
<b>ICS 311</b>	Information on ICS 311 based on ICU 66.1 and Unicode 13.0.
<b>Configuration and Administration of the Unicode/Code Page Environment</b>	Information on profile parameters which provide Unicode and code page support, and on the encoding of code page data.
<b>Development Environment</b>	How to customize your environment and how Unicode is handled by the Natural editors. Information on code page support for Natural editors, system commands and Natural source objects.
<b>Unicode and Code Page Support in the Natural Programming Language</b>	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.
<b>Unicode Input/Output Handling in Natural Applications</b>	How to display and enter Unicode data. Information on the Natural Web I/O Interface client which is used in SPoD and runtime environments.
<b>Bidirectional Language Support</b>	How Natural supports bidirectional languages.
<b>Unicode Data Storage</b>	Information on database access, and on the work file types and print files which provide Unicode and code page support.
<b>Migrating Existing Applications</b>	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).
<b>Frequently Asked Questions</b>	Answers to frequently asked questions.

---

# 1 About this Documentation

---

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

## Document Conventions

---

Convention	Description
<b>Bold</b>	Identifies elements on a screen.
Monospace font	Identifies service names and locations in the format <code>folder.subfolder.service</code> , APIs, Java classes, methods, properties.
<i>Italic</i>	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>.

In addition, you can also access the cloud product documentation via <https://www.software-ag.cloud>. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

### Product Training

You can find helpful product training material on our Learning Portal at <https://knowledge.softwareag.com>.



## Tech Community

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

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG 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 AG resources.

## Product Support

Support for Software AG 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

---

- 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-dependant 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.

---

## 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.

If no code page is defined (`CP=OFF`), a default code page is not defined. `CP=AUTO` is intended to adjust the Natural session to the code page of the current I/O device.

- 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 and DB2.

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 *International Components for Unicode for Software AG (ICS)* later in this documentation.



# 3 Enabling Unicode and Code Page Support

---

- International Components for Unicode for Software AG (ICS) ..... 10
- ICS Module SAGICU ..... 10
- Alternative ICS Modules for Support of Architecture Levels ..... 12
- Load Modules with Minimum Collation Data ..... 13
- ICU Data Libraries ..... 13
- ICU Data Items ..... 14
- Unicode and Code Page Support for Adabas ..... 15
- Translation Tables ..... 15
- Support of Multi-Byte Code Pages ..... 16

## International Components for Unicode for Software AG (ICS)

---

Code page conversion and Unicode support make use of functionality provided by International Components for Unicode for Software AG (ICS). If you want to enable Natural for Unicode and code page support, you have to install the components provided with ICS: the **ICS module SAGICU** or an **alternative ICS module** (z/VSE and z/OS only) and *ICU data libraries*.



### Notes:

1. No ICS component must be installed to execute applications without Unicode and code page support, that is, when the profile parameters `CFICU` and `CP` are set to `OFF`.
2. Information on the currently used ICU version and Unicode specification is provided in the main menu of the `SYSCP` utility. See *Invoking and Terminating SYSCP* in the *Utilities* documentation of the Natural for Mainframes documentation.

## ICS Module SAGICU

---

If you want to enable Natural for Unicode and code page support, you need to link and load an ICU data library during the installation of Natural as described in *Installing International Components for Unicode for Software AG* for z/OS (see *ICS Transition Version 222* and *ICS 311*), z/VSE and BS2000.

The ICS module `SAGICU` is intended to be used independently from localization data. It contains no statically-linked code pages and locales. A dataset containing the entirety of the ICU localization data, modulated in data items, is part of the ICS 311 delivery. Its name can be specified by the `CFICU STEPLIB` parameter or statically in the JCL as a Natural `steplib`.

Statically-linked collation data (set of code pages and locale IDs) is still supported and is part of the *ICS Transition Version 222*.

- [Collation Services](#)



- Code Pages and Locales

## Collation Services

Another feature of this module is collation services. Collation services are used to compare Unicode strings. They consider the fact that the alphabetical order varies from language to language. It is a big challenge to accommodate the world's languages and writing systems and the different orders that are used. However, the ICU collation service provides excellent means for comparing strings in a locale-sensitive fashion. For example, in German locale, the character "Ä" is sorted between "A" and "B"; in Swedish locale, it is sorted after "Z". In Lithuanian, the character "y" is sorted between "i" and "k". The ICU implementation of collation services is compliant to the Unicode Collation Algorithm and conforms to ISO 14651. The algorithms have been designed and reviewed by experts in multi-lingual collation, and are therefore robust and comprehensive.

## Code Pages and Locales

Statically-linked collation data (set of code pages and locale IDs) is not supported with ICS 311. It is still supported and is part of the ICS Transition Version 222 (see [ICS Transition Version 222](#)).

ICS 311 uses all of the ICU localization data.

The ICS module `SAGICU` provides the following code pages and locales:

Code Pages	Locales
IBM037	de_DE
IBM273	en_US
IBM1025	es_ES
IBM1026	fr_FR
IBM1047	sv_SE
IBM1097	
IBM01140	
IBM01141	
IBM01145	
IBM01146	
IBM01147	
US (alias for IBM01140)	
DE (alias for IBM01141)	
ES (alias for IBM01145)	
EN (alias for IBM01146)	
FR (alias for IBM01147)	
IBM-37_P100-1995,SWAPLFNL	
IBM-1047_P100-1995,SWAPLFNL	
IBM-1140_P100-1997,SWAPLFNL	
EBCDIC-XML-US	
EDF03DRV (BS2000 code page)	
EDF03IRV (BS2000 code page)	
EDF04DRV (BS2000 code page)	

Code Pages	Locales
EDF04IRV (BS2000 code page)	
EDF041 (BS2000 code page)	
EDF04F (BS2000 code page)	
IBM-290 (Japanese code page SBCS)	
IBM-930 (Japanese code page SBCS/DBCS)	
IBM-939 (Japanese code page SBCS/DBCS)	
IBM-1390 (Japanese code page SBCS/DBCS)	
IBM-1399 (Japanese code page SBCS/DBCS)	
IBM-932 (Japanese code page ASCII MBCS)	
IBM-942 (Japanese code page ASCII MBCS)	
IBM-943 (Japanese code page ASCII MBCS)	
EUC-JP (Japanese code page ASCII MBCS)	
IBM-420 (RTL code page)	
IBM-424 (RTL code page)	
IBM-916 (RTL code page)	


## Alternative ICS Modules for Support of Architecture Levels

---

This section does not apply to BS2000.

If your Natural system runs on z/OS or z/VSE with an IBM processor with architecture level 9 or higher, you can replace the ICS module `SAGICU` by `SAGICUA9`. `SAGICUA9` is built to use advanced machine instructions introduced with IBM's ESA/390 and z/Architecture. You can use the system command `TECH` (see the *System Commands* documentation) to find out the architecture level supported on your current machine.

`SAGICUA9` improves the execution performance, especially for Natural statements that use Unicode variables or code-page encoding instructions (for example, `MOVE ENCODED`). For more information on architecture levels, refer to the related documentation from IBM (z/Architecture, Principles of Operation).

 **Caution:** An operation exception error (abend code S0C1) can occur if the ICS module `SAGICUA9` is used, but the underlying machine architecture level is lower than 9.

## Load Modules with Minimum Collation Data

These modules are not delivered with ICS 311, as the load modules of ICS 311 (SAGICU and SAGICUA9) are already minimal in size and contain no statically-linked ICU localization data.

If ICS Version 222 is installed at your site, you can use the load modules SAGICUM and SAGICUM9 to include only the bare minimum of collation data in the module build. This enables a light-weight configuration and better performance for particular use cases.

## ICU Data Libraries

Data libraries provided by Software AG are not supported with ICS 311. They are still supported and are part of the ICS Transition Version 222 (see [ICS Transition Version 222](#)).

ICS 311 uses all of the ICU localization data.

If you want to enable Natural for Unicode and code page support, you need to link and load an ICU data library during the installation of Natural as described in *Installing International Components for Unicode for Software AG for z/OS, z/VSE and BS2000*.

ICU data libraries are supplied with the following ICS data modules where *nn* denotes the current version of the module as announced in the current Natural *Release Notes* for Mainframes.

Data Module	Description
ICS $SDT_{nnE}$	Contains the most popular code pages and locales. The code pages are already declared in NATCONFIG.
ICS $SDT_{nnJ}$	Same as ICS $SDT_{nnE}$ , but enhanced by Japanese code pages. ICS $SDT_{nnJ}$ is already linked to the ICS module SAGICU (or an <a href="#">alternative ICS module</a> on z/VSE or z/OS). It contains the above mentioned code pages and locales.
ICS $SDT_{nnX}$	<p>Contains all possible converters and locales offered by the currently supported ICU version. It supports about 230 different code pages (predominantly EBCDIC code pages) and 238 locales. Therefore, the module size is huge.</p> <p>ICS<math>SDT_{nnX}</math> supports all code pages and locale IDs which are supported by the currently supported ICU version (see <a href="http://demo.icu-project.org/icu-bin/convexp">http://demo.icu-project.org/icu-bin/convexp</a>).</p> <p><b>Note:</b> Due to technical restrictions, ICS<math>SDT_{nnX}</math> is not delivered for z/VSE and BS2000.</p>

It is possible to create your own ICU data library that exactly matches your requirements (see [Customizing the ICU Data Library](#)).

## ICU Data Items

---

The ICU data items supported by Natural include converters and collators. For example: a converter is used when a `MOVE ENCODED` statement executes, and a collator when strings are compared in an `IF` statement.

An ICU data item is either statically linked to an ICU data library or it is dynamically loaded on request during the Natural session.

ICU data items are supplied as loadable modules on the ICS data set supplied for installation of Natural, and must be accessible through the Natural steplib chain.

When a data item is used for the first time, ICS attempts to open it from the linked or loaded ICU data library. If no data item is associated with a library, ICS attempts to dynamically load the data item from the ICS data set.

This section covers the following topics:

- [Naming Conventions for Data Item Modules](#)
- [ICU Dynamically Loaded Single Data Items](#)

### Naming Conventions for Data Item Modules

The name of a data item module in the ICS data set is restricted to eight characters. As indicated in the table below, it consists of the following:

- A prefix (*I*),
- A two-digit ICU version (*xx*),
- A logical group identifier (*C, B, S, L, M* or *D*), and
- A four-digit sequence number (*nnnn*).

Module Name	Contents
<i>IxxCnnnn</i>	Charset mapping tables (converter modules)
<i>IxxBnnnn</i>	Break iterators
<i>IxxSnnnn</i>	Collators (collation services)
<i>IxxLnnnn</i>	Localization (formatting, display names and other localized data)
<i>IxxMnnnn</i>	Miscellaneous data (rule-based number formats and transliterators)
<i>IxxDnnnn</i>	Base data

Example:

`I58C0074` is the name of a converter for ICU Version 58.2 and code page `ibm-1148_P100-1997`.

However, in a `MOVE ENCODED` statement, Natural expects the long name of the code page that corresponds to the data item module. Any valid alias name of the code page can be used. The name of the code page is automatically mapped to the eight-character short name when the data item module is loaded.

For further information, see the appropriate ICU web site.

### **ICU Dynamically Loaded Single Data Items**

Using dynamically loaded single data item modules allows for extensive flexibility. Data is loaded on demand and supports all code pages. A dataset containing all of the ICU localization data, modulated in single data items, is part of the ICS 311 delivery.

A single data item module is loaded when first accessed (e.g. by a `MOVE ENCODED` statement) and is available for future use instantly without the need to reload. Only the already used code pages will be kept in memory and no statically-linked data or a separate data library as was the case with previous ICS versions.

Single data item modules are especially useful for z/VSE and B2000, as they do not support the extended data library functionality (which was available with previous ICS versions).

## **Unicode and Code Page Support for Adabas**

---

If a Natural session is enabled for code page or Unicode support, you should ascertain that Natural's Adabas user session also uses the appropriate user encoding for accessing Adabas data.

Because Adabas uses Entire Conversion Services (ECS) for conversion, the ECS name must be specified in the related `NTCPAGE` entry in module `NATCONFIG`. To ascertain that Natural's Adabas user session uses the correct code page, specify the `ACODE` and/or `WCODE` option in the `OPRB` parameter for the databases used.

For more information on Adabas Unicode and code page support conversion, see the Adabas documentation for mainframes.

## **Translation Tables**

---

Natural uses various tables for character translation and character property definition. The contents of the tables can be modified via profile parameters (`TAB`, `UTAB1`, `UTAB2` and `SCTAB`) during the start of a Natural session.

If Natural is running with code page support (that is: the `CP` profile parameter is set to a value other than `OFF`), the tables cannot be modified by the user. In this case, the following Natural

startup message will be issued to notify the user that the above mentioned session parameters are not considered:

```
Character translation parameter table-name ignored due to CFICU=ON.
```

Natural adjusts the tables automatically, according to the code page used for the Natural session (value of the system variable \*CODEPAGE). See also *Translation Tables* in the *Operations* documentation.

## Support of Multi-Byte Code Pages

---

Natural supports multi-byte code pages (MBCS) such as IBM-939 which is a Japanese code page based on EBCDIC and DBCS. Multi-byte code pages can be selected using the CP parameter (by setting CP to AUTO (if supported) or to the name of a code page). If Natural is running with a multi-byte code page, it uses internal I/O buffers which are based on Unicode. This means that all data written into the internal I/O buffers by an I/O statement are converted to Unicode. Due to the requirements of Unicode and multi-byte code pages, the size of the I/O buffers is increased as compared to the traditional I/O since Unicode characters need twice as much space as EBCDIC characters and enhanced attributes are needed to describe a field.

In the case of single-byte code pages (SBCS) such as IBM-1140, the traditional EBCDIC-based I/O is still used to preserve resources.

# 4 ICS Transition Version 222

---

- ICS 222 - Available Functionality ..... 18
- ICU Data Customization as of ICU 64 ..... 18

This chapter covers the following topics:

## ICS 222 - Available Functionality

---

The ICS Transition Version 222 is now available to customers in addition to ICS 311. The ICS transition version 222 is based on the last service pack of the previously released ICS 221 with ICU 58.2 and Unicode 9.0. It includes all cumulated fixes of ICS 221 and allows for using ICS without any configuration changes. ICS 222 still allows customers to:

1. Use data libraries provided by Software AG (see [ICU Data Libraries](#)).
2. Create their own data files (see [Customizing the ICU Data Library](#)).

## ICU Data Customization as of ICU 64

---

As introduced by the ICU project, the [ICU Data Build Tool](#), available as of ICU 64, allows for customization of the ICU locale data file and replaces the makefiles. This change no longer guarantees the integrity and completeness of ICU data libraries provided by Software AG with ICS 311. It compels the change of direction for the new ICS 311 and the release of the ICS transition version 222 to ease Natural installations and makes the continuous use of ICS seamless.



# 5 ICS 311

---

▪ Data Scope and Data Handling .....	20
▪ The CFICU STEPLIB Parameter .....	20
▪ Validating the CFICU STEPLIB Parameter .....	21

This chapter covers the following topics:

## Data Scope and Data Handling

---

ICS 311 provides all of the available ICU customization data, modulated in so-called **Data Items**. The dataset containing the data items is part of the delivery, along with the ICS 311 load modules SAGICU and SAGICUA9.

A data item (collator, converter) is located on the disk in the dataset. The data item is loaded in the memory on demand (e.g. by a `MOVE ENCODED` statement). Once a data item is loaded, it is available for future use instantly without the need to reload.

This allows for the ICS 311 load modules SAGICU and SAGICUA9 to be kept *minimal in size*.

A new SYSCP function to list all loaded data items has been introduced (see SYSCP function *Loaded Code Pages* in section *Utilities > SYSCP Utility - Code Page Administration*).

## The CFICU STEPLIB Parameter

---

Data files provided by Software AG are no longer supported with ICS 311. ICU localization data is loaded only dynamically from a dataset containing the data items (collators, converters, etc.).

The name of this dataset can be:

- *statically* specified in the JCL as a Natural Steplib.
- *dynamically allocated* with the CFICU STEPLIB parameter (see section *The CFICU STEPLIB Parameter in Unicode and Code Page Support > ICS 311*).

ICS uses both allocation methods to search for data items, starting from the CFICU STEPLIB dataset (if given) and the statically specified Natural Steplibs in the JCL.

This dynamic approach allows for flexibility. The JCL must not be changed in order to run Natural, only a Natural session parameter must be added.

The dataset given by the CFICU STEPLIB parameter will be dynamically allocated only once by the first Natural session in a given TP system under the DD Card ICSxxxDD (xxx corresponds to ICS version) and used by all Natural sessions afterwards.

Example: `CFICU=(STEPLIB='I311ITEMS.LOAD')`

---

## Validating the CFICU STEPLIB Parameter

---

The first Natural session in a given TP system will try to validate the `CFICU STEPLIB` parameter. If the validation is successful, ICS will initialize. All following Natural sessions in the TP system will *disregard* the `CFICU STEPLIB` parameter and use the dynamically allocated dataset of the first session. Validation criteria for the `dataset name` include the following:

- the dataset must exist.
- the dataset name must be enclosed in '' - see example above.
- the dataset name must conform to the [z/OS naming conventions](#):
  - maximal length of 44 characters
  - no special characters allowed
  - must not be a high level qualifier, i.e. must contain at least one dot '.'

If the validation is not successful, Natural error `NAT3414 STEPLIB DSN <data set name> cannot be loaded` will be issued. The next Natural session will try to validate the `STEPLIB` parameter and initialize ICS again. This iteration will continue until a successful ICS initialization is reached. All sessions after a successful ICS initialization will disregard the `STEPLIB` parameter and use the already allocated resource.



# 6 Configuration and Administration of the Unicode/Code Page

## Environment

---

- Customizing the ICU Data Library ..... 24
- Profile Parameters and Macros ..... 27
- Encoding Information ..... 32

**Notation *vr*:**

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

## Customizing the ICU Data Library

---

ICU makes use of a wide variety of data tables to provide many of its services. Examples include converter mapping tables, collation rules, transliteration rules, break iterator rules and dictionaries, and other locale data. The ICU data library for Natural is provided as a package that contains the desired data items. The usage of packages instead of single data item files increases the performance since there is only one file access during the initialization to load the package. However, it is not so flexible since it requires a rebuild of a package if data items need to be added.

The ICU data library may be customized in order to add existing or new converter mapping tables or to add other data items such as collation rules, break iterator rules and other locale data.

The customization tool for the ICU data library is available from the Download Components area in Empower (<https://empower.softwareag.com/>). Use the supplied *icudtvr.zip* file (*vr* = **version**) to customize the data libraries for the ICU version required in your Natural environment: ICS Version 2.2 requires *icudt58.zip*. The files described in this section are contained in the *icudtvr.zip* file.

Several steps are required to create a new ICU data library package. Some steps are performed on a PC and other steps are performed on the appropriate mainframe platform.

The following topics are covered below:

- [Obtaining New Data Items](#)
- [Compiling Conversion Tables and Locales](#)
- [Creating a New Package](#)
- [Transferring the Assembler Source to the Mainframe Platform](#)
- [Using the New Data Library on the Target Mainframe Platform](#)

### Obtaining New Data Items

There are different sources for new data items:

- The ICU Data Library Customizer at <http://apps.icu-project.org/datacustom/>.
- ICU converter data archive at <http://source.icu-project.org/repos/icu/data/trunk/charset/data/ucm/>.
- User-defined converter mapping tables.

The ICU Data Library Customizer is a web-based tool provided by IBM. The ICU Data Library Customizer is version-dependent. Therefore, an ICU Data Library Customizer is provided for

each supported version. The ICU version can be retrieved using the `SYSCP` utility (see the function **ICU Information**).

The Data Library Customizer displays the data items in a tree view. Primarily, all data items are selected. It is possible to deselect all items by expanding the advanced options and choosing the **Deselect All** button. Expanding a tree shows all available items of that type. It is possible to reduce the amount of displayed items by using the **Filter Items** button of the advanced options. For example, to reduce the tree view to show only items for Japanese countries, enter the string "japanese" in the text box and choose the **Filter Items** button. Now, several items can be selected or deselected. All selected items are later added to the delivered base ICU data library and will be available for Natural.

The second possibility is to obtain or create a `.ucm` (source) mapping data file for the desired converter. A large archive of converter data is maintained by the ICU team. This archive is version-independent. If the desired conversion table is not available in the archive, it is possible to create a new one. For the documentation of the layout of converter mapping tables (`.ucm` files), refer to the chapter *Conversion Data* of the *ICU User Guide* (<http://userguide.icu-project.org/conversion/data>). It is recommended to take a similar mapping table from the archive, rename it and adjust it to the new requirements.



**Note:** It is forbidden to change the character mapping of an existing converter. In fact, this is the creation of a new converter and requires a new converter name to avoid confusion.

Detailed information on how to customize the ICU data library is provided in the `readme.txt` file which is part of the downloaded zip file.

## Compiling Conversion Tables and Locales

Converter source files are compiled into binary converter files (`.cnv` files) by using the `makeconv.exe` tool. It is possible to specify more than one converter.

Example:

Command	Description
<code>makeconv ↵ ibm-1142_P100-1997.ucm</code>	Compile the Danish character mapping table of code page IBM-1142 into the binary format. With a subsequent step, the output file <code>ibm-1142_P100-1997.cnv</code> can be added to the new data library package.

Converters obtained from ICU are already registered in the alias name table. No additional step is necessary. If the converter source file is a user-defined file, there will be no appropriate entry in the alias name table. In this case, it is necessary to register the new code page in the alias name table. Open the text file `convtrs.txt` and append an appropriate entry at the end of that file in the section "User defined code pages". The name of the code page is required and the IANA name is optional. The string `{ IANA* }` declares `iana-name` as an IANA name. Each user-defined code page requires an entry in the alias name table.

The entry has the following format:

```
name-of-code-page iana-name { IANA* }
```

If the code page "my\_cp-100" with the IANA name "MYCP" is to be added, the following line in *convrts.txt* is required:

```
my_cp-100 MYCP { IANA* }
```

For more information, refer to the header of *convrts.txt* or to the *ICU User Guide*.

The modified alias name table has to be compiled with *gencnval.exe* into a binary file (*cnvalias.icu*) to be linked to the new data library package.

### Creating a New Package

A new package is created with the tool *makepkg.bat*. It uses the delivered package *icudtvr.dat* (*vr = version*) and merges new, user-defined items. A user-defined item can be an additional package that contains new data items, a single data item such as a new converter (*.cnv* file), or a text file that contains a list of new items.

Examples:

Command	Description
makepkg icudtvr1.dat	Add the data items contained in <i>icudtvr1.dat</i> to the base package <i>icudtvr.dat</i> .
makepkg ibm-1142_P100-1997.cnv	Add the Danish code page IBM-1142 to the base data library package <i>icudtvr.dat</i> .
makepkg newitems.txt	Add all data items (converters) that are listed in the text file <i>newitems.txt</i> to the base data library package <i>icudtvr.dat</i> .

*makepkg.bat* produces two files, a big-endian EBCDIC-based binary file and an HL assembler source. The assembler source contains the binary image of the first file packed into `DC X' . . . '` statements. The name of the binary file is *icudtvre.dat* and the name of the assembler source is *icudtvre\_dat.s*. The file *icudtvre* is a copy of *icudtvre\_dat.s*. The files must never be renamed since the package name "icudtvre" is used as a part of internal references of data items. It is used by the ICU runtime to access data items such as converters and to validate the data file. "icudt" identifies an ICU data file, "vr" is the *version* and "e" identifies the file as big-endian EBCDIC-encoded.

If more than one item is to be added or if the alias name table has been changed, the items have to be declared as a list in the *newitems.txt* file.

Examples:



**■ Add code pages `ibm-939_P120-1999` and `ibm-942_P12A-1999`**Entries of *newitems.txt*:*ibm-939\_P120-1999.cnv**ibm-942\_P12A-1999.cnv***■ Add user defined code page `my_cp-100`**Entries of *newitems.txt*:*cnvalias.icu**my\_cp-100.cnv*

For more information, refer to the *ICU User Guide*.

**Transferring the Assembler Source to the Mainframe Platform**

The result of the previous step is an assembler source module. The assembler module with the new data library package *icudtore* has to be transferred to the target platform. The File Transfer Protocol (FTP) is available on each PC and can be used for this task. Ask the system administrator for the required information (such as host name, port number, user name and password) for accessing the target machine via FTP. Since *icudtore* is a text file, the transfer mode must be set to ASCII to ensure the correct translation of the file on the target platform. The name of the file on the target platform is arbitrary. However, it is recommended to use the name *icudtore*. If it is desired to rename *icudtore*, the *renamepkg.bat* tool has to be used.

**Using the New Data Library on the Target Mainframe Platform**

The assembler source module must be assembled and linked on the target platform. It can either be linked to the nucleus or it can be loaded dynamically with `RCA=name` and `CFICU=(DATFILE=name)`.

**Profile Parameters and Macros**

---

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

Unless otherwise noted, the profile parameters and macros mentioned in this section are explained in detail in the *Parameter Reference*.

Parameter or Macro	Description
CFICU or NTCFICU macro	<p>Enables Unicode support for various Unicode settings.</p> <p>See also <a href="#">CFICU Parameter</a> and <a href="#">CFICU and CP: Session Modes</a></p>
CMPO or CPAGE keyword subparameter of NTCMPO macro	<p>Generates code page-sensitive Natural programs.</p> <p>See also <a href="#">CPAGE Compiler Option</a>.</p>
CP	<p>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).</p> <p>Only platform-suitable code pages can be used. This means, for example, that no ASCII code page can be defined for a mainframe platform. An initialization error message occurs if a wrong code page is used.</p> <p>See also <a href="#">CFICU and CP: Session Modes</a>.</p>
CPCVERR	<p>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.</p> <p>This parameter is not regarded for the conversion of Natural sources when loading them into the source area or when cataloging them.</p> <p>It is not regarded whether a Unicode field is converted into the code page before an I/O on a terminal emulation. In this case, the substitution character defined by ICU is replaced by the placeholder character which is defined in NATCONFIG.</p>
CPOBJIN	<p>Specifies the code page in which the batch input file for data is encoded. This file is defined in the data set CMOBJIN.</p>
CPPRINT	<p>Specifies the code page in which the batch output file shall be encoded. This file is defined in the data set CMPRINT.</p>
CPSYNIN	<p>Specifies the code page in which the batch input file for commands is encoded. This file is defined in the data set CMSYNIN.</p>
NTCPAGE macro	<p>In the NATCONFIG module, this macro defines a code page and all related information, such as placeholder character, locale ID and collation tables.</p> <p>See also <a href="#">NTCPAGE Macro</a>.</p> <p>NTCPAGE and NATCONFIG are explained in detail in the <i>Operations</i> documentation.</p>
OPRB or NTOPRB macro	<p>Sets the ACODE and/or WCODE option to define the user encoding if the used Adabas database is enabled for UES (universal encoding support).</p>
PRINT or CP keyword subparameter of NTPRINT macro	<p>Defines the code page for a report.</p>
SRETAIN	<p>Specifies that all existing sources have to be saved in their original encoding format. See also <a href="#">Customizing Your Environment</a>.</p>

See also:

- *Natural in Batch Mode* in the *Operations* documentation.
- For valid code pages, see <http://www.iana.org/assignments/character-sets>.

This section covers the following topics:

- [CFICU Parameter](#)
- [CFICU and CP: Session Modes](#)
- [CPAGE Compiler Option](#)
- [NTCPAGE Macro](#)
- [Natural Development Server](#)

## CFICU Parameter

The parameter `CFICU` and its subparameters are explained in detail in the *Parameter Reference*. Some of the subparameters have an impact on the performance.

If collation services are used to compare Unicode strings, both strings are checked whether they are normalized or not. The check itself consumes a lot of CPU time. If you are sure that the strings are already normalized, you can switch off the check (`COLNORM=OFF`).

In Unicode, it is possible to represent the same character as one code point or as a combination of two or more code points. For example, the German character "ä" can be represented by "U+00E4" or by the combination of the code points "U+0061" and "U+0308". The conversion from Unicode to, for example, IBM01140 treats combined characters as single code points and produces an "a" followed by a substitution character since code point "U+0308" is not represented in the target code page. With `CNVNORM=ON`, a normalization is performed right before the actual conversion. The normalization consumes additional CPU time and temporary storage. If you are sure that no combining characters are involved in `MOVE` statements (except `MOVE NORMALIZED`), you should set `CNVNORM` to `OFF` to increase performance. Note that all possible combinations are represented by a single coded Unicode code point.

Conversion from Unicode to code page and vice versa is not high-performance. The reason is that the ICU implementation is written in C++ and that it covers nearly all Unicode, code page and language aspects in the world. However, some code pages can be mapped to Unicode (and vice versa) via translation tables to accelerate conversion. Accelerator tables are activated with the `CPOPT` subparameter. If it is set to `ON`, Natural automatically creates two accelerator tables during session initialization by using ICU conversion functions. The first table (with a size of 512 bytes) is used for conversion from code page to Unicode and the other table (with a size of 65535 bytes) is used for conversion from Unicode to code page. During a Natural session, all conversions are then executed via the accelerator tables instead of ICU calls. Accelerator tables are only provided for the default code page (`*CODEPAGE`). Temporary code pages (for example, in `MOVE ENCODED` statements) do not use accelerator tables if the module `NATCPTAB` is not linked. If it linked, up to 30 accelerator tables based on the ICU database are used to speed up performance.

## CFICU and CP: Session Modes

The parameters `CFICU` and `CP` can be used to adjust Natural to specific purposes:

Settings	Description
<code>CFICU=OFF, CP=OFF</code>	Compatibility mode. For running existing applications without Unicode and without code page support. Legacy translation tables are used for I/O translation. Compared with former versions, there is no significant increase in resource consumption (CPU time and buffer usage). This mode does not need the ICS module <code>SAGICU</code> (or an <b>alternative ICS module</b> on z/VSE and z/OS) to be linked to the Natural nucleus.
<code>CFICU=ON, CP=OFF</code>	For new applications that are using Unicode and code page conversion ( <code>MOVE ENCODED</code> ) but not default code page support. Therefore, the system variable <code>*CODEPAGE</code> is empty. It is possible to use U format variables, but it is not possible to use, for example, <code>MOVE A TO U</code> , since this requires the default code page information. The error <code>NAT3411</code> will be issued indicating that no default code page is available.
<code>CFICU=ON, CP=value*</code>	For new applications that are using full Unicode as well as code page support.
<code>CFICU=OFF, CP=value*</code>	This combination does not make sense, because code page support needs ICU services for conversion. Therefore, <code>CFICU=ON</code> is enforced in this case and a session initialization message is issued.

\* where *value* is any value other than `OFF`.

## CPAGE Compiler Option

The compiler option `CPAGE` creates objects that can be executed with a code page which is different from the code page used at creation time. This means that all alphanumeric constants of the object which are coded with the code page at creation time have to be converted to the code page which is active at execution time. To make it possible for the Natural object loader to find and convert alphanumeric constants, an additional table is created by the compiler. This increases the size of the generated object, depending on the number of used alphanumeric constants. The conversion at runtime consumes additional CPU time. If the default code page (value of the system variable `*CODEPAGE`) is the same as the code page at creation time or if the session has no default code page (`CP=OFF`), no conversion is done. Conversion errors are ignored, independent from the setting of the parameter `CPCVERR`. If the compiler option `CPAGE` is set to `OFF`, no conversion is performed at runtime and the alphanumeric constants are treated as they are.

The following sample program is cataloged with code page `IBM01141` (German) and is executed with default code page `IBM01140` (us). The characters "Ä", "Ö" and "Ü" are defined in both code pages, but at different code points.

Example 1 - `CPAGE=OFF`:

```

OPTIONS CPAGE=OFF
WRITE *CODEPAGE 'ÄÖÜ'
END

```

Output with code page IBM01140 (us):

```

Page      1
IBM01140                                ¢\!

```

**Example 2** - CPAGE=ON:

```

OPTIONS CPAGE=ON
WRITE *CODEPAGE 'ÄÖÜ'
END

```

Output with code page IBM01140 (us):

```

Page      1
IBM01140                                ÄÖÜ

```

## NTCPAGE Macro

The most common standard for code page names is the IANA name. Therefore, the system variable \*CODEPAGE contains the IANA name of the default code page. On z/VSE and z/OS, a code page is qualified by its Coded Character Set ID (CCSID). On BS2000, the Coded Character Set Name (CCSN) is most popular. Currently, Adabas uses the Entire Conversion Service definition (ADA ECS). The macro NTCPAGE can be used to assign these different names to the unambiguous IANA name. NTCPAGE is part of the Natural configuration module (NATCONFIG).

It does not matter whether the IANA name, the CCSID/CCSN or the alias name is entered with the CP parameter. The alias name can be a user-defined name which is used to assign a more significant name to the code page. In any case, \*CODEPAGE contains the IANA name of the selected code page.

In addition, a placeholder character can be defined for a code page. It overwrites the default substitution character of that code page, which is normally a non-displayable character (for example, H'3F' in an EBCDIC code page). The placeholder character can be used to avoid that non-displayable characters are sent to terminals.

Example:

```
NTCPAGE IANA=IBM01140,CCSID=1140,ECS=1140,ALIAS='US',PHC=003F
```

The values IBM01140, 1140 or US can be entered with the CP parameter to activate the code page. \*CODEPAGE contains the name IBM01140. The substitution character of the code page will be replaced by "U+003F", which is a quotation mark (?).

The number of available code pages depends on the used ICU data library.

All code pages defined in the currently used data package can be used by Natural. An NTCPAGE entry is only necessary if an alternative alias name or placeholder character is desired.

### Natural Development Server

The following configuration parameter is available with Natural Development Server (NDV):

Settings	Description
TERMINAL_EMULATION=WEBIO	Specifies that the Natural Web I/O Interface client (which supports Unicode) is used for input and output.

---

## Encoding Information

The code page information of the object is part of the object directory displayed with the LIST system command. For details, see *Displaying Directory Information* in the *System Commands* documentation.

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.

### 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).

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

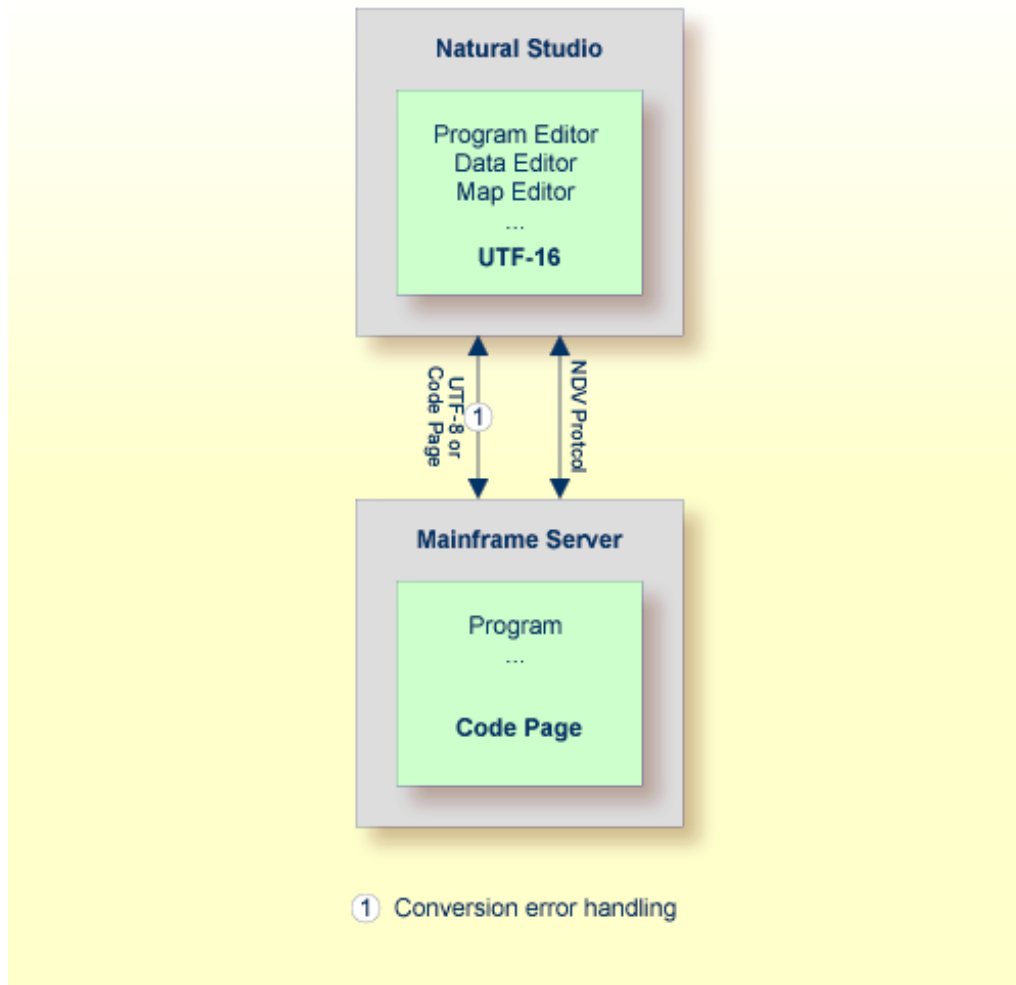
# 7 Development Environment

---

▪ Development Environment .....	34
▪ Customizing Your Environment .....	35
▪ Editors in the SPoD Environment .....	35
▪ Code Page Support for Editors, System Commands and Utilities .....	36
▪ Code Page Support for Natural Source Objects .....	38

## Development Environment

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



---

## Customizing Your Environment

---

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.

---

## Editors in the SPoD Environment

---

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



**Note:** The editors provided with Natural for Mainframes provide code page support. See [Code Page Support for Editors, System Commands and Utilities on the Mainframe](#).

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).

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.

## Code Page Support for Editors, System Commands and Utilities

---

The following topics are covered below:

- [Editors](#)
- [System Commands and Utilities](#)

### Editors

The program, map and data area editors are not Unicode-enabled. Instead the sources are stored with code page information. According to the setting of the profile parameter `SRETAIN`, Natural sources with code page information may be converted automatically from the current code page of the source into the default code page of the current Natural session (value of the system variable `*CODEPAGE`) if the source is loaded into the editor. If there are any characters that cannot be converted, a window displays a code point conversion error and asks for substitute values for those code points that cannot be converted. The display of this message is independent from the current setting of the parameter `CPCVERR`. In this case, the user can decide to open the editor with or without converting the source into the default code page. Saving or stowing a converted source will save the new code page information. Sources without code page information (for example, sources that have been saved or stowed with previous Natural versions) are loaded into the editors without any conversion. According to the setting of the profile parameter `SRETAIN`, the current code page information of the source will be retained.

Inserting sources with the `.I` command or the split screen function will also convert sources, if necessary, according to the setting of the profile parameter `SRETAIN`. If characters cannot be converted, the defined substitution character will be inserted instead.

The check and conversion of the source is performed when the editor is started, not when the program is loaded into the source area. If a program is executed via `RUN program-name`, a conversion is not performed. This causes different behavior, depending on whether `RUN program-name` is entered on the `NEXT` screen or on an editor screen. If `RUN program-name` is entered on the `NEXT` screen, no conversion follows; if it is entered on an editor screen, the editor is started right after the execution of the program and a conversion is performed.

See the table below for the code page that is assigned to an existing Natural source that is saved or stowed, depending on the values of the profile parameters `SRETAIN` and `CP`.

Original Source Code Page Information	Setting of SRETAIN	Source Code Page Information after SAVE or STOW if CP is Set to a Value other than OFF	Source Code Page Information after SAVE or STOW if CP is set to OFF
Source without code page information	SRETAIN=ON SRETAIN=(ON, EXCEPTNEW)	No code page information	No code page information
Source without code page information	SRETAIN=OFF	Code page resulting from evaluation of CP	No code page information
Source is encoded in <i>code page 1</i>	SRETAIN=ON SRETAIN=(ON, EXCEPTNEW)	Original code page ( <i>code page 1</i> )	Original code page ( <i>code page 1</i> )
Source is encoded in <i>code page 1</i>	SRETAIN=OFF	Code page resulting from evaluation of CP	Original code page ( <i>code page 1</i> )

The table below shows the code page that is assigned to a new Natural source that is saved or stowed, depending on the values of the profile parameters SRETAIN and CP.

Setting of SRETAIN	Source Code Page Information after SAVE or STOW if CP is Set to a Value other than OFF	Source Code Page Information after SAVE or STOW if CP is set to OFF
SRETAIN=ON	Code page resulting from evaluation of CP	No code page information
SRETAIN=OFF	Code page resulting from evaluation of CP	No code page information
SRETAIN=(ON, EXCEPTNEW)	No code page information	No code page information

## System Commands and Utilities

### LIST

By default, the system command LIST displays sources as they are stored in the system file without any conversions.

The CONVERTED option of the LIST command converts the source into the default code page (value of the system variable \*CODEPAGE) if the code page information of the source is provided. All non-convertible characters are then replaced by the defined substitution character.

## LIST DIR

The system command `LIST DIR` shows the used code page information of a Natural source in the directory window.

## SCAN

Similar to the editors, the system command `SCAN` converts the sources before executing the actual `SCAN` command.

## Object Handler (SYSOBJH)

The Object Handler unloads and loads sources with different code page information and preserves the original code page information.

The transfer format option `UTF-8` converts sources from any code page to UTF-8 format while unloading, and stores information about the original code page in the work file. The corresponding load function converts the source back to the original code page or to another code page, if specified. This option can also be used to provide code page information for sources which have been saved or stowed with previous Natural versions and which therefore do not contain any code page information.

Unload and load sources in internal format will keep the code page information, if available.

## SYSCP Utility - Code Page Administration

The `SYSCP` utility can be used to obtain information on code pages and to check or change the code page assignment of a source.

## Code Page Support for Natural Source Objects

---

The Natural compiler, the editors and the Natural system file do not support object sources that are encoded in Unicode. Unicode constants coded in an object source are saved in the default code page, and the cataloged object contains the Unicode code points. The only way to define Unicode constants which do not have an equivalent in the default code page is to use hexadecimal definitions (UH).

Since Natural sources are not converted to Unicode or UTF-8 before saving, they can still be read by previous Natural versions. Code page information is stored in the header of the source. The code page information in the header is simply ignored if a source is accessed by a Natural version which is not code page enabled.

- [Programs, Data Areas, Maps and Map](#)
- [DDMs](#)

- [Error Messages](#)
- [Help Texts](#)

## Programs, Data Areas, Maps and Map

These object sources are not stored in Unicode format but in the default code page of the current Natural session. The name of the code page is stored in the directory of the source. Therefore, as compared to previous Natural versions, the size of a source remains unchanged. But there is a check by the editor whether the code page of the source is equal to the default code page of the Natural session. If the code pages are different, the source is converted into the default code page with the possibility of conversion errors. If a character of the source is not mapped in the default code page, a window appears in the editor to allow manual conversion of the failed characters. For example, a program which has been created with code page IBM01140 contains the following line:

```
WRITE '100 €'
```

If the program is edited again with Natural running with code page IBM037, a conversion error occurs since the character "€" is not mapped in code page IBM037.

Note that the conversion is done when the editor is started and not when the source is loaded.

## DDMs

DDMs are not stored in Unicode format but in the default code page of the current Natural session. The name of the code page is stored in the directory of the DDM. Note that there is no DDM on the system file. As compared to previous Natural versions, the size of a DDM increases slightly. When reading a DDM, there is a check by the editor whether the code page of the DDM is equal to the default code page of the Natural session. If the code pages are different, the DDM is converted into the default code page with the possibility of conversion errors. If a character of the DDM is not mapped in the default code page, a window appears in the editor to allow manual conversion of the failed characters. For example, a DDM which has been created with code page IBM01140 contains the following line:

```
* 100 €
```

If the DDM is edited again with Natural running with code page IBM037, a conversion error occurs since the character "€" is not mapped in code page IBM037.

## **Error Messages**

Natural error messages are not stored in Unicode format but in the default code page of the current Natural session. The name of the code page is stored in an additional Adabas field on the system file. There is a check by the `SYSERR` utility and by user exits whether the code page of the error message is equal to the default code page of the Natural session. If the code pages are different, the error message is converted into the default code page. Errors will be ignored - this means, the substitution character (or if defined, the placeholder character) will be used.

## **Help Texts**

Help texts are always maintained with code page IBM01140 (English). They are not stored with a code page definition. If the default code page of the Natural session is not IBM01140, the help text is converted into the default code page. Errors will be ignored - this means, the substitution character (or if defined, the placeholder character) will be used.

# 8 Unicode and Code Page Support in the Natural Programming

## Language

---

▪ Natural Data Format U for Unicode-Based Data .....	42
▪ Statements .....	43
▪ Logical Condition Criteria .....	47
▪ System Variables .....	48
▪ Large and Dynamic Variables .....	48
▪ Session Parameters .....	48
▪ Sample Programs .....	51

## 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'00C4007000660065006C'
```

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 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, for example:

```
EXAMINE S FOR P WITH DELIMITER D REPLACE R
```

where S is U format, and P, D and R are A format.

In the above example, the variables P, D and R are temporarily converted into the target format U before the actual execution of the `EXAMINE` statement. The conversion from Unicode to code page or vice versa requires calling an ICU function. The conversion requires additional computing time and additional memory. This disadvantage is even greater with very large variables. To avoid frequent conversions, it is recommended that you use only one format within one statement. When all operands in the above example are specified in either U format or A format, a conversion is not necessary. However, if you choose to specify only U operands, this variant will be slower since (due to its nature) this operand type consumes more resources; one character is then coded with 2 bytes (instead of 1 byte which is used with A format).

With a conversion (especially from U format to A format), there is always the risk that characters cannot be represented in the target code page. For example, you want to convert the Unicode character "U+05D0" (Hebrew letter Alef) into the code page IBM01140 (English). Since this character is not contained in the code page IBM01140, either the substitution character for this code page is used, or the placeholder which was specified when defining the code page in `NATCONFIG`. When the parameter `CPCVERR` is set to `ON`, an error message will be issued in this case, indicating a conversion error. In any case, the original information will be lost.

The following statements are particularly affected when using Unicode:

- `MOVE NORMALIZED`
- `MOVE ENCODED`
- `EXAMINE`
- `PARSE XML`
- `REQUEST DOCUMENT`
- `DEFINE PRINTER`

- [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.

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:

â



**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'âüçñüçüü' 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
-----
aပိငခိ်bနိငယိ်d      1              1              1
aပိငခိ်bနိငယိ်d      2              2              2
aပိငခိ်bနိငယိ်d      3              4              1
aပိငခိ်bနိငယိ်d      4              5              3
aပိငခိ်bနိငယိ်d      5              8              1
aပိငခိ်bနိငယိ်d      6              9              3
aပိငခိ်bနိငယိ်d      7             12              1
aပိငခိ်bနိငယိ်d      8             13              3
aပိငခိ်bနိငယိ်d      9              0              0

```

## PARSE XML

The document to be parsed is always internally converted to UTF-16 (if the document is not already encoded in UTF-16).

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

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

## DEFINE PRINTER

The `DEFINE PRINTER` statement provides a `CODEPAGE` clause to provide for conversion of print report data into a code page different from the default code page (value of the system variable `*CODEPAGE`).

## 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 <i>Display Length for Output - DL Parameter</i> in the <i>Programming Guide</i> .
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, 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€	
CAUDAL	ALBERT	167,350€	
VERDIE	BERNARD	170,100€	

## Sample Programs

The library SYSEXPB 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.



# 9 Unicode Input/Output Handling in Natural Applications

---

- [Displaying and Entering Unicode Data](#) ..... 54
- [Natural Web I/O Interface Client](#) ..... 54

The Natural runtime environment is enabled for Unicode support. Unicode characters are converted to the default code page (value of the system variable \*CODEPAGE) before they are displayed on the terminal. Unicode characters which have no equivalent in the default code page are replaced by a substitution character.

With the Natural Web I/O Interface under SPoD, Unicode characters are fully supported by the terminal emulation. In this case, U format fields are displayed and can be entered correctly as Unicode. They are not converted to the equivalent in the default code page. The Natural Web I/O Interface is activated by the Natural Development Server configuration parameter `TERMINAL_EMULATION=WEBIO`. The system variable \*DEVICE contains `BROWSER`.

## Displaying and Entering Unicode Data

---



### Notes:

1. Unicode data cannot be displayed on 3270 terminals.
2. When running applications with Natural for Mainframes, see [Natural Web I/O Interface Client](#) below.

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. However, it is strongly recommended that you use the Natural Web I/O Interface when displaying characters not contained in the default code page.

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

---

Full Unicode I/O is only supported at runtime with the Natural Web I/O Interface. If an application is run in the terminal emulation and Unicode strings are displayed, some Unicode characters may not be displayed correctly.

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:

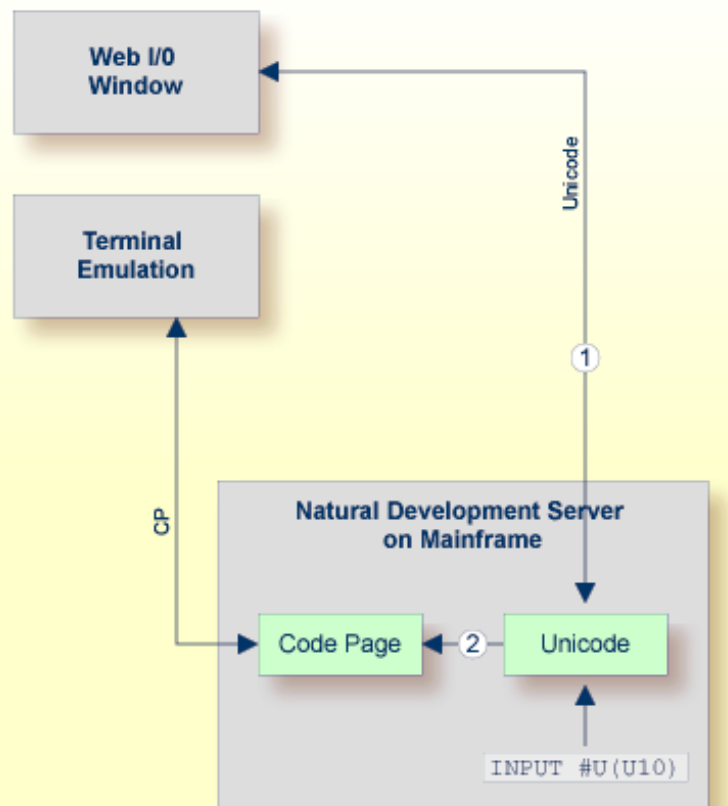
- SPoD Environment
- Runtime Environment

## 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 mainframe environments.

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



- ① NDV parameter `TERMINAL_EMULATION=WEBIO`
- ② NDV parameter `TERMINAL_EMULATION` not specified

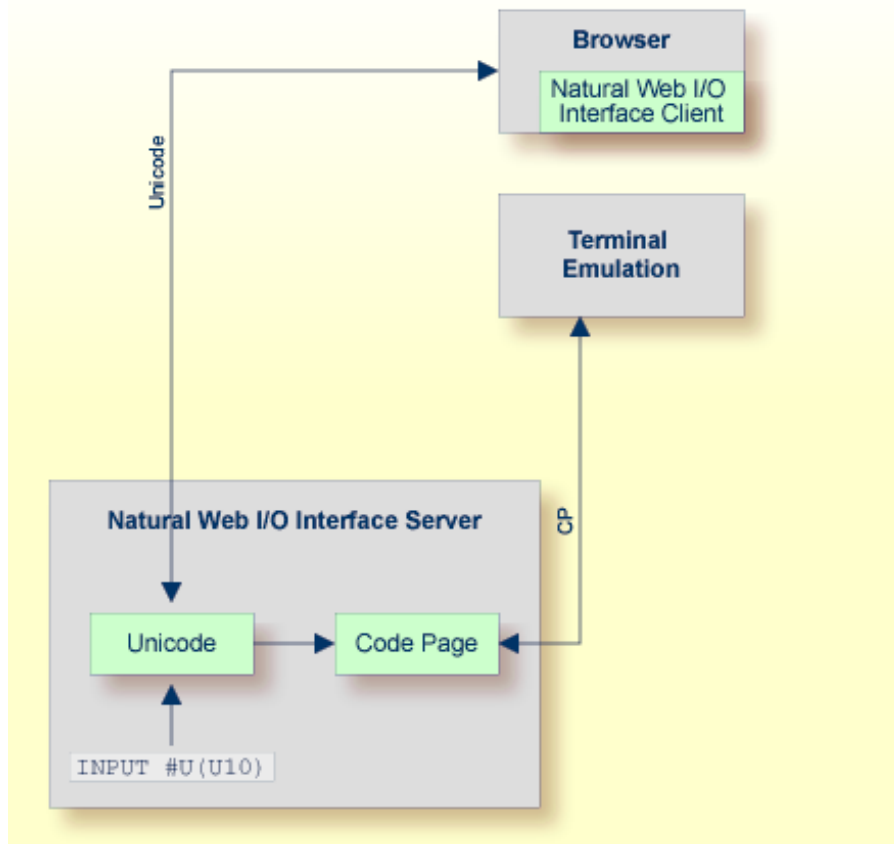
So that the Natural Web I/O Interface client can be invoked, the Natural Development Server has to be configured as follows:

- 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.

### 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.

It is required that the Natural Web I/O Interface server has been installed and configured. See *Natural Web I/O Interface*. Moreover, the Web I/O terminal converter module `NATWEB` must be linked

to the Natural nucleus. The Natural profile parameter `TMODEL` can be used to determine the user screen size.





# 10 Bidirectional Language Support

---

▪ General Information .....	60
▪ Screen Direction .....	60
▪ Field Direction .....	60
▪ Arabic Shaping .....	62

## 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.

Support for bidirectional languages is not activated automatically; the user always has to specify all required parameters (for example, `PM=I`) as described below.

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

The profile parameter `D0` (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 `D0`, 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, system variables and PF key lines are automatically inverted by Natural so that they are displayed correctly from right-to-left if the screen direction is right-to-left.

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=0) TEST2 (AD=0)

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 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. 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  ѵ 1 1  1 2 3
```

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 123'

SET CONTROL 'voff'

INPUT TEST (AD=0) /
      TEST (AD=0 PM=I)

SET CONTROL 'von'

INPUT TEST (AD=0) /
      TEST (AD=0 PM=I)
END
    
```

This program produces two identical screens because the statements SET CONTROL 'voff' and SET CONTROL 'von' do not apply to alphanumeric fields. Both screens look as follows:

```

TEST abc 123
TEST          321 cba
    
```

## Arabic Shaping

---

In Arabic text, all characters of a string are normally connected with each other. For this reason, Arabic characters have up to 4 presentation forms: the isolated, the final, the initial and the medial form. The form that will be used depends on the position of the character in the string. For example, the Arabic character "MEEM" has the following forms in Unicode:

U+0645	م	ARABIC LETTER MEEM
U+FEE1	م	ARABIC LETTER MEEM ISOLATED FORM
U+FEE2	م	ARABIC LETTER MEEM FINAL FORM
U+FEE3	م	ARABIC LETTER MEEM INITIAL FORM
U+FEE4	م	ARABIC LETTER MEEM MEDIAL FORM

Moreover, some characters are combined to a new form if they appear consecutively in a string. This is called a “ligature”. For example, the characters

U+0644	ﻝ	ARABIC LETTER LAM
U+0627	ﺀ	ARABIC LETTER ALEF

have the following combined form:

U+FEFB	ﻝﺀ	ARABIC LIGATURE LAM WITH ALEF ISOLATED FORM
--------	----	---



Unicode strings should include only the Arabic characters in the Arabic block (U+0600 through U+06FF) or the Arabic Supplement block (U+0750 through U+077F); it is not recommended to use the presentation forms in regular Arabic text. It is up to the user interface to display the correct shapes of the characters.

“Shaped” means that every Arabic base character is converted to the appropriate Arabic presentation form. The string may contain each of the four presentation forms of a character. For example, if U+0645 (ARABIC LETTER MEEM) is used as the last character of a string, it is converted to U+FEE2 (ARABIC LETTER MEEM FINAL FORM).

“Unshaped” means that each character is represented only by its basic form. For example, instead of U+FEE2 (ARABIC LETTER MEEM FINAL FORM), U+0645 (ARABIC LETTER MEEM) is used. The conversion to the correct presentation form is performed by the rendering engine of the output device.

Natural strings are internally represented as unshaped alpha or Unicode strings. If strings are displayed with a browser using the Natural Web I/O Interface client or the `PROCESS PAGE` statement, no transformation is required since the rendering engine of the browser takes care of the correct presentation. Incoming strings from such devices are already unshaped and can be directly passed to Natural. If a string is displayed on a terminal such as 3279 or a terminal emulator such as IBM Personal Communications, it must be converted into the shaped form since the terminal itself does not take care of the correct presentation. Accordingly, incoming strings are in the shaped form and must be transformed into the unshaped form to be processed correctly by Natural. The most popular code page for Arabic terminals on the mainframe is IBM420. Compared to Unicode, the number of characters is reduced and not each form of a character is contained. The conversion of strings into IBM420 substitutes unavailable forms of a character by a similar presentation form. For example, the medial form of the Arabic letter MEEM (U+FEE4) is substituted by the initial form (U+FEE3) of the character.

In the Arabic EBCDIC code page IBM420, the Arabic character "MEEM" is represented by the following presentation forms:

H'BA'		ARABIC LETTER MEEM
H'BB'		ARABIC LETTER MEEM INITIAL FORM

### Arabic Tail Fragment

The Arabic characters SEEN (U+0633), SHEEN (U+0634), SAD (U+0635) and DAD (U+0636) (Seen Family) are displayed on terminals as two bytes if they appear in the final form. Code page IBM420 contains a so-called "Arabic tail fragment" that completes the final form of a Seen Family character on terminals or terminal emulators. Of course, the Arabic tail fragment needs an additional position on the screen. The Arabic tail fragment is not required by the browsers. If a string with the final form of a Seen Family character is entered in a browser (Natural Web I/O Interface client or PROCESS PAGE statement) and subsequently displayed on a terminal, the Arabic tail fragment is appended to the string with the consequence that the length of the string increases. If a string with the final form of a Seen Family character is entered via a terminal or terminal emulator and subsequently displayed in a browser, the Arabic tail fragment is removed from the string.



**Note:** For more information about control of character shaping, see *SHAPED - Control of Character Shaping* in the *Parameter Reference* documentation.

# 11 Unicode Data Storage

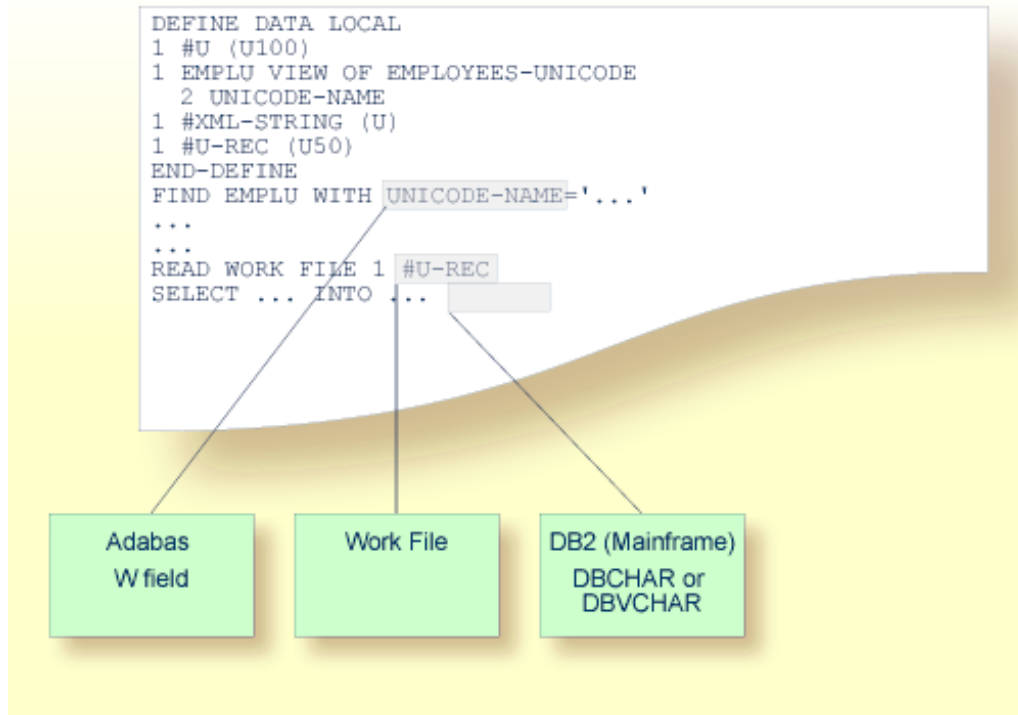
---

- Unicode Data/Parameter Access ..... 66
- Database Management System Interfaces ..... 66
- Work Files and Print Files ..... 67

## Unicode Data/Parameter Access

---

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



## Database Management System Interfaces

---

The following topics are covered below:

- [Accessing Unicode Data in an Adabas Database](#)



- [Accessing Unicode Data in a DB2 Database](#)

## 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.

This encoding is specified with the `OPRB` parameter and is sent to Adabas with the open request. It is used for wide-character fields and applies 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*.

## Accessing Unicode Data in a DB2 Database

Natural enables users to access `CHAR` and/or `WCHAR` fields in a DB2 database as Unicode data.

See also *Natural for DB2* in the *Database Management System Interfaces* documentation.

## Work Files and Print Files

---

The following topics are covered below:

- [Work Files](#)
- [Print Files](#)

### Work Files

No special consideration is given to Unicode data when writing or reading work files. Like all other data types, Unicode data is written and read as is, without conversion.

## Print Files

When sending Unicode data to print files, one or two conversion steps take place.

In a first step, Unicode data contained in a print line is converted to the default code page of the session. As a consequence, all characters which are not contained in this default code page are replaced with the substitution character.

Before passing this converted print line to the actual print access method, it is additionally checked whether a code page has been specified for the logical printer. This may have been accomplished with the `CODEPAGE` operand of the `DEFINE PRINTER` statement or the `CP` subparameter of the `PRINT` parameter. If such a code page has been given, the whole print line (not only the Unicode part of it) is converted accordingly in a second step.

The converted print line is passed to the access method, which means that print access methods do *not* receive Unicode data.

Example:

```
DEFINE PRINTER (1) CODEPAGE 'IBM01140'  
WRITE (1) 'HELLO' U'WORLD'  
END
```

# 12 Migrating Existing Applications

---

- Impact of Unicode on Existing Applications ..... 70
- Migrating Existing Objects ..... 70
- Adding Unicode Support to Existing Applications ..... 72
- Migrating Natural Remote Procedure Calls (RPC) ..... 72

## Impact of Unicode on Existing Applications

---

There is no impact of Unicode on existing applications. This means that existing Natural applications should execute without any changes. Make sure that the parameters `CFICU` and `CP` are set to `OFF`. In this case, it is not necessary to install any of the components supplied with International Components for Unicode for Software AG (ICS). Only the I/O buffers have been noticeably increased since the attributes have been enhanced to support potential Unicode fields. If `CP` is set to `OFF`, the system variable `*CODEPAGE` is cleared and the well-known translation tables (such as standard table or alternative table) are continued to be used for I/O translations.

## 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, print reports, Adabas files) an object-specific code page can be defined.

If neither an object-specific code page nor a default code page is defined (that is, `CP=OFF` applies), Natural does not convert any data.

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 <code>CP</code> has to be set to the correct code page. Make sure that the I/O device supports this code page. <code>CP=AUTO</code> forces Natural to run with the code page of the I/O device.
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: <ul style="list-style-type: none"> <li>■ <b>Sources</b> Save each object in the session with the correct code page.</li> </ul>

Status	Effort	Action
		<ul style="list-style-type: none"> <li>■ <b>Batch Files</b> Set the Natural profile parameters CPOBJIN, CPSYNIN and CPPRINT to the correct code page.</li> <li>■ <b>Adabas Files (ECS enabled)</b> Set the Natural profile parameter OPRB with the ACODE option.</li> </ul>
Different code pages are mixed in one object (for example, in a source)	High	The object has to be rewritten in the appropriate code page format.

Sources which have been saved or stowed with previous Natural versions do not have code page information. The code page field of the directory is empty.

Since Natural sources are not saved in Unicode format, the source has to be converted into the default code page (value of the system variable \*CODEPAGE) that applies to the session. If code page support is switched off (CP=OFF), the code page information of the source is ignored and no conversion is performed. Alphanumeric constants have to be adjusted to the default code page when they are loaded into the source area.

Since Natural sources are not saved in Unicode format, alphanumeric constants have to be adjusted to the default code page during start of the object. This can be achieved with the CPAGE compiler option. If CPAGE is set to ON, an additional table is generated into the object. The Natural loader uses this table to convert every alphanumeric constant to the default code page (value of the system variable \*CODEPAGE). Depending on the amount of alphanumeric constants, the additional table increases the size of the resulting object and the conversion consumes additional CPU time.

It is important that dependent objects (for example, a program and a local data area used by the program) use the same code page. If dependent objects use different code pages, it should be ascertained that the used characters (for example, "#") are mapped to the same code points in the used code pages. The following objects and data do not have an associated object-specific or data-specific code page:

- Data definition modules (DDMs),
- Predict rules,
- Predict XRef data.

Care should be taken if such data is used in or produced by objects for which an object-specific code page has been defined. If the application itself does not necessarily have to be code page enabled and you want the application to be code page sensitive with respect to the data that is being processed, you should consider to use the profile parameter SRETAIN with the value (ON, EXCEPTNEW).

## 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.
- 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 parameter `CP` is used in conjunction with the parameter macro `NTCPAGE` (in the source module `NATCONFIG`) to specify the name of the default code page for Natural data or to automatically take the code page name from the user terminal.

The parameter `CPRPC` is used with the profile parameter `RPC` and the corresponding macro `NTRPC`.

# 13

## Frequently Asked Questions

---

- Why do I get the startup error "Invalid code page specified"? ..... 74
- What is the "default code page"? ..... 74
- What default code page is used? ..... 74
- How can I display all relevant Natural code page settings? ..... 74
- How can I handle UTF-8 encoding with Natural code? ..... 74
- Why are some characters not displayed correctly? ..... 75
- Why do I get an error when I want to edit a Natural source? ..... 75
- Why do I get an error when I want to save a Natural source? ..... 75
- How can I find out the encoding of a Natural source? ..... 75
- How can I change the encoding of a Natural source? ..... 76
- Which substitution character is used if a character cannot be converted? ..... 76
- Can I use Natural sources with previous Natural versions that are not code page enabled? ..... 76

## Why do I get 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 ASCII code page cannot be used on a mainframe platform).

Check whether the same IANA name, CCSID/CCSN or alias name as specified in `NATCONFIG` is used.

## What is the "default code page"?

---

The default code page is the code page which is the result of the evaluation of the profile parameter `CP`.

## What default code page is used?

---

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`.

## How can I display all relevant Natural code page settings?

---

Use the system command `CPINFO`.

## How can I handle 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.



## Why are some characters not displayed correctly?

---

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.

## Why do I get an error when I want to edit a Natural source?

---

The source is saved with the code page at creation time. You get a conversion error when the source could not be converted from the code page of the saved source into the code page of the current Natural session. You can start Natural with the code page of the source to avoid conversion or you can adjust non-convertible characters in the window which appears when the editor is started.

## Why do I get an error when I want to save a Natural source?

---

If you are connected to a mainframe environment via SPoD, the source from the mainframe is converted and edited in Unicode in the SPoD environment. If it is saved, it has to be converted into the code page of the Natural server. A conversion error may occur if a Unicode character is not mapped in the code page of the Natural server session.

If you are in a native Natural for Mainframes environment (without SPoD) you do not get errors when saving a source since a conversion is not performed. The source is saved with the code page information of the current Natural session.

## How can I find out the encoding of a Natural source?

---

Code page information is part of the Natural source directory. Use the `LIST DIR` command to display the directory.

## **How can I change the encoding of a Natural source?**

---

You should start your Natural session with the desired code page using the `CP` parameter. Set the parameter `SRETAIN` to `OFF`, edit the source and save it. Now the source has the modified code page information. Or, you can use the `SYSCP` utility to check or change the code page assignment of a source.

## **Which substitution character is used if a character cannot be converted?**

---

The substitution character of the code page or, if specified in the configuration file, the placeholder character is used.

## **Can I use Natural sources with previous Natural versions that are not code page enabled?**

---

With previous Natural versions that are not code page enabled, it is possible to access sources that have been saved with code page information. The layout of the source has not been changed and the code page information will simply be ignored if the source is accessed with a previous version.

# Index

---

## A

application  
add Unicode support, 72

## B

bidirectional language support, 59

## C

CALLNAT  
statement, 47  
code page, 32  
code page support, v  
constants  
for Unicode, 42

## D

default code page, 7, 32  
DEFINE PRINTER  
statement, 47

## E

encoding of a code page, 32  
EXAMINE  
statement, 45

## I

ICU library, 10

## L

logical condition criteria  
U format, 47

## M

MOVE ENCODED  
statement, 44  
MOVE NORMALIZED  
statement, 44

## P

PARSE XML  
statement, 46  
print files  
Unicode support, 67  
profile parameters  
important for Unicode, 27

## R

REQUEST DOCUMENT  
statement, 46

## S

SAGICU, 10  
SAGICUA9, 12  
session parameters  
important for Unicode, 48  
statements  
important for Unicode, 43  
system code page, 7, 32  
system variables  
important for Unicode, 48

## U

U  
format for Unicode-based data, 42  
Unicode support, v

## W

work files  
Unicode support, 67

