Unicode and Code Page Support in the Natural Programming Language

Unicode and Code Page Support in the Natural Programming Language

This chapter covers the following topics:

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

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-dependant. 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 place holder which was specified when defining the code page in NATCONFG (mainframe only). 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. On Windows, UNIX and OpenVMS platforms, the default substitution character (defined by ICU) for the conversion from Unicode to the default code page (`CP`) can be changed with the profile parameter `SUBCHAR`.

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

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

EXAMINE

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

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

ậ

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

A "supplementary code point" is a Unicode code point between "U+10000" and "U+10FFFF". A supplementary code point is in UTF-16, represented by a surrogate pair which consists of two code units where the first value of the pair is a "high-surrogate code unit", and the second is a "low-surrogate code unit". Such characters are generally rare, but some are used, for example, as part of Chinese and Japanese personal names, and therefore support for these characters is commonly required for government applications in East Asian countries.

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

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

Example:

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

MOVE U’a▯cȫbŭcȫd ’ 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
--------------- ---------------- ---------------- --------------

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

PARSE XML

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

On mainframe platforms, 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

On mainframe platforms, 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`). On Windows, UNIX and OpenVMS platforms, the `DEFINE PRINTER` statement does not provide such a clause; if the `CODEPAGE` clause is defined, it is ignored on Windows, UNIX and OpenVMS platforms.

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

This section covers the following topics:

- *CODEPAGE
- *LOCALE

*CODEPAGE

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

*LOCALE

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

Large and Dynamic Variables

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

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

Session Parameters

The following session parameters are available:

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

Session parameters marked with an asterisk (*) in the above table are only available on Windows, UNIX and OpenVMS platforms.

DL versus AL

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

Example:

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

The above code results in the following output:

```
Page      1 ...

computer
電腦系統
```

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

Example:

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

The above code results in the same output as above:

```
Page      1 ...

computer
電腦系統
```

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

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

The parameters EMU, ICU, LCU and TCU (which are only available on Windows, UNIX and OpenVMS platforms) 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 will assure that the Euro sign is always displayed correctly, no matter what code page is installed on the PC.

Example for EMU:

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

The above code results in the following output:

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

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

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

Sample Programs

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

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