OpenVMS National Character Set Utility Manual

Order Number: AA–PS6FA–TE

May 1993

This manual describes how to use the National Character Set Utility.

Revision/Update Information:	This manual supersedes the *VMS National Character Set Utility Manual*, Version 5.2
Software Version:	OpenVMS AXP Version 1.5 OpenVMS VAX Version 6.0

Digital Equipment Corporation
Maynard, Massachusetts

May 1993

ZK4438

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface

Intended Audience

This manual is intended primarily for system programmers and application programmers.

Document Structure

This document consists of the following three sections:

- Description—This section is in Part I. It provides a description of the National Character Set (NCS) Utility and detailed instructions on how to build NCS definition files.
- Usage Summary—This section is in Part II. It outlines the following NCS information:
  - Invoking the utility
  - Exiting from the utility
  - Directing output
  - Restrictions or privileges required
- Qualifiers—This section is in Part II. It describes the NCS qualifiers, including format, parameters, and examples.

Associated Documents

For related information about the NCS Utility, see the following documents:

- *OpenVMS DCL Dictionary*
- *OpenVMS Programming Concepts Manual*
- *OpenVMS User's Manual*
- *OpenVMS Command Definition, Librarian, and Message Utilities Manual*

Conventions

In this manual, every use of VMS means both the OpenVMS AXP and the OpenVMS VAX operating system.

The following conventions are used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.

$\boxed{\text{Return}}$	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in a VMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in online versions of the manual.
italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error *number*), command lines (for example, /PRODUCER=*name*), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Part I

Introduction to the National Character Set (NCS) Utility

This part of the document introduces you to the National Character Set (NCS) Utility and provides details about building NCS definition files.

NCS Description

In processing strings, two common functions are collating and conversion. Collating sequences provide a means of comparing strings for sorting purposes; conversion functions provide a means for deriving an altered form of an input string based on a conversion algorithm.

The National Character Set (NCS) Utility provides a common facility for defining collating sequences and conversion functions, registering them as definition modules in an NCS library, and providing a means for making them locally accessible to application programmers and system programmers. See Appendix A for a listing of the definition modules that are included in the default NCS library.

Typically, NCS collating sequences are selective subsets of the Multinational Character Set. NCS collating sequences find widespread use in international applications. For example, a Spanish collating sequence would resolve sorting weights for characters you might encounter when processing strings from the Spanish language.

NCS permits you to build collating sequences unique to your particular programming situation. For example, if you want the character *C* to collate between the characters *G* and *H*, you can retrieve the particular collating sequence from a central library, make the appropriate modifications, and then store the modified collating sequence in a local library that is readily accessible to your application program.

You can implement NCS from an application program using the program interface that includes 10 callable NCS routines. The callable routines permit you to access collating sequences and conversion functions stored as definition modules in an NCS library. You can also use the NCS routines to save the definitions in a local library for subsequent use by comparison and conversion routines called from the application program. For instructions on using the NCS routines, see the *OpenVMS Utility Routines Manual*.

This manual provides information about how to use NCS interactively, using the DIGITAL Command Language (DCL) interface for doing the following library operations:

- Creating NCS libraries
- Inserting definition modules in NCS libraries
- Replacing definition modules in NCS libraries
- Extracting definitions from NCS library modules
- Deleting definition modules from NCS libraries
- Other tasks associated with library operations, including listing the library contents, directing the output of the library, and logging library activities

By default, DCL attempts to replace definition modules in the default NCS library with definitions from a specified source file. That is, if you specify *no* qualifiers, DCL assumes you want to replace definition modules in the default NCS library. If you specify no definition source file, DCL prompts you for one, as shown in the following example:

```
$ NCS
_File:
```

Any other library activity requires you to use one or more command qualifiers. You may designate any NCS library as the object of a command by assigning it the logical name NCS$LIBRARY.

The rest of Part I provides instructions for building NCS definition files.

1 How to Build an NCS Definition File

The following subsections describe the language syntax you must use when building NCS collating sequence definitions and NCS conversion function definitions. You can use any standard text editor to build definition files.

All definition files have the following characteristics:

- Include one or more definitions.
- Have no restriction on including a mix of collating sequence definitions and conversion function definitions in a single definition file.
- Use the file type NCS.
- Have comments delimited by an exclamation point (!).
- Do not permit line continuation for comments.

1.1 Naming NCS Definition Files

Each definition file should begin with a name that describes its contents. For example, if you want to convert strings of multinational characters to lowercase, use a name like *MULTI_TO_LOWER*. You can name NCS definition files using up to 31 characters including the letters *A* through *Z* and *a* through *z*, the numbers *0* through *9*, the dollar sign ($), and the underscore character (_). Note that spaces are *not* allowed and that you may use either uppercase or lowercase letters when formulating definition names.

1.2 Structuring a Definition File

Each definition file consists of the following elements, in this order:

1. Definition name
2. Equal sign (=)
3. Definition expression
4. Terminating semicolon (;)

A definition file has the following format:

```
definition_name = definition_expression;
definition_name = definition_expression;
.
.
.
definition_name = definition_expression
```

Example NCS–1 includes excerpts from a definition file that provides several EDT conversion functions.

Example NCS–1 Typical Definition File

```
! Define the EDT fallback conversion function, and UNEDT, its inverse.
! Note that applying EDT and then UNEDT to a file may not result in the
! original file, if, for example, the original file contains the string "^A".
!
EDT_VT2xx = CF(
    CF = _IDENTITY,
        MODIFICATIONS=(
        %X00 = "^@",
        %X01 = "^A",
        %X02 = "^B",
   .
   .
   .
        %XF0 = "<XF0>",
        %XFE = "<XFE>",
        %XFF = "<XFF>"));
UNEDT_VT2xx = INVERSE(EDT_VT2xx);
!
!Start of definition EDT (a modified version of EDT_VT2xx)
!
EDT = CF(
    CF = EDT_VT2xx,
    MODIFICATIONS=(
        %XA0 = "<XA0>",
        %XA1 = "<!!>",
        %XA2 = "<C/>",
   .
   .
   .
        %XFB = "<u^>",
        %XFC = "<u"">",
        %XFD = "<y"">"));
!
!Definition UNEDT (the complement of EDT)
!
UNEDT = INVERSE(EDT);
!
!The next 2 definitions - EDT1 and UNEDT1 -  are structured such that ...
!             UNEDT1(EDT1(string))
!... always gives the original string.
!
EDT1 = CF(CF=EDT,MODI=(
        "^" = "^=",
        "<" = "<="));
!
!The complement of EDT1
!
UNEDT1 = INVERSE(EDT1);
```

Section 1.3 describes the notation used in NCS definitions. The expressions and keyword clauses that comprise NCS definitions are described in Sections 1.5 and 1.6.

1.3 Notation Guidelines

Table NCS–1 lists the language notation to be used in NCS definition files.

Table NCS–1 NCS Language Notation

Notation	Meaning
!	Starts a comment
( )	Establishes entity grouping for ordering operations.
=	In a collating sequence clause, reads literally *left term collates as right term*; in a conversion function clause, reads literally *left term converts to right term*
>	In a collating sequence clause, reads literally *left term collates just greater than right term*
<	In a collating sequence clause, reads literally *left term collates just less than right term*
+	Within a clause, indicates string concatenation; in an expression, indicates an appended collating sequence
*	Indicates a series of expressions comprising one definition
–	Indicates a range of string data
" "	Encloses literal string data
%	Indicates the start of a string numeric value
D	Indicates a decimal numeric value
H	Indicates a hexadecimal numeric value
O	Indicates an octal numeric value
,	Delimits expressions and clauses
;	Terminates an NCS definition

You can construct a definition in free form using indentation and one item per line to improve readability. For example, the following definition is functionally acceptable, but is difficult to read:

```
FRENCH_NRC_TO_MULTI =
CF (CF=_IDENTITY,MODIFICATIONS=
("#"="£","@"="à","["="°",
"\"="ç","]"="§","{"="é",
"|"="ù","}"="è","^a"="â",
"^e"="ê","^o"="ô","^u"="û",
"~a"="ä","~e"="ë","~o"="ö")
```

ZK−6571−GE

Using indentation and one item per line makes the same definition much easier to read as shown in the next illustration:

```
FRENCH_NRC_TO_MULTI = CF(
          CF = _IDENTITY,
          MODIFICATIONS=(
                  "#"  = "£",
                  "@"  = "à",
                  "["  = "°",
                  "\"  = "ç",
                  "]"  = "§",
                  "{"  = "é",
                  "|"  = "ù",
                  "}"  = "è",
                  "^a" = "â",
                  "^e" = "ê",
                  "^o" = "ô",
                  "^u" = "û",
                  "~a" = "ä",
                  "~e" = "ë",
                  "~o" = "ö"))
```

ZK−6572−GE

When you format a file for readability, do not omit the required punctuation.

Strings can be represented literally and numerically. If you use literal strings (as in the preceding example), you must enclose them in double quotation marks (").

If you choose to represent strings numerically, you can use either decimal, hexadecimal, or octal numbers. You must precede numeric values by a percent sign (%) and the appropriate radix symbol (D for decimal, X for hexadecimal, O for octal). Note that you can use only digits that represent a single character, and that the radix notation must be less than decimal 256, regardless of the numbering system you use.

For example, the following statement from the previous example may be coded using a literal string or a numeric string:

```
"~e" = "ë",  ! literal string
"~e" = "%XEB",  ! "ë" represented as hexadecimal number
```

Numeric strings are particularly appropriate when processing characters from either the ASCII subset or the EBCDIC subset.

1.4 Built-In Collating Sequences and Conversion Functions

NCS includes two built-in definitions: the _NATIVE collating sequence and the _IDENTITY conversion function. You use the built-in definitions as a basis for creating other collating sequences and conversion functions. The built-in definitions are distinguished by the leading underscore in their names. Note that the built-in collating sequence and the built-in conversion function are not stored in the NCS library and cannot be modified.

The _NATIVE collating sequence collates strings by ascending numeric value. For collating purposes, the null character (NUL) has the lowest value of all characters in the set.

Following is an example of how you might use the _NATIVE collating sequence to specify a collating sequence:

```
MY_CS = _NATIVE * (MULTI_TO_NODIACRITICALS * MULTI_TO_LOWER)
```

The _IDENTITY conversion function reproduces each input string character as an output string character, except for characters that are explicitly being modified. Following is an example of using the _IDENTITY conversion function:

```
MY_CF = CF(CF = _IDENTITY,
                MODIFICATIONS=("ME"    =   "YOU"
                               "NOW"   =   "THEN"
                               "BLUE"  =   "RED")
```

Using this conversion function, the input string *Tell me now the sky is blue* is converted to the output string *Tell you then the sky is red*.

1.5 Definition Expressions

You can define collating sequences and conversion functions using various types of expressions. Section 1.5.1 describes the types of expressions used in formulating collating sequences. Section 1.5.3 describes the types of expressions used to define conversion functions.

Note that vertical ellipses are used in some examples to indicate omitted code.

1.5.1 Collating Sequence Expressions

You can define collating sequences using any combination of the following types of expression:

- The name of an existing collating sequence
- A sequential series of collating sequences
- An expression that includes appended collating sequences
- A modified collating sequence
- A reversed collating sequence
- A reordered collating sequence

1.5.1.1 Definition Name You can create a new collating sequence by equating it to the name of an existing collating sequence, using the following format:

new_collating_sequence = name_of_existing_collating_sequence,

Following is an example of this type of expression:

```
MY_COLLATING_SEQUENCE = MULTINATIONAL_1,
```

1.5.1.2 Sequential Series of Expressions You can create a new collating sequence from a sequential series of expressions, including one or more conversion functions and an existing collating sequence, in the following format:

new_collating_sequence = collating_sequence * conversion_function_1 * conversion_function_2,

NCS processes the conversion functions first, going from right to left, and then applies the collating sequence. All processes are completed within a single pass.

Following is an example of using a sequential series of expressions to create a collating sequence:

```
MY_COLLATING_SEQUENCE = CS(CS = _NATIVE * MULTI_TO_NODIACRITICALS * MULTI_TO_UPPER),
```

The conversion functions convert uppercase letters to lowercase letters and strip the diacritical marks. The result is then combined with the _NATIVE collating sequence to derive a collating sequence that behaves as though the input strings have been converted and then compared by their numeric value.

1.5.1.3 Expression with Appended Collating Sequences You can create a collating sequence using an existing collating sequence and up to two appended collating sequences, in the following format:

new_collating_sequence = collating_sequence_1 + collating_sequence_2 + collating_sequence_3,

NCS processes the leftmost collating sequence in the first pass, and proceeds to process each of the remaining collating sequences going from left to right, using an individual pass for each.

You can include conversion functions with each of the collating sequences using the asterisk (*) operator, in the following format:

new_collating_sequence = collating_sequence_1 * conversion_a * conversion_b + ... + ...,

Although there is no limitation on the number of conversion functions you can use with each collating sequence, you should try to minimize the complexity of the expression by limiting the number of conversion functions. In a complex expression having several collating sequences with associated conversion functions, NCS applies conversion functions only to the related collating sequence. When NCS detects a distinction between the two strings being compared, the comparison function terminates.

Following is an example of creating a collating sequence using an existing collating sequence with an appended collating sequence:

```
MY_COLLATING_SEQUENCE = CS(CS = MULTINATIONAL_1 + MULTINATIONAL_2 * UPCASE * NODIACRITICALS),
```

1.5.1.4 Modified Collating Sequence You can create a collating sequence from a modified collating sequence using keyword clauses, in the following format:

new_collating_sequence = CS(keyword_clause,keyword_clause, ...),

The expression begins with a definition identifier (CS) followed by several keyword clauses enclosed in parentheses and separated by commas. The first keyword clause identifies the collating sequence that serves as a basis for the new collating sequence, and the second keyword clause lists the appropriate modifications.

Following is an example of an expression that uses keyword clauses to create a new collating sequence by modifying an existing collating sequence. Each uppercase character gets the same collating weight as the associated lowercase character.

```
MY_COLLATING_SEQUENCE =
           CF(CF = _IDENTITY,              ! Base collating sequence
             MODIFICATIONS=(
                %X41-%X5A = %X61-%X7A,     ! Modifications
                %XC0-%XCF = %XE0-%XEF,     ! to the base
                %XD1-%XDD = %XF1-%XFD));   ! collating sequence
```

1.5.1.5 Reversed Collating Sequence You can create a new collating sequence by specifying the reverse order of an existing collating sequence, using the following format:

new_collating_sequence = REVERSE(existing_collating_sequence)

Using this form of expression, you might create a collating sequence where the letter *C* would collate greater than *B*, and the letter *B* would collate greater than *A*.

Here is an example of using a reverse order collating sequence expression:

```
MY_COLLATING_SEQUENCE = REVERSE(_NATIVE),
```

Using this expression, you could give collating weight precedence to lowercase characters over uppercase characters.

1.5.1.6 Reordered Collating Sequence

You can create a new collating sequence by reordering an existing expression through the use of parentheses.

In the following example, NCS applies the *conversion_function* and compares the input strings using *collating_sequence_b* as a comparison basis in the first pass. If the strings do not compare, NCS compares the strings using *collating_sequence_a* as a basis during a second pass.

old_collating_sequence = collating_sequence_a + collating_sequence_b * conversion_function,

If you want to do the comparisons in a single pass using the sum of the collating sequences, reorder the operation using parentheses, as shown in the following example:

new_collating_sequence = (collating_sequence_a + collating_sequence_b) * conversion_function,

1.5.2 Collating Strings with Pad Characters

In some instances, you may need to collate strings padded with one or more pad characters. Typically, a string may be padded with the ASCII SPACE character, but the pad character can be defined as any character. In order to avoid ambiguity when collating padded character strings, specify the collating value of the pad character in your collating sequence definition. For example, if you want to specify the collating weight of a pad character in a collating sequence, you might use the following statement:

```
NATIVE_SPACEPAD = CS(CS=_NATIVE, MOD = ("" = %X20));
```

This states that the shorter string should be treated as if it were padded with spaces for collating purposes. For example, the string *ABC<SP><SP>* would have the same collating weight as the string *ABC*.

1.5.3 Conversion Function Expressions

You can define conversion functions using any combination of the following types of expression:

- The name of an existing conversion function
- A sequential series of conversions
- A modified conversion function
- An inverted conversion function
- A reordered conversion function

1.5.3.1 Definition Name

You can create a new conversion function from an existing conversion function by equating the new function to the name of the existing function, using the following format:

new_conversion_function = name_of_existing_conversion_function;

Following is an example of this type of expression:

```
MY_CONVERSION = MULTI_TO_LOWER;
```

1.5.3.2 Sequential Series of Conversions

You can create a new conversion function by expressing it as the result of a sequential series of conversions, using the following format:

new_conversion_function = conversion_1 * conversion_2 * conversion_3;

NCS applies a sequential series of conversions in a single pass beginning with the rightmost conversion and continuing right to left. In the preceding format, *conversion_3* is applied before *conversion_2*, and *conversion_2* before *conversion_1*.

Following is an example of a conversion function derived through a sequential series of conversions:

```
MY_CONVERSION = MCS_NODIACRITICALS * MCS_LOWER;
```

1.5.3.3 Modified Conversion Function

You can create a conversion function from another conversion function appropriately modified through the use of keyword clauses, in the following format:

new_conversion_function = CF(keyword_clause,keyword_clause, ...)

The expression begins with a definition identifier (CF) followed by several keyword clauses enclosed in parentheses and separated by commas. The first keyword clause identifies the conversion function that serves as the basis for the new conversion function. The second keyword clause lists the appropriate modifications.

Following is an example of an expression that uses keyword clauses to create a new conversion function:

```
EDT1 = CF(CF=EDT,MODIFICATIONS=(
        "^" = "^=",
        "<" = "<="));
```

In this example, the new conversion function (EDT1) is a modified representation of an existing conversion function (EDT).

1.5.3.4 Inverted Conversion Function

You can create a new conversion function by logically inverting an existing conversion function, using the following format:

new_conversion_function = INVERSE(CF_expression)

Typically, you use inversion when you want to restore a converted string to its original form. For example, assume you have the following conversion function:

```
EDT1 = CF(
    CF = _IDENTITY, MODI = (
   .
   .
   .
        %X01 = "^A",
        %X02 = "^B",
        %X03 = "^C",
        %X04 = "^D",
        %X05 = "^E",
        %X06 = "^F",
        %X07 = "^G",
   .
   .
   .
          "" = %X00));
```

Now assume you want to convert the output string back to the original input string. To do this, you can use the following conversion function:

```
UNEDT1 = INVERSE(EDT1)
```

1.5.3.5 Reordered Conversion Function

You can effectively reorder conversion functions using parentheses. For example, the following format results in a two-pass comparison where *collating_sequence_a* is used as the basis for comparing the input strings during the first pass:

```
my_cs = collating_sequence_a + collating_sequence_b * conversion_c * conversion_d,
```

If the strings do not compare, *conversion_d* and *conversion_c* are applied to the input strings and then the strings are compared on the second pass based on *collating_sequence_b*.

If you want a one-pass comparison that converts input strings and then compares them using the composite effects of *collating_sequence_a* and *collating_sequence_b*, add parentheses, as follows:

```
my_cs = (collating_sequence_a + collating_sequence_b) * conversion_c * conversion_d,
```

1.6 Keyword Clauses

Keyword clauses are the basic elements of a collating sequence definition. They are used to establish the basis for each collating sequence, to explicitly define modifications where applicable, and to optionally assign a version number to the collating sequence.

As described previously, a definition expression can take the form of a definition identifier (CS or CF) followed by a set of one or more keyword clauses enclosed in parentheses and separated by commas, as follows:

```
new_expression = CS(keyword_clause,keyword_clause, ...)
```

All keyword clauses begin with a keyword followed by the equal sign and a value:

```
keyword = value
```

The value varies with the keyword, as shown in the next two sections.

Note that you can abbreviate any keyword as long as the abbreviated form is not ambiguous.

1.6.1 Collating Sequence Keyword Clauses

There are four types of keyword clauses you can use to build a collating sequence expression:

- CS
- SEQUENCE
- IDENT
- MODIFICATIONS

Each of these is described in the following subsections.

1.6.1.1 CS Keyword Clause

The CS keyword clause equates the keyword CS to a collating sequence expression that can take any one of the forms described in Section 1.5. The CS keyword clause typically establishes the collating basis for a collating sequence unless you use the SEQUENCE keyword clause. The CS keyword clause uses the following format:

```
CS = CS_expression
```

The following example illustrates the use of the CS keyword clause:

```
MULTINATIONAL_2 = CS(      ! Define temporary for second pass.
        IDENT = "V1.0",
        CS = _NATIVE * MULTI_TO_UPPER,
        MODIFICATIONS=(
            %XC6 > "Z",         ! AE diphthong
            %XD8 > %XC6,        ! O with slash
   .
   .
   .
            %XE0-%XEF = %XC0-%XCF,
            %XF1-%XFD = %XD1-%XDD ) );
```

1.6.1.2 SEQUENCE Keyword Clause The SEQUENCE keyword clause establishes the collating basis for a collating sequence when the CS keyword clause is not appropriate. Typically, you use this keyword clause when you do not have an existing collating sequence that is appropriate for your application and thus must fabricate one.

The following example illustrates the use of the SEQUENCE keyword clause in fabricating the Dutch collating sequence:

```
DUTCH = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
           %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Í"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6573−GE

In this example, the first SEQUENCE keyword clause establishes the collating sequence for the entire character set from %X00 through %XFF during the first pass. The associated MODIFICATIONS keyword clause effectively masks the lowercase characters and diacritical characters by equating each lowercase alphabetical character to its related uppercase alphabetical character, and by equating each diacritical character to its related nondiacritical, uppercase character.

During the second pass, the second SEQUENCE keyword clause establishes the collating sequence for the diacritical characters. The associated MODIFICATIONS keyword clause masks all lowercase characters, both diacritical and nondiacritical, by equating them to their related uppercase character.

1.6.1.3 IDENT Keyword Clause

The IDENT keyword clause is a special-purpose keyword clause that permits you to assign a version number to the collating sequence. The following example illustrates using the IDENT keyword to assign Version 1.0 to the Spanish collating sequence:

```
SPANISH = CS(
        IDENT = "V1.0",
        CS = MULTINATIONAL,
   .
   .
   .
                           ) );
```

1.6.1.4 MODIFICATIONS Keyword Clause

The MODIFICATIONS keyword clause is typically used with the built-in collating sequence _NATIVE to derive a modified version of the collating sequence.

In the following example, the new collating sequence (MULTINATIONAL_1) applies two conversion functions (MULTI_TO_NODIACRITICALS and MULTI_TO_UPPER) to the input strings. Then the collating sequence uses a modified version of the built-in collating sequence, _NATIVE, as the basis for comparing the strings.

The first group of modifications assigns collating weights to the listed special characters. For example, the uppercase N tilde character Ñ is assigned the same collating weight as the alphabetic character (N). The second group of modifications gives each lowercase character the same collating weight as the associated uppercase character.

```
MULTINATIONAL_1 = CS(
        CS = _NATIVE * ( MULTI_TO_NODIACRITICALS * MULTI_TO_UPPER ),
        MODIFICATIONS=(
            !
            ! Special characters
            !
            %XC6 > "Z",         ! AE diphthong
            %XD8 > %XC6,        ! O with slash
            %XC5 > %XD8,        ! A with ring
            %XD1 > "N",         ! N tilde
            %XDF = "SS",        ! S sharp
            %XD7 = "OE",        ! OE ligature
            !
            ! Define lowercase to collate the same as uppercase
            !
            %XE0-%XEF = %XC0-%XCF,
            %XF1-%XFD = %XD1-%XDD ) );
```

1.6.2 Conversion Function Keyword Clauses

There are three types of keyword clauses you can use to build a conversion function:

- CF
- IDENT
- MODIFICATIONS

Each of these is described in the following subsections.

1.6.2.1 CF Keyword Clause

The CF keyword clause equates the keyword CF to a conversion function expression, which can take any one of the forms described in Section 1.5. The general form for the CF keyword clause follows:

CF = CF_expression

The following example illustrates the use of the CF keyword clause:

```
MULTI_TO_NODIACRITICALS = CF(
      IDENT = "V1.0",
      CF = _IDENTITY,
        MODIFICATIONS=(
            %XC0-%XC5 = "A",    ! Various forms of "A" assigned same weight
   .
   .
   .
            %XF9-%XFC = "u",    ! Various forms of "u" assigned same weight
            %XFD = "y" ) );     ! Y umlaut assigned weight of "y"
```

See Section 1.4 for details on the use of the _IDENTITY built-in definition.

1.6.2.2 IDENT Keyword Clause

The IDENT keyword clause is a special-purpose keyword clause that permits you to assign a version number to the conversion function. The following example illustrates the use of the IDENT keyword clause to assign Version 1.0 to a conversion function:

```
MULTI_TO_LOWER = CF(
        IDENT = "V1.0",
        CF = _IDENTITY,
        MODIFICATIONS=(
            %X41-%X5A = %X61-%X7A,      ! Characters A-Z
    .
    .
    .
            %XD1-%XDD = %XF1-%XFD ) );  ! Various characters
```

1.6.2.3 MODIFICATIONS Keyword Clause

The MODIFICATIONS keyword clause is used in conjunction with another conversion function, typically the built-in conversion function _IDENTITY, to derive a modified version of the conversion function. The MODIFICATIONS keyword clause causes NCS to modify the specified conversion function, making changes in the specified order.

The following example illustrates the use of the MODIFICATIONS keyword clause in a conversion function:

```
MULTI_TO_UPPER = CF(
        IDENT = "V1.0",
        CF = INVERSE(MULTI_TO_LOWER),
        MODIFICATIONS=(%XDF = "SS",  %XD7 = "OE");)
```

In this example, the MODIFICATIONS keyword clause masks the lowercase sharp s (ß) by equating it to the character pair SS and the uppercase OE ligature × by equating it to the character pair OE.

1.6.3 MODIFICATIONS Keyword Clause Syntax

This section expands on the syntax requirements for developing MODIFICATIONS keyword clauses in an NCS expression.

The conversion function MODIFICATIONS keyword clause uses the following format:

left_string = right_string

NCS treats this as *left string converts to right string.* The following example illustrates the use of a MODIFICATIONS keyword clause in a conversion function:

```
.
.
.
    CF = _IDENTITY,
    MODIFICATIONS=(
        %X41-%X5A = %X61-%X7A,
.
.
.
```

In this example, NCS treats the MODIFICATIONS keyword clause as *uppercase characters convert to lowercase characters*, where the source string is a range of hexadecimal numbers representing uppercase alphabetic characters and the destination string is a range of hexadecimal numbers representing lowercase alphabetic characters.

Note that the equal sign (=) is always the sign of operation for conversion function MODIFICATIONS keyword clauses.

The various formats for the collating sequence MODIFICATIONS keyword clause are listed in Table NCS–2.

Table NCS–2 Formats for Collating Sequence MODIFICATIONS Keyword Clauses

Format	Interpretation
string = string	left string collates as right string
range = string	left range collates as right string
range = range	left range collates as right range
string > string	left string collates just greater than right string
range > string	left range collates just greater than right string
range > range	left range collates just greater than right range
string < string	left string collates just less than right string
range < string	left range collates just less than right string
range < range	left range collates just less than right range

You can specify a range of values in a keyword clause, using the following format:

string1 - string2

The following example shows how to assign the left string the same collating value as the right string:

```
.
.
.
    CS = _NATIVE * ( MULTI_TO_NODIACRITICALS * MULTI_TO_UPPER ),
    MODIFICATIONS=(
        %XDF = "SS",
.
.
.
```

The MODIFICATIONS keyword clause in this example states that the hexadecimal value for *S sharp* (%XDF) has the same collating value as *SS*.

The next example shows various source strings being assigned collating values that are relatively greater than, less than, and equal to the destination strings:

```
   .
   .
   .
MODIFICATIONS=(
            %XDF > "S",         ! S sharp
            %XD7 < "O",         ! OE ligature
            %XF7 = %XD7 ) );    ! oe ligature
   .
   .
   .
```

Here the hexadecimal value for *S sharp* collates just greater than the letter *S*, the hexadecimal value for the uppercase × ligature collates just less than the letter *O*, and the hexadecimal value for the lowercase ÷ ligature collates equal to uppercase ×.

You can also assign a range of strings the same collating value as a single string. For example, if you want to assign all of the lowercase alphabetical characters the same collating weight as a null character that typically has the lowest collating value in a set, you would state it as follows:

```
   .
   .
   .
CS = _NATIVE
   MODIFICATIONS=(
            %X61-%X7A = %X00,
   .
   .
   .
```

In building a definition file, you can compose a definition that is not compatible with NCS because the function is not well defined or because of some restriction NCS may impose on the function. In analyzing a definition, use the NCS /EXTRACT command to retrieve the function (see Part II). The NCS/EXTRACT command restructures the definition in its most basic form. The resulting definition may not be as efficient as the original definition but it is generally more logically structured and easier to read.

Part II

NCS Command and Command Qualifiers

This part of the document describes the NCS command and its qualifiers.

NCS Usage Summary

The NCS command invokes the National Character Set (NCS) Utility, which performs NCS library functions specified by NCS qualifiers. By default, NCS tries to replace the definition modules in the default NCS library with the definitions in the specified input file. All other NCS library functions require explicit command qualifiers.

PARAMETER

NCS [input-filespec]

Parameter

[input-filespec,...]
Where applicable, specifies the name of one or more input files containing definitions that NCS is to use to perform the action specified by the command qualifier. By default, NCS attempts to replace modules in the default NCS library with definitions in the input file.

You must specify an input file when you want to either replace or insert a module in the specified NCS library. Specifying an input file is optional when you want to create a new library.

If you specify several input files, you must separate them with commas (,). The default file type for input files is NCS.

usage summary

Invoke the NCS Utility by entering the DCL command NCS. The NCS Utility exits when the specified command operations are completed. If you use the /LIST qualifier, you can direct output to a specified file. If you do not specify a file, the output is directed to SYS$OUTPUT.

NCS Command Qualifiers

You may use one qualifier, no qualifiers, or several qualifiers with the NCS command, depending on the library functions you want to perform. Most of the qualifiers are compatible with one another and some require that you enter a related qualifier to do a specific task.

Table NCS–3 lists qualifiers that require a related qualifier and qualifiers that are incompatible.

Table NCS–3 NCS Command Qualifier Relationships

Qualifier	Related Qualifiers	Incompatible Qualifiers
/BEFORE[2]	/LIST	
/COMPRESS	/OUTPUT, /LIBRARY	/CREATE, /EXTRACT
/CREATE[1]	/LIBRARY	/COMPRESS, /EXTRACT
/DATA	/COMPRESS, CREATE	/EXTRACT, /INSERT, /REPLACE
/DELETE	/LIBRARY	/CREATE, /EXTRACT
/EXTRACT	/LIBRARY, /MACRO, /OUTPUT	/COMPRESS, /CREATE, /DELETE
/FORMAT[3]	/MACRO	
/FULL[2]	/LIST	
/HISTORY[2]	/LIST	
/INSERT	/LIBRARY	/EXTRACT
/LIBRARY	COMPRESS, /CREATE, /DELETE, /EXTRACT, /INSERT, /LIST, /REPLACE	
/LIST	/BEFORE, /FULL, /HISTORY, /LIBRARY, /ONLY, /SINCE,	/EXTRACT
/LOG	/CREATE, /DELETE, /EXTRACT, /INSERT, /REPLACE	
/ONLY[2]	/LIST	
/OUTPUT	/EXTRACT	/DELETE
/REPLACE	/LIBRARY	/EXTRACT
/SINCE[2]	/LIST	

[1]The /CREATE, /INSERT, and /REPLACE qualifiers are compatible but /CREATE takes precedence. The related qualifiers for /CREATE are applicable only if you enter one or more input files.

[2]This qualifier is meaningful only when used with the /LIST qualifier.

[3]This qualifier is meaningful only when used with the /MACRO qualifier.

/BEFORE

Lists only the modules inserted in the library before the specified time.

Format

/BEFORE[= time]

Qualifier Value

time
The time (and date, where applicable) that NCS uses as the reference cutoff point for accumulating the appropriate list of modules.

Description

When you use the /LIST qualifier, NCS lists all the definitions in the NCS library by default. However, you can use the /BEFORE qualifier with the /LIST qualifier to list only definitions created before a specified time (and date, where applicable). You may specify an absolute time or a combination of absolute time and delta time. For details on specifying times, see the *OpenVMS DCL Dictionary*.

If you use the /BEFORE qualifier without specifying the time, the output list includes all definitions in the library created before today.

Examples

1.
```
$ NCS/LIST/BEFORE=31-DEC-1988:13:30
```

   This command lists all definition modules inserted in the default NCS library before 1:30 p.m. on December 31, 1988.

2.
```
$ NCS/LIST/BEFORE=31-DEC-1988:09:00/LIBRARY=USERDISK:[DOE]LIB.NLB
```

   This command lists all definition modules inserted in user DOE's local NCS library, LIB.NLB, before 9:00 a.m. on December 31, 1988.

/COMPRESS

Recovers disk space previously occupied by deleted definition modules.

Format

/COMPRESS[=(option[,...])]

Qualifier Value

option
A set of options that permits you to change the size or format of the specified library, overriding the values assigned to the library when it was created. See the listing under DESCRIPTION.

Description

The /COMPRESS qualifier effectively recovers disk space previously occupied by modules deleted from an NCS library by creating a more efficiently organized output library file. If you do not explicitly specify a destination library, NCS creates a new compressed version of the default NCS library. When you compress an NCS library, you can override various NCS default values for the size and format of the library, using the following options:

BLOCKS:n	Specifies the number of 512-byte blocks to be allocated for the library. By default, NCS allocates 100 blocks for a new library.
HISTORY:n	Specifies the maximum number of library update history records that the library may maintain. By default, NCS sets the number at 20.
KEYSIZE:n	Changes the maximum length of definition module names.
MODULES:n	Specifies the maximum number of modules in the NCS library.

Examples

1.
```
$ NCS/COMPRESS=(BLOCKS:200)/LIBRARY=USERDISK:[DOE]LIB.NLB
```
   This command compresses user DOE’s local NCS library and simultaneously allocates two hundred 512-byte blocks for it.

2.
```
$ NCS/COMPRESS=(KEYSIZE:35,MODULES:40)/LIBRARY=USERDISK:[DOE]LIB.NLB
```
   This command compresses user DOE’s local NCS library and specifies a maximum key size of 35 bytes together with a maximum of 40 definition modules.

/CREATE

Creates an NCS library.

Format

/CREATE[=(option[,...])]

Qualifier Value

option
A set of options that permits you to override the system defaults for the size and format of the newly created NCS library. See the listing under DESCRIPTION.

Description

Use the /CREATE qualifier to create an NCS library. Note that you must use the /LIBRARY qualifier and explicitly specify the device and directory when you create a new NCS library (See examples). If you do not specify a device and directory in the command line, NCS creates the library in SYS$LIBRARY by default. If you do not use the /LIBRARY qualifier to specify a new library, NCS creates a new version of the default NCS library.

To populate the new library from an existing definition file, enter as the command parameter the specification for the input file containing the definitions.

When you create an NCS library, you can override various NCS default values for the size and format of the library, using the following options:

BLOCKS:n	Specifies the number of 512-byte blocks to be allocated for the library. By default, NCS allocates 100 blocks for a new library.
HISTORY:n	Specifies the maximum number of library update history records that the library may maintain. By default, NCS sets the number to 20.
KEYSIZE:n	Changes the maximum length of definition module names.
MODULES:n	Specifies the maximum number of modules in the NCS library.

Examples

1.
```
$ NCS/CREATE=(BLOCKS:200,KEYSIZE:24)/LIBRARY=DISK1:[DOE]LIB.NLB
```
   This command creates a file named LIB.NLB in the directory DOE, while simultaneously allocating the file 200 blocks and setting the maximum module name length at 24 characters.

2.
```
$ NCS/CREATE=MODULES:40/LIBRARY=USERDISK:[DOE]ABC.NLB MY.NCS
```
   This command creates an NCS library in directory DOE named ABC.NLB. The command limits the library to 40 modules and populates it with definitions from input file MY.NCS.

/DATA

Improves disk space efficiency.

Format

$$\text{/DATA=}\left\{\begin{array}{l}\text{REDUCE}\\\text{EXPAND}\end{array}\right\}$$

Qualifier Values

REDUCE
Stores definitions in a data-reduced format.

EXPAND
Stores definitions in a data-expanded format.

Description

The /DATA qualifier gives you the option of specifying how you want definitions stored in the NCS library, in data-reduced format or data-expanded format. Note that you *must* specify a value, either REDUCE or EXPAND, with this qualifier; there is no default value. Note, too, that NCS does an implicit compression (recovers unused space from previously deleted files) on the specified library, whether you are converting the library to data-reduced form or to data-expanded form.

If the specified NCS library is in standard (nonreduced) form, use the REDUCE option to create a new reduced version of the library. If you do not use the /LIBRARY qualifier to specify a library, NCS creates a new reduced version of the default NCS library. Note that access to libraries in data-reduced format is generally slower than libraries in data-expanded format.

If the specified NCS library is in reduced form, use the EXPAND option to create a new expanded (standard) version of the library. If you do not use the /LIBRARY qualifier to specify a library, NCS creates a new expanded version of the default NCS library. Note that access to libraries in data-expanded (standard) format is generally faster than libraries in data-reduced format.

Examples

1.
```
$ NCS/DATA=REDUCE
```
This command maximizes the space efficiency of the default NCS library.

2.
```
$ NCS/DATA=EXPAND SYS$LIBRARY:NCS$LIBRARY
```
This command reduces the access time to and space efficiency of the default NCS library.

/DELETE

Deletes one or more definition modules from the NCS library.

Format

/DELETE=module[,...]

Qualifier Value

module
The name of the module to be deleted.

Description

The /DELETE qualifier deletes the specified definition module or modules from an NCS library. If you specify several modules, separate the definition names with commas (,) and enclose the list in parentheses. You may use the standard VMS wildcard characters to specify the modules to be deleted.

If you use this qualifier with the /LIST qualifier, NCS deletes the module before it lists the contents of the library. Therefore, the deleted definitions do not appear in the output listing.

Examples

1.
```
$ NCS/DELETE=CH*
```

   This command deletes all NCS library modules that begin with the letters *CH*.

2.
```
$ NCS/DELETE=(LOWER_TO_UPPER,CHANGECASE)/LIST
```

   This command deletes the definition CHANGECASE from the default NCS library and then lists the remaining definitions.

/EXTRACT

Extracts definitions from an NCS library.

Format

```
/EXTRACT=module[,...]  { /OUTPUT= filespec }
                       { /MACRO= filespec  }
```

Qualifier Value

module
The name of the module to be extracted.

Description

The /EXTRACT qualifier is used to retrieve one or more definition modules from an NCS library. Note that you *must* use either the /OUTPUT qualifier or the /MACRO qualifier with the /EXTRACT qualifier to specify a destination file for the extracted definition.

Use the /EXTRACT qualifier with the /MACRO qualifier to extract one or more definitions that you want to include in a MACRO-32 program file. When you use the /MACRO qualifier with the /EXTRACT qualifier, the default output file type is MAR.

Use the /EXTRACT qualifier with the /OUTPUT qualifier to extract one or more definitions that you want to include in an NCS definition file. When you use the /OUTPUT qualifier with the /EXTRACT qualifier, the default output file type is NCS.

If you want to extract several modules, separate the module names with commas (,) and enclose the list in parentheses.

You may use any of the standard VMS wildcard characters to specify the modules to be extracted.

Examples

1.
```
$ NCS/EXTRACT=CHANGECASE/MACRO=MY
```

   This command extracts the definition CHANGECASE from the default NCS library and converts it to MACRO format before storing it in the file MY.MAR.

2.
```
$ NCS/EXTRACT=(CHANGECASE,UPPER_TO_LOWER)/OUTPUT=MY
```

   This command extracts two definitions from the default NCS library and stores them in the definition file MY.NCS.

/FORMAT

Specifies the MACRO format appropriate to your program.

Format

$$\text{/FORMAT=} \left\{ \begin{array}{l} \text{NCS} \\ \text{256} \end{array} \right\}$$

Qualifier Values

NCS
MACRO-32 format for NCS routines.

256
MACRO-32 format for the Run-Time Library routine LIB$MOVTC.

Description

Use the /FORMAT qualifier with the /MACRO qualifier to specify the appropriate MACRO-32 file format. You may select one of two format options, either NCS (the default) or 256. If you select the NCS format, NCS formats the collating sequence and string conversion tables, or both, for use by the NCS routines. If you select the 256 format, NCS formats the collating sequence and string conversion tables, or both, as 256-byte tables that can be used by the Run-Time Library routine LIB$MOVTC. For more information, see the *OpenVMS RTL Library (LIB$) Manual*.

Examples

1.
```
$ NCS/EXTRACT=UP_DOWN/MACRO=MY/FORMAT=256
```
This command extracts the definition UP_DOWN from the default NCS library and converts it to the 256-byte table format before including it in the file MY.MAR.

2.
```
$ NCS/EXTRACT=(ABCDE,XYZ)/MACRO=MY
```
This command extracts two definitions from the default NCS library and converts them, by default, to NCS format before including them in the file MY.MAR.

/FULL

Provides a complete listing of an NCS library.

Format

/FULL

Parameters

None.

Description

Use the /FULL qualifier with the /LIST qualifier to obtain an NCS library listing that includes the date and time each module was inserted into the library. The output has the following format:

module inserted dd-mmm-yyyy hh:mm:ss

Example

```
$ NCS/LIST/FULL
```

This command lists the modules in the default NCS library, together with the date and time that each module was inserted into the library.

/HISTORY

Provides the update history record headers for the NCS library.

Format

/HISTORY

Parameters

None.

Description

You use the /HISTORY qualifier with the /LIST qualifier to obtain a list of the update history record headers for the specified NCS library, in the following format:

username operation n modules on dd-mmm-yyyy hh:mm:ss

The *operation* may be a replacement, insertion, or deletion of definition modules.

If you specify the /FULL qualifier with the /HISTORY and /LIST qualifiers, NCS lists the history record headers for each update, together with a list of the definition modules affected by each update.

Examples

1. `$ NCS/LIST/HISTORY`

   If you had previously deleted two definition modules from the NCS library, this command would produce a listing that includes the following line:

   ```
   JONES       deleted   2 modules on 31-DEC-1988 16:26:36
   ```

2. `$ NCS/LIST/HISTORY/FULL`

   Making the same assumptions as in the previous example, this command would produce a listing that includes the names of the deleted modules:

   ```
   JONES       deleted   2 modules on 31-DEC-1988 16:26:36
   CHANGECASE
   UPCASE
   ```

/INSERT

Adds one or more definition modules to an NCS library.

Format

/INSERT   filename

Parameters

None.

Description

Use the /INSERT qualifier to add one or more definition modules to an NCS library from an input file. If the input file contains more than one definition, NCS creates a separate entry in the library for each.

Before NCS inserts a definition into an existing NCS library, it verifies that the library does not already contain a definition module having the same name. If NCS finds a definition module with the same name, it does *not* add the new definition module to the library, but it does provide an appropriate error message.

Examples

1.
```
$ NCS/INSERT MY_DEFS
```
   This command directs NCS to insert each of the definitions from input file MY_DEFS.NCS into the NCS library.

2.
```
$ NCS/INSERT/LIST/HISTORY/FULL MY_DEFS
```
   This command directs NCS to insert the definitions from input file MY_DEFS.NCS into the NCS library; then NCS lists the history of each definition module in the library, including those inserted by this command.

/LIBRARY

Specifies an alternate NCS library. The default NCS library is SYS$LIBRARY:NCS$LIBRARY.

Format

/LIBRARY= filespec

Qualifier Value

filespec
The alternate NCS library file specification.

Description

The /LIBRARY qualifier allows you to specify an NCS library other than the default NCS library (SYS$LIBRARY:NCS$LIBRARY). Note if you do not include a device and directory when you specify the alternate library, NCS defaults to SYS$LIBRARY. The default file type is NLB.

Examples

1.
```
$ NCS/INSERT/LIBRARY=USERDISK:[DOE]MY_LIB MY_DEFS
```
This command directs NCS to insert definitions from the file MY_DEFS.NCS into the NCS library MY_LIB.NLB.

2.
```
$ NCS/DELETE=UPCASE/LIBRARY=GEN_NCS
```
This command directs NCS to delete the module UPCASE from the library file SYS$LIBRARY:GEN_NCS.NLB.

/LIST

Lists the contents of an NCS library.

Format

/LIST[= filespec]

/NOLIST

Qualifier Value

filespec
The destination file specification for the list output.

Description

The /LIST qualifier allows you to obtain a listing of an NCS library. The optional file specification parameter allows you to store the listing in a file. The default file type is LIS. If you omit the file specification, NCS directs the listing to SYS$OUTPUT.

You are not permitted to use wildcard characters when specifying the destination file.

Note that when you use the /LIST qualifier in conjunction with qualifiers (such as the /DELETE qualifier) that modify the contents of the NCS library, NCS creates the listing *after* the modifications are made. For example, if you delete the definition module UPCASE and simultaneously request a listing, the listing does not include the UPCASE module.

Listings can provide various types of information, depending on the qualifiers you use with /LIST. Each listing, however, contains at least the following information about the library:

```
Directory of NCS library library-filespec on dd-mmm-yyyy hh:mm:ss
Creation date: dd-mmm-yyyy hh:mm:ss Creator: VAX librarian Vnn-nn
Revision date: dd-mmm-yyyy hh:mm:ss Library format: n.n
Number of modules: nnn Max. key length: nnn
Other entries: nnn Preallocated index blocks: nnn
Recoverable deleted blocks: nnn Total index blocks used: nnn
Max. update history records: nnn Update history records: nnn
```

Examples

1. `$ NCS/LIST`

   This command outputs a listing of the definitions in the default NCS library to the SYS$OUTPUT device.

2. `$ NCS/LIST=DEFLIST/FULL`

   This command outputs a listing of the definitions in the default NCS library to a file named DEFLIST.LIS. The listing includes the date each definition was inserted in the NCS library.

/LOG

Determines whether or not NCS verifies library operations.

Format

/LOG

/NOLOG

Parameters

None.

Description

Use the /LOG or /NOLOG qualifier to specify whether or not you want NCS to confirm the result of a specified operation (such as a replacement, insertion, or deletion). By default, NCS does not confirm operations.

Example

```
$ NCS/DELETE=(TJL,RRR)/LOG
```

This command directs NCS to delete the definitions TJL and RRR in the default NCS library. After deleting the definitions, NCS issues the following messages:

```
   %NCS-S-DELETED, module TJL deleted
   %NCS-S-DELETED, module RRR deleted
```

/MACRO

Specifies that the extracted definition table is coded in MACRO-32.

Format

/MACRO= filespec

Qualifier Value

filespec
The file specification for the destination MACRO file.

Description

Use the /MACRO qualifier with the /EXTRACT qualifier if you want to retrieve a definition module from the NCS library, convert the definition to VAX MACRO format, and then output the VAX MACRO-formatted definition to the specified destination file. Note that each collating sequence and conversion function bears a global label that you can use to pass the address of the definition to an NCS routine.

You must include the destination file specification with the /MACRO qualifier. You cannot use wildcard characters to specify the destination file. The default value for MACRO-32 files generated by the /MACRO qualifier is MAR.

Note that you can further define a specific MACRO-32 format with the /FORMAT qualifier, as shown in the second example.

Examples

1. `$ NCS/EXTRACT=DOWNCASE/MACRO=MY_MACRO`

   This command directs NCS to extract the definition DOWNCASE from the default NCS library, convert the definition to VAX MACRO format, and then output it to file MY_MACRO.MAR. The following example shows an NCS definition module converted to VAX MACRO:

```
.PSECT  NCS$RO_DATA NOVEC,NOWRT,RD,NOEXE,SHR,LCL,REL,CON,PIC,LONG DOWNCASE::

.LONG   ^X00000158,^X00000000,^X00000000,^X00000000
.LONG   ^X00000000,^X776F6408,^X7361636E,^X00000065
.LONG   ^X00000000,^X00000000,^X00000000,^X00000000
.LONG   ^X00000000,^X00000000,^X00000000,^X00000000
.LONG   ^X00000005,^X00000158,^X00000000,^X00000000
.LONG   ^X00000000
.LONG   ^X00000000
.LONG   ^X03020100,^X07060504,^X0B0A0908,^X0F0E0D0C
.LONG   ^X13121110,^X17161514,^X1B1A1918,^X1F1E1D1C
  .          .          .            .           .
  .          .          .            .           .
  .          .          .            .           .
.LONG   ^X23222120,^X27262524,^X2B2A2928,^X2F2E2D2C
.LONG   ^X33323130,^X37363534,^X3B3A3938,^X3F3E3D3C
.LONG   ^X63626140,^X67666564,^X6B6A6968,^X6F6E6D6C
.LONG   ^X73727170,^X77767574,^X5B7A7978,^X5F5E5D5C
.LONG   ^X63626160,^X67666564,^X6B6A6968,^X6F6E6D6C
.LONG   ^X73727170,^X77767574,^X7B7A7978,^X7F7E7D7C
.LONG   ^XF3F2F1F0,^XF7F6F5F4,^XFBFAF9F8,^XFFFEFDFC
.END
```

2. `$ NCS/EXTRACT=DOWNCASE/MACRO=MY_MACRO/FORMAT=256`

   This command directs NCS to extract the DOWNCASE definition from the default NCS library, as in the first example. However, this command explicitly specifies that NCS convert the definition to the MACRO-32 format for use with the Run-Time Library routine LIB$MOVTC instead of the default NCS format. The following example illustrates the converted definition:

```
.PSECT  NCS$RO_DATA NOVEC,NOWRT,RD,NOEXE,SHR,LCL,REL,CON,PIC,LONG DOWNCASE::
.LONG   ^X03020100,^X07060504,^X0B0A0908,^X0F0E0D0C
.LONG   ^X13121110,^X17161514,^X1B1A1918,^X1F1E1D1C
.LONG   ^X23222120,^X27262524,^X2B2A2928,^X2F2E2D2C
.LONG   ^X33323130,^X37363534,^X3B3A3938,^X3F3E3D3C
.LONG   ^X63626140,^X67666564,^X6B6A6968,^X6F6E6D6C
.LONG   ^X73727170,^X77767574,^X5B7A7978,^X5F5E5D5C
.LONG   ^X63626160,^X67666564,^X6B6A6968,^X6F6E6D6C
.LONG   ^X73727170,^X77767574,^X7B7A7978,^X7F7E7D7C
.LONG   ^X83828180,^X87868584,^X8B8A8988,^X8F8E8D8C
.LONG   ^X93929190,^X97969594,^X9B9A9998,^X9F9E9D9C
.LONG   ^XA3A2A1A0,^XA7A6A5A4,^XABAAA9A8,^XAFAEADAC
.LONG   ^XB3B2B1B0,^XB7B6B5B4,^XBBBAB9B8,^XBFBEBDBC
.LONG   ^XE3E2E1E0,^XE7E6E5E4,^XEBEAE9E8,^XEFEEEDEC
.LONG   ^XF3F2F1D0,^XF7F6F5F4,^XFBFAF9F8,^XDFDEFDFC
.LONG   ^XE3E2E1E0,^XE7E6E5E4,^XEBEAE9E8,^XEFEEEDEC
.LONG   ^XF3F2F1F0,^XF7F6F5F4,^XFBFAF9F8,^XFFFEFDFC
.END
```

/ONLY

Limits the modules being listed.

Format

/ONLY=module[,...]

Qualifier Value

module
The module or modules to be listed.

Description

Use the /ONLY qualifier with the /LIST qualifier to specify which definition modules in the NCS library you want listed. If you specify more than one definition module, separate the module names with commas (,) and enclose the list in parentheses.

You may use wildcard characters to specify the definition module(s).

Example

```
$ NCS/LIST/ONLY=(M*,P*)
```

This command directs NCS to list only the definition modules that have names beginning with the letter M or the letter P.

/OUTPUT

Specifies the output definition file.

Format

/OUTPUT= filespec

Qualifier Value

filespec
The file specification for the destination file.

Description

You use the /OUTPUT qualifier with the /EXTRACT and /COMPRESS qualifiers to specify a destination file.

Use the /OUTPUT qualifier with the /EXTRACT qualifier to specify a file to store the definitions extracted from an NCS library. The default file type is NCS. Note that the new definition file is not the same as the definition file used to originally create the definition. NCS does not store the original definition file, only the resultant definition.

Use the /OUTPUT qualifier with the /COMPRESS qualifier to specify a destination library file for the compressed library. If you do not use the /OUTPUT qualifier with the /COMPRESS qualifier, NCS puts the compressed library in a new version of the default NCS library. The default file type is NLB.

Examples

1.
```
$ NCS/EXTRACT=(UPCASE,DOWNCASE)/OUTPUT=USER:[DOE]NEW_DEFS.NCS
```
   This command directs NCS to extract two definitions (UPCASE and DOWNCASE) from the default NCS library and to store them in a file named NEW_DEF.NCS.

2.
```
$ NCS/COMPRESS/OUTPUT=USER:[DOE.NCS]NEW_LIB
```
   This command directs NCS to compress the default NCS library and to store the compressed version in an alternate NCS library specified as USER:[DOE.NCS]NEW_LIB.NLB.

/REPLACE

Replaces one or more definition modules in the default NCS library with modules from the specified input file.

Format

/REPLACE

Parameters

None.

Description

Use the /REPLACE qualifier when you want to replace one or more library modules with definitions from the specified input file. If a replacement module in the input file does not have a corresponding module in the NCS library, NCS inserts, rather than replaces, the new module in the library. If you do not explicitly specify a qualifier for the NCS command, the default is the /REPLACE qualifier.

Example

```
$ NCS MY_DEFS
```

This command directs NCS to replace each definition in the default NCS library with a corresponding definition from the input file, MY_DEFS.NCS.

/SINCE

Limits a library output listing to definitions inserted after the specified date and time. The most recent definitions are listed first, and are in alphabetical order.

Format

/SINCE[= time]

Qualifier Value

time
The time (and date, where applicable) that NCS uses as the reference point to begin accumulating the appropriate list of modules.

Description

Use the /SINCE qualifier with the /LIST qualifier to list only modules inserted into the library after the specified time (and date, where applicable). You can specify an absolute time or a combination of absolute and delta times. For details on specifying times, see the *OpenVMS DCL Dictionary*.

If you omit the /SINCE qualifier, NCS lists all the modules in the library unless you use a different limiting qualifier (/BEFORE or /ONLY). If you specify the /SINCE qualifier without a time or date, NCS lists only the modules inserted today.

Examples

1.
```
$ NCS/LIST/SINCE=31-DEC-1988:10:00
```
This command directs NCS to list only modules inserted in the default NCS library after 10 a.m. on December 31, 1988.

2.
```
$ NCS/LIST/SINCE
```
This command directs NCS to list only modules inserted in the default NCS library today.

A
National Character Set Definitions

This appendix lists the contents of the default NCS library (NCS$LIBRARY). The library includes 13 collating sequences and 24 conversion functions.

```
Danish = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Ä", "Ö", "Å", "["−"‘", "{"−"¿", %XD0,
           %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ã" = "A", "Æ" = "Ä", "Ç" = "C",
          "È"−"Ë" = "E", "Ì"−"Ï" = "I", "Ò"−"Õ" = "O", "Ø" = "Ö", "Ù"−"Û" = "U",
          "Ü"−"Ÿ" = "Y", "à"−"ã" = "A", "ä"−"å" = "Ä"−"Å", "æ" = "Ä", "ç" = "C",
          "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ", "ò"−"õ" = "O", "ö" = "Ö",
          "ø" = "Ö", "ù"−"û" = "U", "ü"−"ÿ" = "Y", "Œ" = "OE", "ß" = "SS",
          "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"‘", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6574−GE

```
Danish_NRC_to_Multi = CF(
    CF = _IDENTITY, MODE = (
        "@" = "Ä",
        "[" = "Æ",
        "\" = "Ø",
        "]" = "Å",
        "^" = "Ü",
        "‘" = "ä",
        "{" = "æ",
        "|" = "ø",
        "}" = "å",
        "~" = "ü",
        "" = %X00));
```

ZK−6575−GE

```
Dutch = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
           %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Ì"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6576−GE

```
EDT_VT2xx = CF(
    CF = _IDENTITY, MODI = (
%X00 − "^@",
%X01 = "^A",
%X02 = "^B",
%X03 = "^C",
%X04 = "^D",
%X05 = "^E",
%X06 = "^F",
%X07 = "^G",
%X08 = "^H",
%X0A = "<LF>",
%X0B = "<VT>",
%X0C = "<FF>",
%X0D = "<CR>",
%X0E = "^N",
%X0F = "^O",
%X10 = "^P",
%X11 = "^Q",
%X12 = "^R",
%X13 = "^S",
%X14 = "^T",
%X15 = "^U",
%X16 = "^V",
%X17 = "^W",
%X18 = "^X",
%X19 = "^Y",
%X1A = "^Z",
%X1B = "<ESC>",
%X1C = "^\",
%X1D = "^]",
%X1E = "^^",
%X1F = "^_",
%X7F = "<DEL>",
%X80 = "<X80>",
%X81 = "<X81>",
%X82 = "<X82>",
%X83 = "<X83>",
%X84 = "<IND>",
%X85 = "<NEL>",
%X86 = "<SSA>",
%X87 = "<ESA>",
%X88 = "<HTS>",
%X89 = "<HTJ>",
%X8A = "<VTS>",
%X8B = "<PLD>",
%X8C = "<PLU>",
%X8D = "<RI>",
%X8E = "<SS2>",
%X8F = "<SS3>",
%X90 = "<DCS>",
%X91 = "<PU1>",
%X92 = "<PU2>",
%X93 = "<STS>",
%X94 = "<CH>",
%X95 = "<MW>",
%X96 = "<SPA>",
%X97 = "<EPA>",
%X98 = "<X98>",
%X99 = "<X99>",
%X9A = "<X9A>",
%X9B = "<CSI>",
%X9C = "<ST>",
%X9D = "<OSC>",
%X9E = "<PM>",
%X9F = "<APC>",
%XA0 = "<XA0>",
%XA4 = "<XA4>",
%XA6 = "<XA6>",
%XAC = "<XAC>",
%XAD = "<XAD>",
%XAE = "<XAE>",
%XAF = "<XAF>",
%XB4 = "<XB4>",
%XB8 = "<XB8>",
%XBE = "<XBE>",
%XD0 = "<XD0>",
%XDE = "<XDE>",
%XF0 = "<XF0>",
%XFE = "<XFE>",
%XFF = "<XFF>",
"" = %X00));
```

ZK−6577−GE

```
English = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"‘", "{"−"¿", %XD0,
           %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Ì"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00)
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"‘", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6579−GE

```
 Finnish = CS(
     SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ"−"Ä", "Ö",  "["−"‘", "{"−"¿", %XD0,
            %XDE, %XF0, %XFE−%XFF),
     MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ã" = "A", "Ç" = "C", "È"−"Ë" = "E",
            "Ì"−"Ï" = "I", "Ò"−"Õ" = "O", "Ø" = "Ö", "Ù"−"Û" = "U", "Ü"−"Ÿ" = "Y",
            "à"−"ã" = "A", "ä"−"æ" = "Ä"−"Æ", "ç" = "C", "è"−"ë" = "E",
            "ì"−"ï" = "I", "ñ" = "Ñ", "ò"−"õ" = "O", "ö" = "Ö", "ø" = "Ö",
            "ù"−"û" = "U", "ü"−"ÿ" = "Y", "Œ" = "OE", "ß" = "SS", "œ" = "Œ",
            "" < %X00))
+ CS( SEQUENCE = (%X00−"A", "À"−"Ä", "B"−"C", "Ç", "D"−"E", "È"−"Ë",
      "F"−"I", "Ì"−"Ï", "J"−"N", "Ñ", "Œ", "O", "Ò"−"Ö", "P"−"R", "ß",
      "S"−"U", "Ù"−"Ü", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"‘", "{"−"¿",
      %XD0, %XDE, %XF0, %XFE−%XFF),
MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ý" = "Ñ"−"Ÿ"))
+ REVERSE(_NATIVE);
```

ZK−6580−GE

```
Finnish_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "[" = "Ä",
        "\" = "Ö",
        "]" = "Å",
        "^" = "Ü",
        "‘" = "é",
        "{" = "ä",
        "|" = "ö",
        "}" = "å",
        "~" = "ü",
        "" = %X00));
```

ZK−6581−GE

```
FrCan_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "@" = "à",
        "[" = "â",
        "\" = "ç",
        "]" = "ê",
        "^" = "î",
        "´" = "ô",
        "{" = "é",
        "|" = "ù",
        "}" = "è",
        "~" = "û",
        "" = %X00));
```

ZK−6582−GE

```
French = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
          %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Ì"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö""−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ý"))
    + REVERSE(_NATIVE);
```

ZK−6583−GE

```
French_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "#" = "£",
        "@" = "à",
        "[" = "°",
        "\" = "ç",
        "]" = "§",
        "{" = "é",
        "|" = "ù",
        "}" = "è",
        "^a" = "â",
        "^e" = "ê",
        "^i" = "î",
        "^o" = "ô",
        "^u" = "û",
        "~a" = "ä",
        "~e" = "ë",
        "~i" = "ï",
        "~o" = "ö",
        "~u" = "ü",
        "" = %X00));
```

ZK−6584−GE

```
German = CS(
    SEQUENCE = (%X00–"N", "Ñ", "O"–"Z", "Æ", "Ø", "Å", "["–"´", "{"–"¿", %XD0,
          %XDE, %XF0, %XFE–%XFF),
    MODIFICATIONS=("a"–"z" = "A"–"Z", "À"–"Ä" = "A", "Ç" = "C", "È"–"Ë" = "E",
          "Ì"–"Ï" = "I", "Ò"–"Ö" = "O", "Ù"–"Ü" = "U", "Ÿ" = "Y", "à"–"ä" = "A",
          "å"–"æ" = "Å"–"Æ", "ç" = "C", "è"–"ë" = "E", "ì"–"ï" = "I", "ñ" = "Ñ",
          "ò"–"ö" = "O", "ø" = "Ø", "ù"–"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00–"A", "Ä"–"À", "B"–"C", "Ç", "D"–"E", "Ë"–"È",
          "F"–"I", "Ï"–"Ì", "J"–"N", "Ñ", "Œ", "O", "Ö"–"Ò", "P"–"R", "ß",
          "S"–"U", "Ü"–"Ù", "V"–"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["–"´", "{"–"¿",
          %XD0, %XDE, %XF0, %XFE–%XFF),
    MODIFICATIONS=("a"–"z" = "A"–"Z", "à"–"ï" = "À"–"Ï", "ñ"–"ÿ" = "Ñ"–"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK–6585–GE

```
German_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "@" = "§",
        "[" = "Ä",
        "\" = "Ö",
        "]" = "Ü",
        "{" = "ä",
        "|" = "ö",
        "}" = "ü",
        "~" = "ß",
        "" = %X00));
```

ZK–6586–GE

```
Italian = CS(
    SEQUENCE = (%X00–"N", "Ñ", "O"–"Z", "Æ", "Ø", "Å", "["–"´", "{"–"¿", %XD0,
          %XDE, %XF0, %XFE–%XFF),
    MODIFICATIONS=("a"–"z" = "A"–"Z", "À"–"Ä" = "A", "Ç" = "C", "È"–"Ë" = "E",
          "Ì"–"Ï" = "I", "Ò"–"Ö" = "O", "Ù"–"Ü" = "U", "Ÿ" = "Y", "à"–"ä" = "A",
          "å"–"æ" = "Å"–"Æ", "ç" = "C", "è"–"ë" = "E", "ì"–"ï" = "I", "ñ" = "Ñ",
          "ò"–"ö" = "O", "ø" = "Ø", "ù"–"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00–"A", "Ä"–"À", "B"–"C", "Ç", "D"–"E", "Ë"–"È",
          "F"–"I", "Ï"–"Ì", "J"–"N", "Ñ", "Œ", "O", "Ö"–"Ò", "P"–"R", "ß",
          "S"–"U", "Ü"–"Ù", "V"–"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["–"´", "{"–"¿",
          %XD0, %XDE, %XF0, %XFE–%XFF),
    MODIFICATIONS=("a"–"z" = "A"–"Z", "à"–"ï" = "À"–"Ï", "ñ"–"ÿ" = "Ñ"–"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK–6587–GE

```
Italian_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "#" = "£",
        "@" = "§",
        "[" = "°",
        "\" = "ç",
        "]" = "é",
        "´" = "ù",
        "{" = "à",
        "|" = "ò",
        "}" = "è",
        "~" = "ì",
        "" = %X00));
```

ZK−6588−GE

```
Multinational = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
          %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Ì"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È",
          "F"−"I", "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß",
          "S"−"U", "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿",
          %XD0, %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "À"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ý"))
    + REVERSE(_NATIVE);
```

ZK−6589−GE

```
Multinational_1 = CS(
    SEQUENCE = (%X00−"N", "Ñ", "O"−"Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
          %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "À"−"Ä" = "A", "Ç" = "C", "È"−"Ë" = "E",
          "Ì"−"Ï" = "I", "Ò"−"Ö" = "O", "Ù"−"Ü" = "U", "Ÿ" = "Y", "à"−"ä" = "A",
          "å"−"æ" = "Å"−"Æ", "ç" = "C", "è"−"ë" = "E", "ì"−"ï" = "I", "ñ" = "Ñ",
          "ò"−"ö" = "O", "ø" = "Ø", "ù"−"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00));
```

ZK−6590−GE

```
Multinational_2 = CS (
    SEQUENCE = (%X00−"A", "Ä"−"À", "B"−"C", "Ç", "D"−"E", "Ë"−"È", "F"−"I",
          "Ï"−"Ì", "J"−"N", "Ñ", "Œ", "O", "Ö"−"Ò", "P"−"R", "ß", "S"−"U",
          "Ü"−"Ù", "V"−"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["−"´", "{"−"¿", %XD0,
          %XDE, %XF0, %XFE−%XFF),
    MODIFICATIONS=("a"−"z" = "A"−"Z", "à"−"ï" = "Á"−"Ï", "ñ"−"ÿ" = "Ñ"−"Ÿ",
          "" = %X00));
```

ZK−6591−GE

```
Multi_to_Danish_NRC = CF(
    CF = _IDENTITY, MODI = (
        "Ä" = "@",
        "Å" = "]",
        "Æ" = "[",
        "Ø" = "\",
        "Ü" = "^",
        "ä" = "`",
        "å" = "}",
        "æ" = "{",
        "ø" = "|",
        "ü" = "~",
        "" = %X00));
```

ZK−6592−GE

```
Multi_to_Finnish_NRC = CF(
    CF = _IDENTITY, MODI = (
        "Ä" = "[",
        "Å" = "]",
        "Ö" = "\",
        "Ü" = "^",
        "ä" = "{",
        "å" = "}",
        "é" = "`",
        "ö" = "|",
        "ü" = "~",
        "" = %X00));
```

ZK−6593−GE

```
Multi_to_FrCan_NRC = CR(
    CF = _IDENTITY, MODI = (
        "à" = "@",
        "â" = "[",
        "ç" = "\",
        "è" = "}",
        "é" = "{",
        "ê" = "]",
        "î" = "^",
        "ô" = "`",
        "ù" = "|",
        "û" = "~",
        "" = %X00));
```

ZK−6594−GE

```
Multi_to_French_NRC = CF(
    CF = _IDENTITY, MODI = (
        "£" = "#",
        "§" = "]",
        "°" = "[",
        "à" = "@",
        "ç" = "\",
        "é" = "}",
        "è" = "{",
        "ù" = "|",
        "â" = "^a",
        "ä" = "~a",
        "ê" = "^e",
        "ë" = "~e",
        "î" = "^i",
        "ï" = "~i",
        "ô" = "^o",
        "ö" = "~o",
        "û" = "^u",
        "ü" = "~u",
        "" = %X00));
```

ZK−6595−GE

```
Multi_to_German_NRC = CF(
    CF = _IDENTITY, MODI = (
        "§" = "@",
        "Ä" = "[",
        "Ö" = "\",
        "Ü" = "]",
        "ß" = "~",
        "ä" = "{",
        "ö" = "|",
        "ü" = "}",
        "" = %X00));
```

ZK−6596−GE

```
Multi_to_Italian_NRC = CF(
    CF = _IDENTITY, MODI = (
        "£" = "#",
        "§" = "@",
        "°" = "[",
        "à" = "{",
        "ç" = "\",
        "è" = "}",
        "é" = "]",
        "ì" = "~",
        "ò" = "|",
        "ù" = "´",
        "" = %X00));
```

ZK−6597−GE

```
Multi_to_Lower = CF(
    CF = _IDENTITY, MODI = (
        "A"-"Z" = "a"-"z",
        "À"-"Ï" = "à"-"ï",
        "Ñ"-"Ÿ" = "ñ"-"ÿ",
        "" = %X00));
```

ZK−6598−GE

```
Multi_to_NoDiacriticals = CF(
    CF = _IDENTITY, MODI = (
        "À"-"Å" = "A",
        "Ç" = "C",
        "È"-"Ë" = "E",
        "Ì"-"Ï" = "I",
        "Ñ"-"Ò" = "N"-"O",
        "Ó"-"Ö" = "O",
        "Ø" = "O",
        "Ù"-"Ü" = "U",
        "Ÿ" = "Y",
        "à"-"å" = "a",
        "ç" = "c",
        "è"-"ë" = "e",
        "ì"-"ï" = "i",
        "ñ"-"ò" = "n"-"o",
        "ó"-"ö" = "o",
        "ù"-"ü" = "u",
        "ÿ" = "y",
        "Æ" = "AE",
        "Œ - "OE",
        "ß" = "ss",
        "æ" = "ae",
        "œ" = "oe",
        "" = %X00));
```

ZK−6599−GE

```
Multi_to_Norwegian_NRC = CF(
    CF = _IDENTITY, MODI = (
        "Ä" = "@",
        "Å" = "]",
        "Æ" = "[",
        "Ø" = "\",
        "Ü" = "^",
        "ä" = "´",
        "å" = "}",
        "æ" = "{",
        "ø" = "|",
        "ü" = "~",
        "" = %X00));
```

ZK−6600−GE

```
Multi_to_Swedish_NRC = CF(
    CF = _IDENTITY, MODI = (
        "Ä" = "[",
        "Å" = "]",
        "É" = "@",
        "Ö" = "\",
        "Ü" = "^",
        "ä" = "{",
        "å" = "}",
        "é" = "`",
        "ö" = "|",
        "ü" = "~",
        "" = %X00));
```

ZK−6601−GE

```
Multi_to_Swiss_NRC = CF(
    CF = _IDENTITY, MODI = (
        "à" = "@",
        "ä" = "{",
        "ç" = "\",
        "è" = "_",
        "é" = "[",
        "ê" = "]",
        "î" = "^",
        "ô" = "`",
        "ö" = "|",
        "ù" = "#",
        "û"−"ü" = "}"−"~",
        "" = %X00));
```

ZK−6602−GE

```
Multi_to_Uk_NRC = CF(
     CF = _IDENTITY, MODI = (
         "£" = "#",
         "" = %X00));
```

ZK−6603−GE

```
Multi_to_Upper = CF(
    CF = _IDENTITY, MODI = (
        "a"−"z" = "A"−"Z",
        "à"−"ï" = "À"−"Ï",
        "ñ"−"ÿ" = "Ñ"−"Ÿ",
        "Œ" = "OE",
        "ß" = "SS",
        "" = %X00));
```

ZK−6604−GE

```
Norwegian = CS(
    SEQUENCE = (%X00-"N", "Ñ", "O"-"Z", "Æ", "Ö", "Å", "["-"´", "{"-"¿", %XD0,
           %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"z" = "A"-"Z", "À"-"Ã" = "A", "Ç" = "C", "È"-"Ë" = "E",
          "Ì"-"Ï" = "I", "Ò"-"Õ" = "O", "Ø" = "Ö", "Ù"-"Û" = "U", "Ü"-"Ÿ" = "Y",
          "à"-"ã" = "A", "å"-"æ" = "Å"-"Æ", "ç" = "C", "è"-"ë" = "E",
          "ì"-"ï" = "I", "ñ" = "Ñ", "ò"-"õ" = "O", "ö" = "Ö", "ø" = "Ö",
          "ù"-"û" = "U", "ü"-"ÿ" = "Y", "Ä" = "AE" = "Œ" = "OE", "ß" = "SS",
          "ä" = "Ä", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00-"A", "Ä"-"À", "B"-"C", "Ç", "D"-"E", "Ë"-"È",
          "F"-"I", "Ï"-"Ì", "J"-"N", "Ñ", "Œ", "O", "Ö"-"Ò", "P"-"R", "ß",
          "S"-"U", "Ü"-"Ù", "V"-"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["-"´", "{"-"¿",
          %XD0, %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"z" = "A"-"Z", "à"-"ï" = "À"-"Ï", "ñ"-"ÿ" = "Ñ"-"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6605−GE

```
Norwegian_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "@" = "Ä",
        "[" = "Æ",
        "\" = "Ø",
        "]" = "Å",
        "^" = "Ü",
        "´" = "ä",
        "{" = "æ",
        "|" = "ø",
        "}" = "å",
        "~" = "ü",
        "" = %X00));
```

ZK−6606−GE

```
Portuguese = CS(
    SEQUENCE = (%X00-"N", "Ñ", "O"-"Z", "Æ", "Ø", "Å", "["-"´", "{"-"¿", %XD0,
           %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"z" = "A"-"Z", "À"-"Ä" = "A", "Ç" = "C", "È"-"Ë" = "E",
          "Ì"-"Ï" = "I", "Ò"-"Ö" = "O", "Ù"-"Ü" = "U", "Ÿ" = "Y", "à"-"ä" = "A",
          "å"-"æ" = "Å"-"Æ", "ç" = "C", "è"-"ë" = "E", "ì"-"ï" = "I", "ñ" = "Ñ",
          "ò"-"ö" = "O", "ø" = "Ø", "ù"-"ü" = "U", "ÿ" = "Y", "Œ" = "OE",
          "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00-"A", "Ä"-"À", "B"-"C", "Ç", "D"-"E", "Ë"-"È",
          "F"-"I", "Ï"-"Ì", "J"-"N", "Ñ", "Œ", "O", "Ö"-"Ò", "P"-"R", "ß",
          "S"-"U", "Ü"-"Ù", "V"-"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["-"´", "{"-"¿",
          %XD0, %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"z" = "A"-"Z", "à"-"ï" = "À"-"Ï", "ñ"-"ÿ" = "Ñ"-"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6607−GE

```
Spanish = CS(
    SEQUENCE = (%X00-"B", "Ç", "CH", "D"-"L", "LL", "M"-"N", "Ñ", "O"-"Z", "Æ",
          "Ø", "Å", "["-"´", "{"-"¿", %XD0, %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"b" = "A"-"B", "d"-"k" = "D"-"K", "m"-"z" = "M"-"Z",
          "À"-"Ä" = "A", "È"-"Ë" = "E", "Ì"-"Ï" = "I", "Ò"-"Ö" = "O",
          "Ù"-"Ü" = "U", "Ÿ" = "Y", "à"-"ä" = "A", "å"-"ç" = "Å"-"Ç",
          "è"-"ë" = "E", "ì"-"ï" = "I", "ñ" = "Ñ", "ò"-"ö" = "O", "ø" = "Ø",
          "ù"-"ü" = "U", "ÿ" = "Y", "C" = "Ç", "Ch" = "CH", "Ll" = "LL",
          "c" = "Ç", "cH" = "CH", "ch" = "CH", "l" = "L", "lL" = "LL",
          "ll" = "LL", "Œ" = "OE", "ß" = "SS", "œ" = "Œ", "" < %X00))
    + CS( SEQUENCE = (%X00-"A", "Ä"-"À", "B"-"C", "Ç", "D"-"E", "Ë"-"È",
          "F"-"I", "Ï"-"Ì", "J"-"N", "Ñ", "Œ", "O", "Ö"-"Ò", "P"-"R", "ß",
          "S"-"U", "Ü"-"Ù", "V"-"Y", "Ÿ", "Z", "Æ", "Ø", "Å", "["-"´", "{"-"¿",
          %XD0, %XDE, %XF0, %XFE-%XFF),
    MODIFICATIONS=("a"-"z" = "A"-"Z", "à"-"ï" = "À"-"Ï", "ñ"-"ÿ" = "Ñ"-"Ÿ"))
    + REVERSE(_NATIVE);
```

ZK−6608−GE

```
Swedish_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "@" = "É",
        "[" = "Ä",
        "\" = "Ö",
        "]" = "Å",
        "^" = "Ü",
        "´" = "é",
        "{" = "ä",
        "|" = "ö",
        "}" = "å",
        "~" = "ü",
        "" = %X00));
```

ZK−6609−GE

```
Swiss_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "#" = "ù",
        "@" = "à",
        "[" = "é",
        "\" = "ç",
        "]" = "ê",
        "^" = "î",
        "_" = "è",
        "´" = "ô",
        "{" = "ä",
        "|" = "ö",
        "}"-"~" = "ü"-"û",
        "" = %X00));
```

ZK−6610−GE

```
Uk_NRC_to_Multi = CF(
    CF = _IDENTITY, MODI = (
        "#" = "£",
        "" = %X00));
```

ZK−6611−GE

Index