HP 3000 Computer Systems

# NATIVE LANGUAGE SUPPORT REFERENCE MANUAL

HEWLETT PACKARD
19447 PRUNERIDGE AVENUE, CUPERTINO, CA 95014

Part No. (32414-90001)
E1187

Printed in U.S.A. 11/87

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

# List of Effective Pages

The List of Effective Pages gives the date of the current edition, and lists the dates of all changed pages. Unchanged pages are listed as "ORIGINAL". Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars and dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

| | |
|---|---|
| First Edition | September 1984 |
| Update #1 | August 1986 |
| Update #2 | November 1986 |
| Second Edition | November 1987 |

# Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update.

| | |
|---|---|
| First Edition | September 1984 |
| Update #1 | August 1986 |
| Update #1 Incorporated | November 1986 |
| Update #2 | November 1986 |
| Update #2 Incorporated | November 1986 |
| Second Edition | November 1987 |

# MPE V Manual Plan

There are many manuals applicable to the HP 3000 that are not listed here. A complete list may be found in each issue of the MPE V Communicator. Please contact your System Manager.

INTRODUCTORY LEVEL:

GENERAL INFORMATION
Manual
5953-7553

GUIDE FOR THE NEW USER
32033-90009

GUIDE FOR THE NEW OPERATOR
32033-90021

STANDARD USER LEVEL:

MPE V COMMANDS
Reference Manual
32033-90006

MPE V INTRINSICS
Reference Manual
32033-90007

MPE V UTILITIES
Reference Manual
32033-90008

SEGMENTER
Reference Manual
30000-90011

DEBUG/STACK DUMP
Reference Manual
30000-90012

FILE SYSTEM
Reference Manual
30000-90236

ADMINISTRATIVE LEVEL:

MPE V SYSTEM OPERATION & RESOURCE MANAGEMENT
Reference Manual
32033-90005

SUMMARY LEVEL:

MPE V QUICK REFERENCE GUIDE
32033-90023

# Conventions

**NOTATION** **DESCRIPTION**

UPPERCASE

Within syntax statements, characters in uppercase must be entered in exactly the order shown, though you can enter them in either uppercase or lowercase. For example:

```
SHOWJOB
```

Valid entries: showjob ShowJob SHOWJOB

Invalid entries: shojwob Shojob SHOW_JOB

*italics*

Within syntax statements, a word in italics represents a formal parameter or argument that you must replace with an actual value. In the following example, you must replace *filename* with the name of the file you want to release:

RELEASE *filename*

punctuation

Within syntax statements, punctuation characters (other than brackets, braces, vertical parallel lines, and ellipses) must be entered exactly as shown.

{ }

Within syntax statements, braces enclose required elements. When several elements within braces are stacked, you must select one. In the following example, you must select ON or OFF:

```
        { ON }
SETMSG  {OFF}
```

[ ]

Within syntax statements, brackets enclose optional elements. In the following example, brackets around ,TEMP indicate that the parameter and its delimiter are optional:

PURGE *filename*[,TEMP]

When several elements within brackets are stacked, you can select any one of the elements or none. In the following example, you can select *devicename* or *deviceclass* or neither:

```
         [devicename]
SHOWDEV  [ deviceclass]
```

# Conventions (Continued)

**NOTATION** **DESCRIPTION**

[...]

Within syntax statements, a horizontal ellipsis enclosed in brackets indicates that you can repeatedly select elements that appear within the immediately preceding pair of brackets or braces. In the following example, you can select *itemname* and its delimiter zero or more times. Each instance of *itemname* must be preceded by a comma:

[,*itemname*][...]

If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, the comma cannot precede the first instance of *itemname*:

[*itemname*][,...]

|...|

Within syntax statements, a horizontal ellipsis enclosed in parallel vertical lines indicates that you can select more than one element that appears within the immediately preceding pair of brackets or braces. However, each element can be selected only one time. In the following example, you must select ,A or ,B or ,A,B or ,B,A :

{,A}  
{,B}|...|

If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, you must select A or B or A,B or B,A . The first element cannot be preceded by a comma:

{A}  
{B}|,...|

... ⋮

Within examples, horizontal or vertical ellipses indicate where portions of the example are omitted.

△

Within syntax statements, the space symbol △ shows a required blank. In the following example, you must separate *modifier* and *variable* with a blank:

SET[(*modifier*)]△(*variable*);

# Conventions (Continued)

**NOTATION** | **DESCRIPTION**

▭

The symbol ▭ indicates a key on the terminal's keyboard. For example, CTRL indicates the Control key.

CTRL *char*

CTRL *char* indicates a control character. For example, CTRL Y means you have to simultaneously press the Control key and the Y key on the keyboard.

base prefixes

The prefixes %, #, and $ specify the numerical base of the value that follows:

> *%num* specifies an octal number
> *#num* specifies a decimal number
> *$num* specifies a hexadecimal number

When no base is specified, decimal is assumed.

Bit (*bit:length*)

When a parameter contains more than one piece of data within its bit field, the different data fields are described in the format Bit (*bit:length*), where *bit* is the first bit in the field and *length* is the number of consecutive bits in the field. For example, Bits (13:3) indicates bits 13, 14, and 15:

most significant | least significant

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | 13 | 14 | 15 |

Bit(0:1) | Bits(13:3)

# Table of Contents

## Chapter 1: Introduction

## Chapter 2: Application Message Facility

## Chapter 3: NLS in MPE Subsystems

## Chapter 4: Native Language Intrinsics

## Appendix A: System Utilities

## Appendix B: SUPPORTED LANGUAGES AND CHARACTER SETS

## Appendix C: COLLATING IN EUROPEAN LANGUAGES

## Appendix D: EBCDIC MAPPINGS

## Appendix E: PERIPHERAL CONFIGURATION

## Appendix F: CONVERTING 7-BIT TO 8-BIT DATA

## Appendix G: APPLICATION GUIDELINES

## Appendix H: EXAMPLE PROGRAMS

# Preface

Native Language Support (NLS) provides the HP 3000 with the features necessary to produce localized application programs for end users without reprogramming for each country or language.

Native Language Support consists of Multi-Programming Executive (MPE) intrinsics, additional features in COBOLII, and the FCOPY/3000, IMAGE/3000, KSAM/3000, QUERY/3000, SORT-MERGE/3000, RAPID/3000, and VPLUS/3000 subsystems, the Application Message Facility, plus utilities to install and implement native language capabilities.

This release of Native Language Support incorporates new languages and their character sets. It also presents additions to the set of Intrinsics that are available to the user:

NLJUDGE Judges whether a character is a one-byte or two-byte Asian character.

NLSUBSTR Extracts one string from another string.

NLFINDSTR Searches a string for another string.

# Introduction 1

Hewlett-Packard Native Language Support (NLS) features enable the applications designer/programmer to create local language applications for the end user.

## Background

A well-written application program manipulates data and presents it appropriately for its use and user. Users who are less technically sophisticated benefit from application programs which interact with them in their native language, and which conform to their local customs. Native language refers to the user's first language (learned as a child), such as Finnish, Portuguese, or Japanese. Local customs refer to conventions such as local date, time, and currency formats.

Programs written with the intention of providing a friendly user interface often make assumptions about the local customs and language of the user. Program interface and processing requirements vary from country to country, and sometimes within a country. Most existing software does not take this into account and is appropriate for use only in the country or locality in which it is written.

The solution to this problem is to design application programs that can be easily localized. Localization is the adaptation of a software application or system for use in different countries or local environments. In such an environment, the user's native language and/or data processing requirements may differ from those in the environment of the software developer. Traditionally, localization has been achieved by modifying a program for each specific country. Applications designed with localization in mind provide a better solution. Localization can then be accomplished with (ideally) no modification of code at all.

An applications designer must write the application program with built-in provisions for localization. Functions which are local language or custom dependent cannot be hard-coded. For example, all messages and prompts must be stored in an external file or catalog. Character comparisons and upshifting must be accomplished by external system-level routines or instructions. The external files and catalogs can be translated, and the program localized without rewriting or recompiling the application program.

Native Language Support (NLS) provides the tools for an applications designer/programmer to produce localizable applications. These tools may include architecture and peripheral support, as well as software facilities within the operating systems and subsystems. NLS addresses the internal functions of a program (for example, sorting) as well as its user interface (for example, messages and formats).

## Scope

HP 3000 Native Language Support (NLS) consists of features within MPE, as well as in the FCOPY/3000, IMAGE/3000, KSAM/3000, QUERY/3000, SORT-MERGE/3000, VPLUS/3000, RAPID/3000, and COBOLII subsystems. These facilities allow application programs to be designed and written with a local language interface for the end user and locally correct internal processing. The end user can see localized programs produced by an applications designer/programmer who has used the NLS tools.

The MPE interface, subsystems, programmer productivity tools, and compilers have not been localized. The applications designer must still interact with MPE and the subsystems using American English. For the designer/programmer, the interface has not changed. For example, it is possible to write a complete local language application program using COBOLII and VPLUS/3000, but the COBOLII compiler and the VPLUS/3000 FORMSPEC program retain their English-like characteristics.

Not all functions which vary from one language to another, or one country to another, are provided by NLS. For example, tax calculation rules are usually country-specific or local-specific, and rules for word hyphenation are related to individual languages. Functions such as these are considered to be application-specific, and are beyond the scope of NLS.

## Supported Native Languages

NLS is based on languages and character sets which have been predefined and built into the operating system. These are referred to as supported languages. A unique language name and language ID number has been assigned to each language supported in NLS. In some cases, more than one supported language has been introduced corresponding to a single natural language. For example, NLS supports FRENCH (language number 7) and CANADIAN-FRENCH (language number 2). Upshifting is handled differently in FRENCH and CANADIAN-FRENCH. When language-dependent characteristics differ within the same natural language, NLS can create separate native languages to represent these differences.

Each of the supported languages may also be considered a "language family" which is applicable in several countries. GERMAN (language number 8), for example, may be used in Germany, Austria, Switzerland, and any other place it is requested.

In addition to the native languages supported, an artificial language, NATIVE-3000 (language number 0), represents the way the computer used to deal with language before the introduction of NLS. For example, the collating sequence (the sequence in which characters acceptable to the computer are ordered) for NATIVE-3000 is the order of characters in the USASCII code and the date format is returned by the existing MPE intrinsic, FMTDATE. Whenever language number 0 is used in a native language function, the result will be identical to the function performed before the introduction of NLS. NLS intrinsic calls with the language parameter equal to 0 will always work correctly, even if no native languages have been configured on the system.

Refer to Appendix B, "Supported Languages and Character Sets" for listings of the languages supported, their character sets, and their identification numbers (*langnum* values).

## Character Sets

Within NLS, each supported language is associated with an 8-bit or 16-bit character set (one character set may support many languages). Like languages, character sets have defined names and ID numbers assigned, although these names and numbers are not widely used, except, in documentation. Before the introduction of NLS, the only widely-supported character set was USASCII, a 128-character set designed to support American English text. USASCII uses only seven bits of an 8-bit byte to encode a character. The eighth or high order bit is always zero. For this reason, USASCII is referred to as a "7-bit" code.

An 8-bit byte has the capacity to contain 256 unique values, which means it is possible to build supersets of USASCII which permit encoding and manipulation of characters required by languages other than American English. These supersets are referred to as "8-bit" or "extended" character sets. New characters are added with code values in the range 161-254.

Another method of providing foreign characters (not supported by NLS) involves replacing as many as 12 existing characters in USASCII with substitution characters. The 7-bit substitution set eliminates some characters in favor of others needed by a particular local language. A different substitution set is necessary for each language. NLS 8-bit character sets support all USASCII characters (with the exception of "\" in KANA8) in addition to the characters needed to support several western European-based languages and KATAKANA.

The use of 8-bit or 16-bit character sets for NLS implies that in character data, all bits of every byte have significance. Application software must take care to preserve the eighth (high order) bit, nowhere allowing it to be modified or reused for any special purpose. Also, no differentiation should be made between characters having the eighth bit turned off and those with it turned on, because all are characters of equal status in the extended character set.

Refer to Appendix B, "Supported Languages and Character Sets" for a list of native languages supported by each character set.

## Language-Dependent Characteristics

For each native language which is supported by NLS, a number of characteristics are known. These are lexical conventions (for example, collating sequence and upshifting rules), country or local custom-dependent formats (currency symbols, date, time, and number formats), and data processing conversion tables:

- Lexical conventions vary from country to country. The collating sequence is affected by the local alphabet and usage of each language. Upshifting tables maintained by NLS for each supported language contain the appropriate result of upshifting any character in the corresponding character set. This category of information is really language-related in the literal sense.
- Currency symbols, date, time, and number formats are country and local custom dependent. Currency symbols and their position in relation to numbers depend on local custom. Date, time, and number formats also vary from country to country.
- Data processing tables for ASCII-to-EBCDIC and EBCDIC-to-ASCII conversion are affected by language because the EBCDIC codes are different from country to country.

Within NLS, characteristics that are language related, custom-dependent, and data processing oriented are all considered to be language-dependent. All information used by, or available from, NLS is based on the application's choice of language(s). For example, NLS maintains an ENGLISH collating sequence and an ENGLISH time-of-day format. In this context, ENGLISH refers specifically to that used in England rather than the English language. (AMERICAN refers to the language, formats, and tables used in the United States.)

Refer to Appendix B, "Supported Languages and Character Sets" for a complete list of supported languages and language characteristics. Detailed information on any particular installed language is available programmatically via the NLINFO intrinsic (refer to Chapter 4, "Native Language Intrinsics") or in report form from the NLUTIL program.

## Native Language Support in MPE

The MPE components of NLS consist of utility programs (LANGINST and NLUTIL), system intrinsics, and an application message facility.

### NLS System Utilities

LANGINST is used by system managers to select the native languages to be supported on their system(s). NLUTIL is used to obtain the details of languages installed on a system. Refer to Appendix A, "System Utilities" for a description of LANGINST and NLUTIL.

## Configuring Native Languages

Before any native languages (except NATIVE-3000) can be used on a system, they must be configured by the System Manager using the LANGINST utility program. Refer to Appendix A, "System Utilities" for the LANGINST user dialog. The System Manager can select which supported languages to configure and can modify several formats associated with the language(s) being configured. For example, this feature is useful to a System Manager in Austria who wants to install GERMAN with a different currency symbol than the default for this language. Changes to a system's language configuration are effective after the next system startup, at which time the configured languages are installed. After a language has been installed, language-specific information available in NLS may be used by any application program requesting it.

## NLS Intrinsics

The NLS intrinsics may be called by application programs and Hewlett-Packard subsystems to provide language-dependent functions and information for any language installed on a system. For example, the NLFMTDATE intrinsic returns a locally formatted date, and the NLCOLLATE intrinsic compares two character strings using a language-dependent collating sequence. Refer to Chapter 4, "Native Langauge Intrinsics" for a complete list of NLS intrinsics. Some HP 3000 subsystems call NLS intrinsics to perform certain functions. For example, configured native languages can affect the collating sequence used by SORT-MERGE/3000, the numeric formatting done by VPLUS/3000, and the EBCDIC conversions performed by FCOPY/3000. Refer to Chapter 3, "NLS in the Subsystems" for specific information.

**NOTE**

None of the above changes are automatic. All existing applications and jobs will function the same way they did previous to the installation of NLS, unless they are modified to request NLS functions.

## Peripheral Support

Peripherals configured for any of the 7-bit substitution sets are not supported by NLS.

Most Hewlett-Packard peripherals are designed for 8-bit operation. Most peripherals that have been configured for 7-bit operation can be reconfigured for 8-bit operation.

NLS has no direct control over what peripherals are configured on a system. The user must configure the peripherals which will support the character set(s) necessary for the desired languages. Refer to Appendix E, "Peripheral Configuration" for instructions.

## Conversion Utilities

Data encoded according to any 7-bit substitution set is not supported by NLS. Users with data encoded in one or more of the European 7-bit substitution sets supported on the older Hewlett-Packard terminals and printers have the option to convert this data. A set of utilities is available to convert 7-bit data to 8-bit (ROMAN8) data in KSAM files, IMAGE/3000 databases, VPLUS/3000 forms files, and MPE files. Refer to Appendix F, "Converting 7-Bit to 8-Bit Data," for conversion instructions.

## Application Message Facility

A localizable program contains no text (prompts, commands, messages) stored in the code itself. This allows the text to be translated (part of the localization process) without modifying the source code of a program or recompiling it. Therefore, a good text handling facility is essential to Native Language Support.

The principal tool supplied within NLS for text handling is the Application Message Facility. The application message catalog facility consists of the GENCAT utility program and the CAT intrinsics (CATREAD, CATOPEN, and CATCLOSE). The application message catalog facility provides efficient storage and retrieval of program messages, commands, and prompts. The GENCAT program is used to convert an ASCII source file containing messages into a binary application catalog that can be accessed by the intrinsics. Application programs use the CAT intrinsics to retrieve messages from it. An application message catalog consists of a file containing character strings (messages), each uniquely identifiable by a set number and a message number within a set. Key features of the Application Message Facility include:

- Each message in a catalog can allow up to five parameters which may be specified by position or number.
- An editor is used to create the source catalog (an MPE ASCII file). The GENCAT program is used to read the source catalog and to create a formatted catalog. The formatted catalog has an internal directory for efficient access and is compacted (for example, by deleting trailing blanks) to optimize storage space.
- GENCAT has a facility to merge two message source files; a master file and a maintenance file. The maintenance file contains changes to be made in the master file. Updates of a localized version of an application may be made by translating the maintenance file, then merging it with the localized source catalog.
- Multiple localized versions of an application can be supported with translations of the original source catalog. If a naming convention is established, the application program can determine which localized catalog to open at run time (using the CATOPEN intrinsic). Refer to Chapter 2, "Application Message Facility" for suggested naming conventions.

The application message facility is documented in Chapter 2, "Application Message Facility."

## File Naming Conventions

An application which has been localized into several languages will have separate message catalogs, VPLUS/3000 forms files, and/or various other language-dependent data files for each of these languages. It is suggested that a naming convention be established for these files which follows the language numbering used by NLS. To do this, a file name should be used which is up to five identifying characters followed by a three-digit language number, corresponding to the language of the file contents. For example, the original, unlocalized data might be stored in a file whose name is FILE000; FILE008 would contain the same data modified for German, and FILE012 would contain the data modified for Spanish. It is the responsibility of the application program to determine, at run time, which file to open. Once the language number is determined, the NLAPPEND intrinsic may be used to form the file name if this convention is followed.

## NLS in the Subsystems

In addition to the new utilities and MPE intrinsics, NLS provides features in COBOLII, FCOPY/3000, IMAGE/3000, KSAM, QUERY/3000, SORT-MERGE/3000, VPLUS/3000, and RAPID/3000. NLS features in these subsystems are intended to provide the applications designer/programmer with the tools to design local language applications. The subsystems themselves are not localized. The application end user, not the programmer or subsystem user, will see the localized interface.

MPE Native Language Support intrinsics provide the means to implement NLS features contained in the subsystems. This means that native language definitions are consistent in all the subsystems. For example, the collating sequence is consistant within MPE and in the subsystems and can be defined for a specific native language by calling the NLCOLLATE and NLKEYCOMPARE intrinsics. The same collating sequence is used by SORT-MERGE/3000 in ordering records, by KSAM/3000 in ordering keys, and by IMAGE/3000 in ordering sorted chains when these subsystems are dealing with sorted character strings that have been associated with the same native language.

The MPE operating system and its subsystems function independently of native language features configured on the system. NLS features are optional and must be requested to be invoked; existing application software and stream files will operate as they did before the introduction of NLS.

## Accessing NLS Features

On HP 3000 systems using MPE and subsystems with NLS features, all NLS features are optional. These features must be requested by the applications programmer through intrinsic calls or interactively by the user of a subsystem program through a LANGUAGE command or keyword.

### Intrinsics

NLS features may be obtained from application programs through calls to specific NLS intrinsics, primarily in MPE. For example, to get a local language date format, an application should call the NLS intrinsic NLFMTDATE instead of the old FMTDATE intrinsic.

### Additional Parameter Values In Existing Intrinsics

Another way is by specifying values for extended or new parameters in existing intrinsics. For example, SORTINIT in SORT-MERGE/3000 has been extended to allow the specification of a CHARACTER key and a native language ID number (*langnum*) which determines the collating sequence to be used. These additional parameters must be used in an application to sort according to native language values.

### Native Language Attribute

Some subsystem structures, including IMAGE/3000 databases, KSAM/3000 files, and VPLUS/3000 forms files may be assigned a language attribute by their creators. The language attribute will ensure that certain functions will perform according to localized specifications at run time. VPLUS/3000, for example, will perform its upshift function according to the language of the forms file.

### Commands

Commands or keywords have been added to certain subsystems which make NLS features available on request. For example, entering LANGUAGE=FRENCH within QUERY/3000 would cause sorted character data of IMAGE/3000 types X and U to be sorted according to the FRENCH collating sequence in its output reports. If the language command is not entered, QUERY/3000 (or any other subsystem) will perform as it did before the introduction of NLS. If these commands are not used, the default language(s) used by subsystem utility programs can be influenced by the values of the two NLS Job Control Words, NLUSERLANG and NLDATALANG.

Some general suggestions for designing applications incorporating NLS features and specific strategies for using major programming languages are included in Appendix G, "Application Guidelines."

Refer to Chapter 3, "NLS in MPE Subsystems" for information on how and when the individual subsystems are influenced.

## Implicit Language Choice in Subsystems

Two NLS Job Control Words (JCWs), NLUSERLANG and NLDATALANG, permit the subsystem user to designate a default language other than NATIVE-3000 for the subsystems. Each of the five subsystem programs (SORT, MERGE, FCOPY/3000, QUERY/3000, ENTRY) looks at one of these JCWs, and its value is used as a default language by the program. The default can be superseded by a specific command. Utility programs in the subsystems are often run within user-defined commands (UDCs). UDCs are often created for the convenience of a less sophisticated computer user than the person who designed them. To add to this convenience, NLS has established a convention for designating the native language choice for operation of the subsystem programs that does not require the user to enter a language explicitly. This is accomplished through the use of two reserved Job Control Words (JCWs), NLUSERLANG and NLDATALANG:

- NLUSERLANG designates the user interface and report output language for programs. If the subsystems were localized, this would be the language of choice for prompts and messages. If user input data is modified (for example, upshifted by QUERY or VPLUS), this language determines which language's attributes are used. The default language for all language-dependent operations in QUERY/3000 and ENTRY can be designated.
- NLDATALANG designates the internal data manipulation language. One reason this is distinct from NLUSERLANG is that multiple users with different interface languages may wish to share some common internal data (for example, sorted according to one language). The data manipulation language is used in the SORT, MERGE, and FCOPY/3000 programs to control their language-dependent functions, such as collating, upshifting, and conversions to and from EBCDIC.

**NOTE**

If the user interface of one of these programs were localized, it would use NLUSERLANG as its default for messages, prompts, etc.

NLUSERLANG and NLDATALANG are independent JCWs, and are treated independently by NLS. In many cases, they will specify the same language, but examples already exist in which they could have been used with distinct values.

### The NLGETLANG Intrinsic

NLUSERLANG and NLDATALANG values are retrieved by the subsystems through calls to the NLGETLANG intrinsic. Application programs may also use this intrinsic. NLGETLANG retrieves the value of the language attribute requested, and verifies its installation. If the value is that of an unconfigured or undefined language, NLGETLANG will return a language ID number of 0 (NATIVE-3000) and an error. To use either JCW, set the integer value corresponding to the language ID number desired, using :SETJCW. Refer to the *MPE V/E Commands Reference Manual* (32033-90006), for the :SETJCW command syntax.

### User-Defined Commands (UDCs)

ENTRY, FCOPY/3000, QUERY/3000, SORT, and MERGE are often run from within user-defined commands (UDCs). The two NLS Job Control Words (JCWs) give the user the option of establishing a native language within a UDC.

## Application Programs

The focus of NLS is the application program. Most NLS tools are accessed programmatically from applications according to the requirements of the designer or programmer. Several common application models are possible. These are illustrated in Figures 1-1 to 1-5. NLS capabilities can be used in single language applications, multilingual applications, in subsystem utility programs, or not at all.

### General Application Program

The functions language can influence an application in terms of data manipulation (internals) and user interaction (externals) is illustrated in Figure 1-1. The core application program is flanked by functions that can differ according to language and local customs (local date, time, and currency formats).

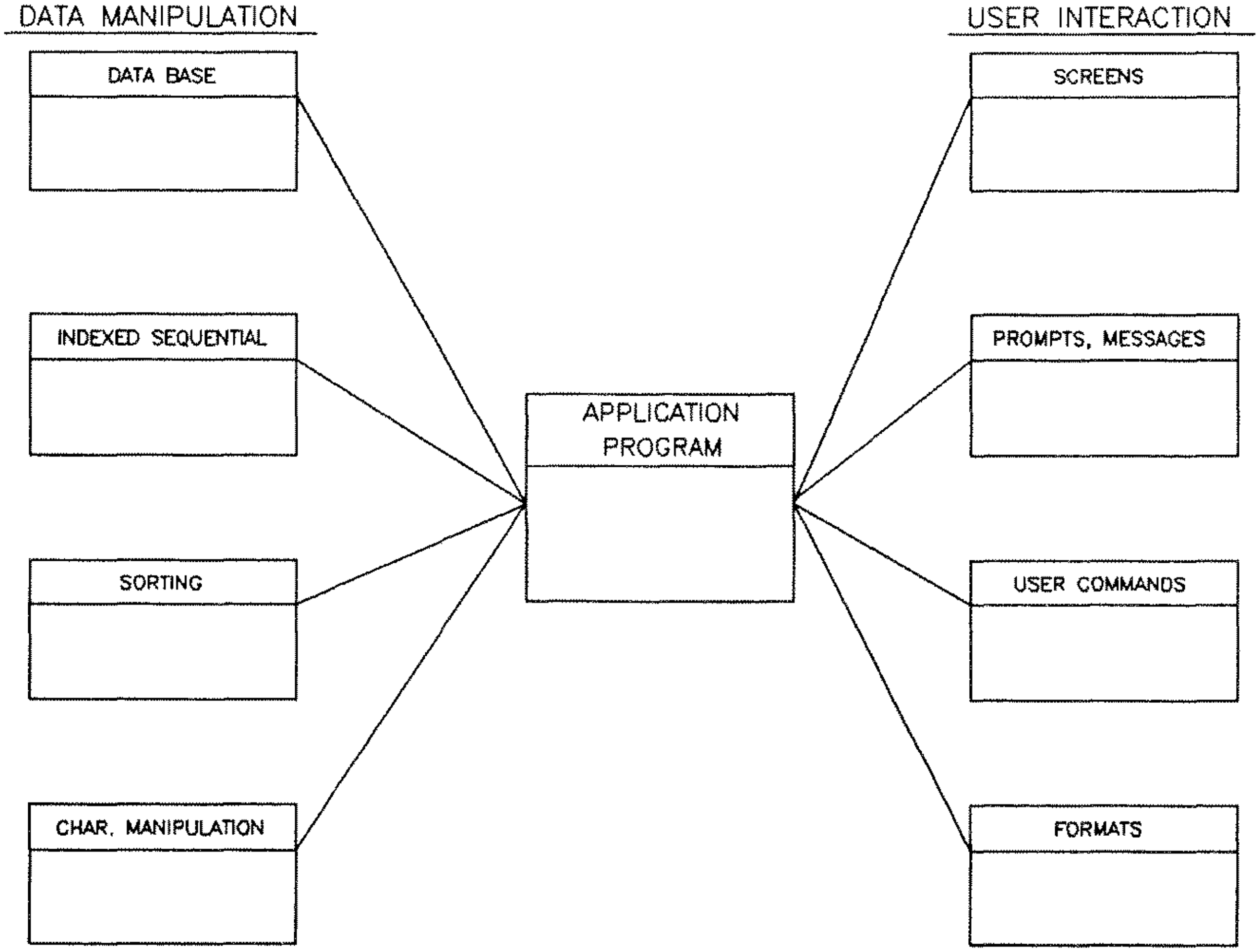


**Figure 1-1. Application Program Format**

## Application Program Without NLS

Figure 1-2 shows an application program which does not make use of NLS capabilities. This NATIVE-3000 application makes use of conventional programming techniques and standard MPE and subsystem features to achieve the key language-dependent functions. It cannot be localized without reprogramming and is unaffected by the introduction of NLS.

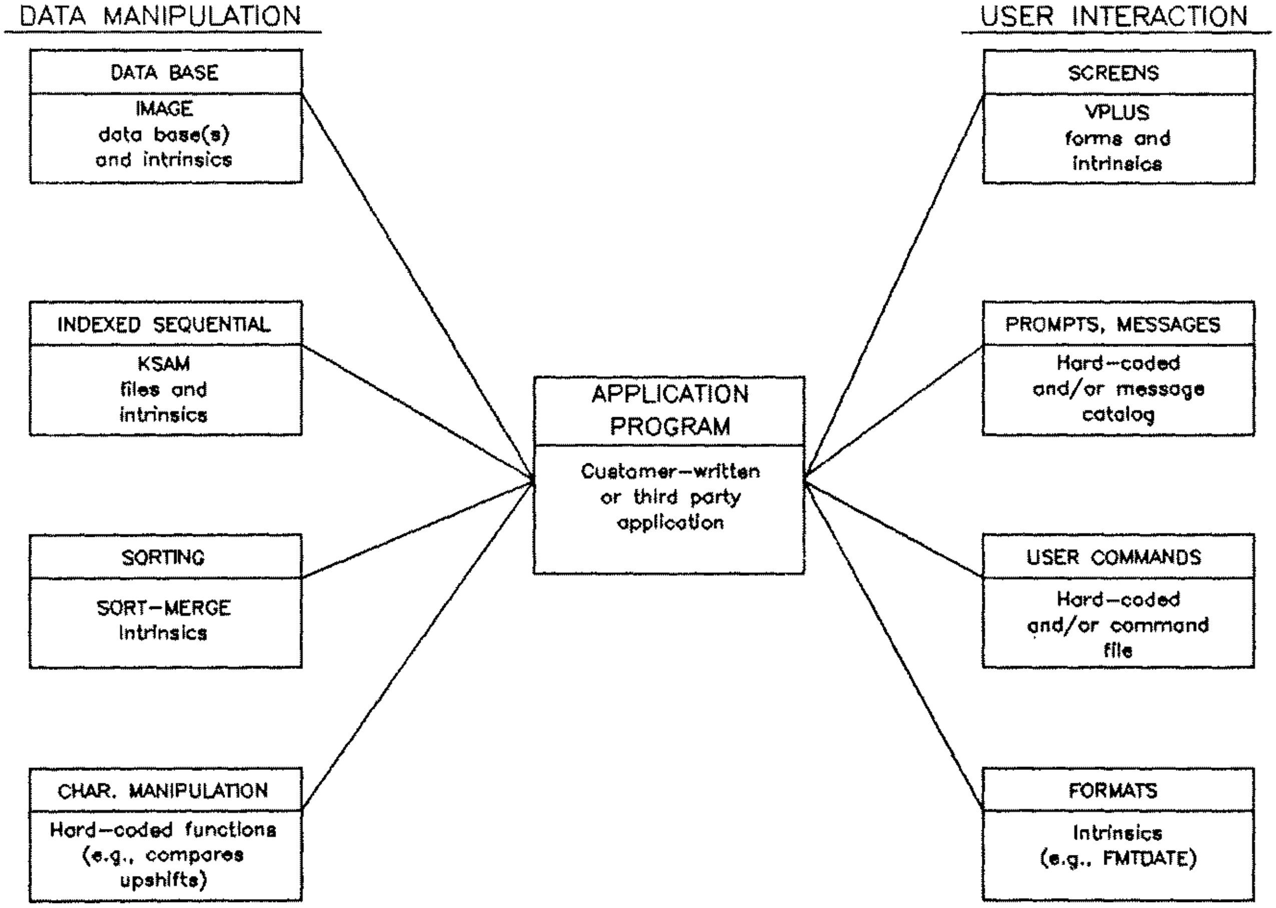

DATA MANIPULATION
DATA BASE
IMAGE data base(s) and intrinsics
INDEXED SEQUENTIAL
KSAM files and intrinsics
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
Hard-coded functions (e.g., compares upshifts)
APPLICATION PROGRAM
Customer-written or third party application
USER INTERACTION
SCREENS
VPLUS forms and intrinsics
PROMPTS, MESSAGES
Hard-coded and/or message catalog
USER COMMANDS
Hard-coded and/or command file
FORMATS
Intrinsics (e.g., FMTDATE)

**Figure 1-2. Application Program Without NLS**

## Single Language Application

French is used as the single language application example in Figure 1-3. The applications designer has determined that only French is required, and has hard-coded its language ID number (*langnum*) 7 into the program. The *langnum* is used as a parameter in calling various native language-dependent intrinsics. In addition, the designer has created IMAGE/3000 databases, KSAM/3000 files, and VPLUS/3000 forms files with the French language attribute, and has expressed all prompts and messages in French. This use of NLS is for programs which will only be used in one country or location, or with only one language.

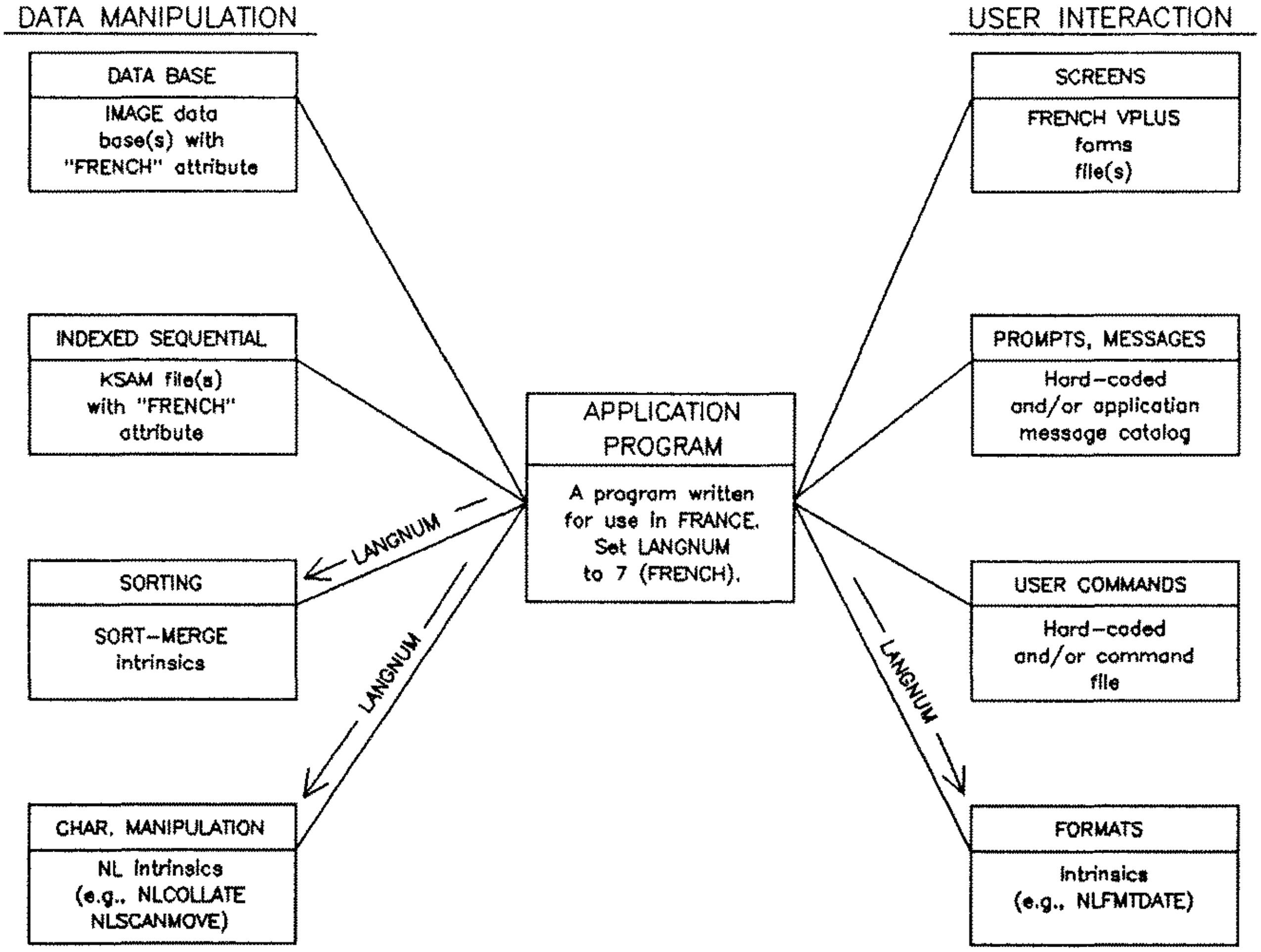

DATA MANIPULATION
USER INTERACTION
DATA BASE
IMAGE data base(s) with "FRENCH" attribute
INDEXED SEQUENTIAL
KSAM file(s) with "FRENCH" attribute
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
APPLICATION PROGRAM
A program written for use in FRANCE. Set LANGNUM to 7 (FRENCH).
SCREENS
FRENCH VPLUS forms file(s)
PROMPTS, MESSAGES
Hard-coded and/or application message catalog
USER COMMANDS
Hard-coded and/or command file
FORMATS
Intrinsics (e.g., NLFMTDATE)
LANGNUM
LANGNUM
LANGNUM

**Figure 1-3. Single Language Application**

## Multilingual Application

The program in Figure 1-4 shows a localizable or multilingual application. This application can be used in several countries or in multiple languages by different users on the same system. The key attribute of this program is that it selects its language(s) at run time.

When installing an application on a system, the manager of the application may establish configuration files for that application. These files store information about various users or transactions and their native language requirements. At run time the application program can determine which language(s) to use.

The program may call the NLGETLANG intrinsic to obtain the system default language (set by the System Manager when native languages are configured) or it may prompt the user to enter a language name or ID number (*langnum*).

The application may call NLGETLANG to obtain the user interface language and/or the data manipulation language. The Job Control Words NLUSERLANG and NLDATALANG must be in place before invoking this type of application. This method could be restrictive if many users or transactions are handled from one job or session.

Once the languages have been determined, the program opens the appropriate VPLUS/3000 forms files, message catalogs, and/or command files, based on the user interface language choice. It also opens any needed IMAGE/3000 databases, KSAM/3000 files, or general data files; these may or may not depend upon language choice. The appropriate language ID numbers are used in calling the various native language intrinsics. Different users may concurrently run the same program with different languages. The application can be designed to use more than one language within a single execution. For example, one language may be used for data manipulation and a different one for user interactions.

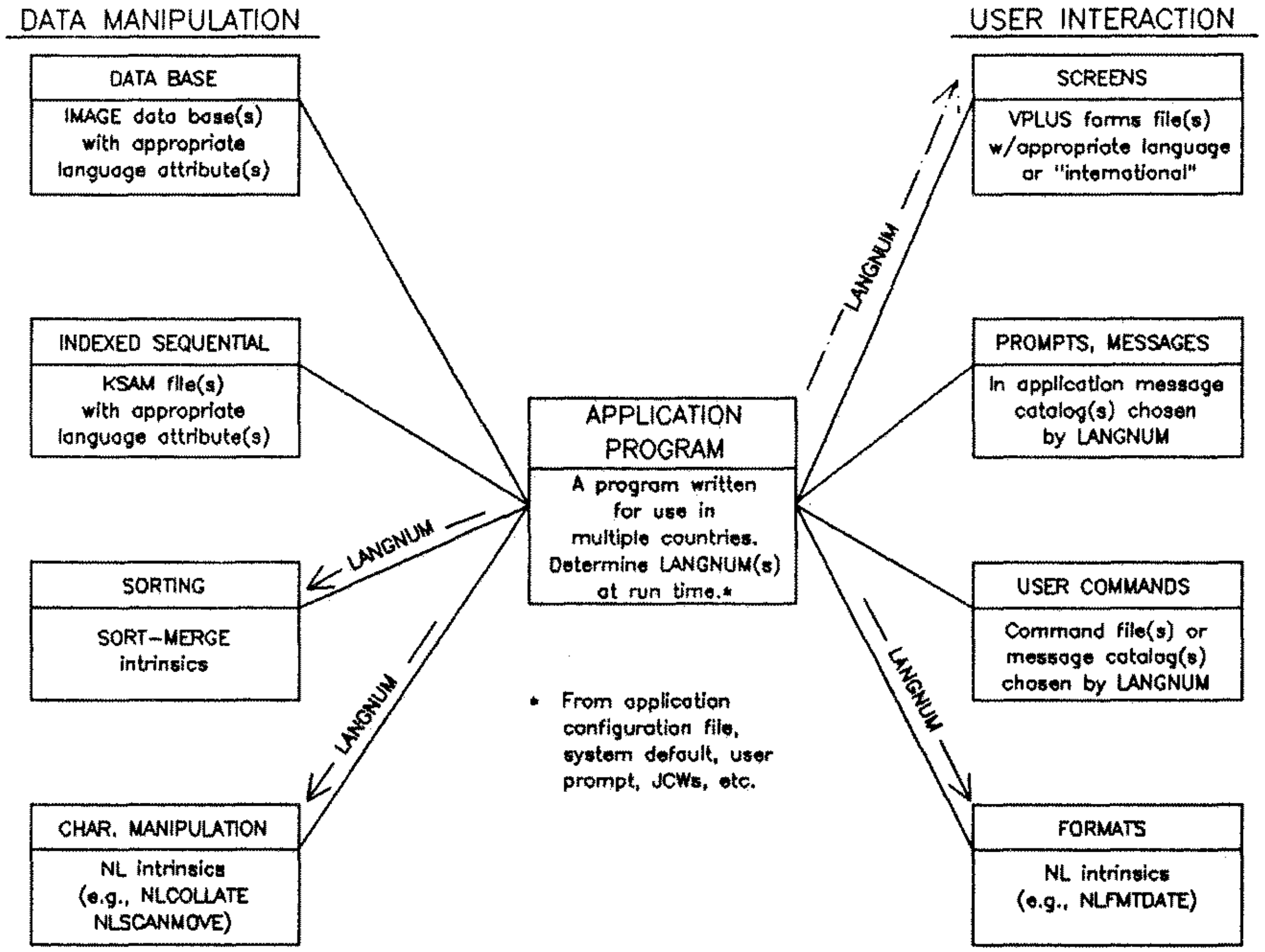
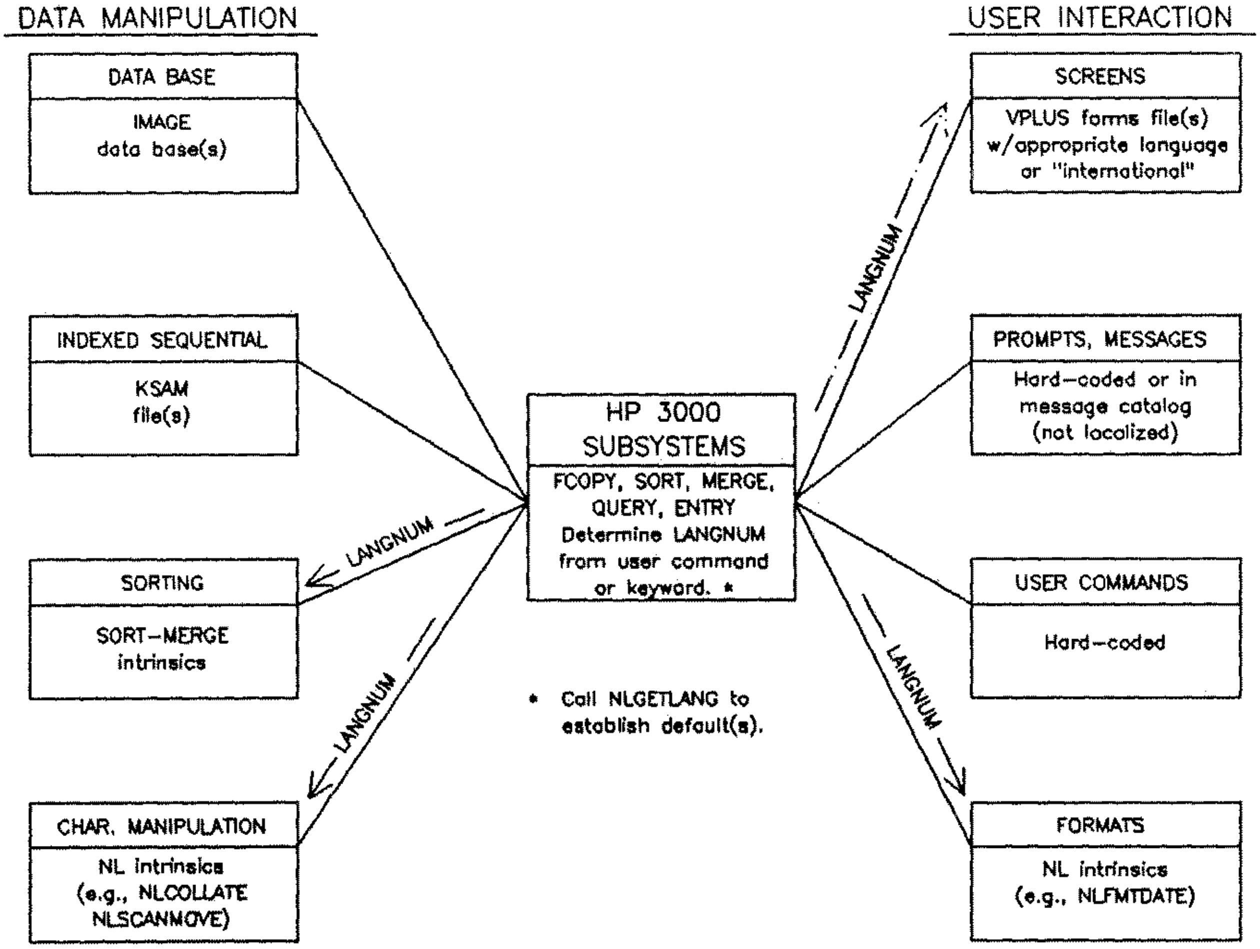

DATA MANIPULATION
USER INTERACTION
DATA BASE
IMAGE data base(s) with appropriate language attribute(s)
INDEXED SEQUENTIAL
KSAM file(s) with appropriate language attribute(s)
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
APPLICATION PROGRAM
A program written for use in multiple countries. Determine LANGNUM(s) at run time.*
* From application configuration file, system default, user prompt, JCWs, etc.
SCREENS
VPLUS forms file(s) w/appropriate language or "international"
PROMPTS, MESSAGES
In application message catalog(s) chosen by LANGNUM
USER COMMANDS
Command file(s) or message catalog(s) chosen by LANGNUM
FORMATS
NL intrinsics (e.g., NLFMTDATE)
LANGNUM
LANGNUM
LANGNUM
LANGNUM

**Figure 1-4. Multilingual Application**

## HP Subsystem Utility Program

Figure 1-5 shows a special category of a multilingual application, the Hewlett-Packard subsystem utility program. Many of these programs are not typically used by end users, but are used to manipulate user data in conjunction with application programs. They determine which language to use at run time via a user-entered keyword or command, or defaults.

The user interaction in these programs has not been made localizable since many of these programs are not end user tools.



DATA MANIPULATION
USER INTERACTION
DATA BASE
IMAGE data base(s)
INDEXED SEQUENTIAL
KSAM file(s)
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
HP 3000 SUBSYSTEMS
FCOPY, SORT, MERGE, QUERY, ENTRY Determine LANGNUM from user command or keyword. *
* Call NLGETLANG to establish default(s).
LANGNUM
LANGNUM
LANGNUM
LANGNUM
SCREENS
VPLUS forms file(s) w/appropriate language or "international"
PROMPTS, MESSAGES
Hard-coded or in message catalog (not localized)
USER COMMANDS
Hard-coded
FORMATS
NL intrinsics (e.g., NLFMTDATE)

Figure 1-5. HP Subsystem Utility Program

# Application Message Facility 2

The Application Message Facility is a Native Language Support (NLS) tool that provides a programmer with the flexibility needed to create application catalogs for localization. Text such as prompts, commands, and messages intended for the user's interaction with an application can be stored in separate ASCII editor files. This allows the programmer to maintain files and localize applications without changing the program code.

The NLS Application Message Facility contains the GENCAT utility program and the CAT intrinsics, CATOPEN, CATREAD, and CATCLOSE, as shown in Figure 2-1.

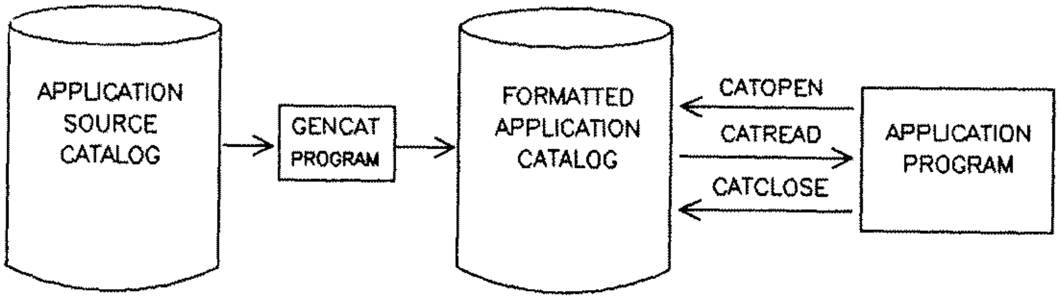


**Figure 2-1. GENCAT Utility Program**

The GENCAT utility creates and maintains message catalogs which meet the NLS requirements for efficient storage and retrieval of messages. For a comparison of GENCAT and MAKECAT, an MPE utility which is also used to create and maintain message catalogs, refer to Table 2-2 at the end of this chapter.

## Accessing Application Catalogs

Catalogs formatted with GENCAT can be accessed by applications via the CAT intrinsics:

CATOPEN Opens a catalog for access by an application.
CATREAD Retrieves text from a catalog.
CATCLOSE Closes a catalog.

These intrinsics are documented in Chapter 4, "Native Language Intrinsics." Refer to Program L in Appendix H for an example of their use.

The NLAPPEND intrinsic can be called to concatenate the language ID number and the catalog filename before the catalog is opened. Refer to "Catalog Naming Convention" in this section for more information.

## Source Catalogs

First, the user creates an MPE ASCII file in an editor with an EDIT/3000 compatible format. The catalog may contain 8-bit characters. The GENCAT program reads the source catalog and creates a binary formatted catalog which can be accessed by application programs. Calls to the CAT intrinsics access the formatted catalogs. An internal directory, which expedites accessing the formatted catalog, is created in the catalog. The text in the formatted catalog is compressed for efficient storage. The source catalog's record size may vary from 20 words to 128 words. Often a message is split over several records.

Figure 2-2 illustrates the three functions GENCAT performs on an application message catalog: modifying, formatting, and expanding.

## Directives

A source catalog contains directives which partition information in the message catalog. The three types of directives include $ to denote a comment line, $SET to mark the beginning of a new set of messages, and message numbers to indicate messages.

### $SET Records

A $SET record initiates a logical grouping of messages. Sets break the catalog into manageable segments containing logical groupings of messages (for example, one set of messages for prompts, one set for instructions, one set for error messages).

The format of a $SET record, where *xxx* is a required number for that set of messages (ranging from 1 to 255) is:

```
$SET xxx [comment] or $set xxx [comment].
```

A $SET record can contain comment as an optional character string. If there is not at least one blank between *xxx* and the comment, GENCAT will issue an error message and terminate the formatting.

Set records must begin in column 1. For example, to indicate that set number 1 is being defined:

```
$SET 1 Set one contains all prompts.
```

See Figure 2-3 for an example of a $SET record.

Figure 2.3
Example of $set rec.
was not given
in documentation

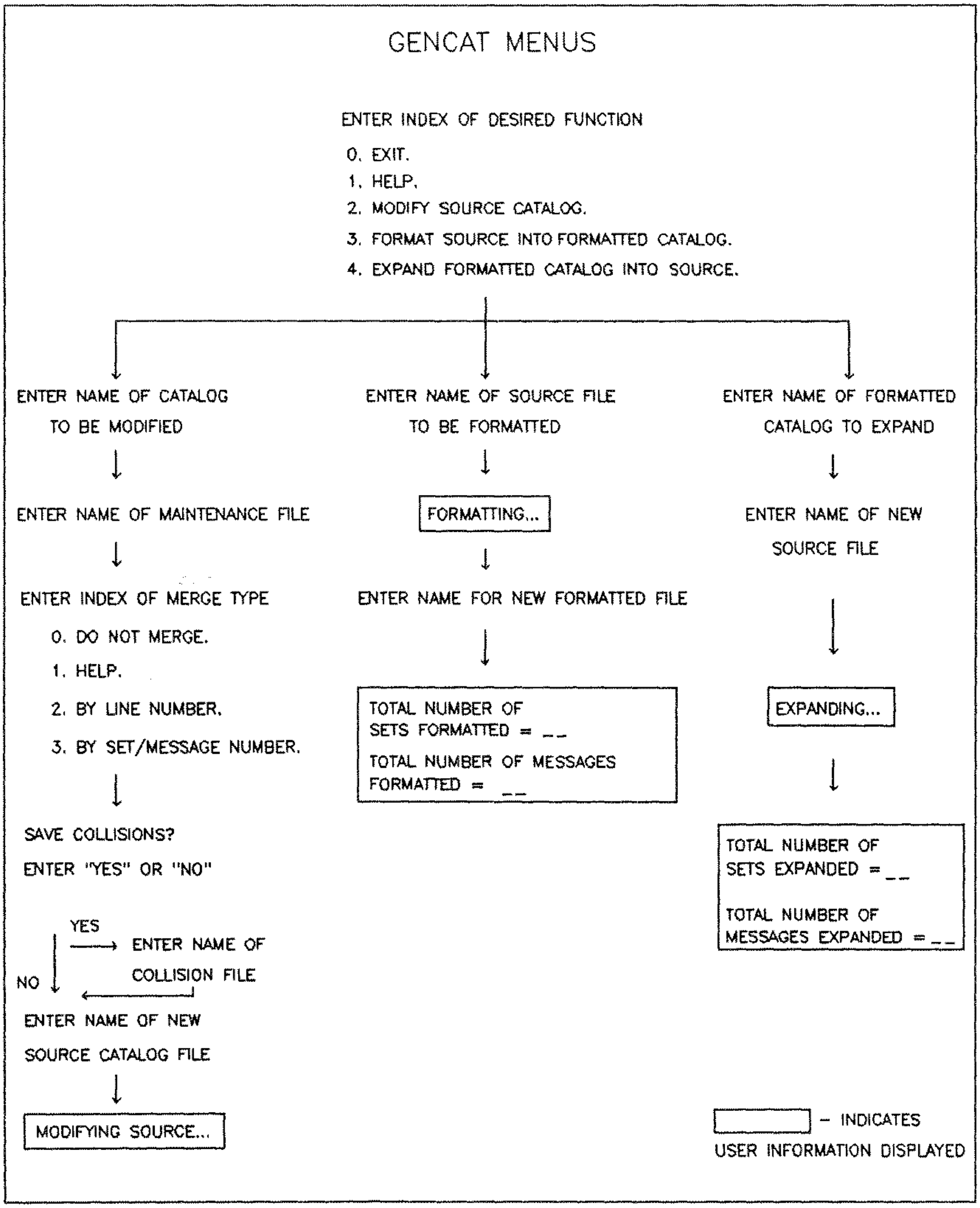

GENCAT MENUS
ENTER INDEX OF DESIRED FUNCTION
0. EXIT.
1. HELP.
2. MODIFY SOURCE CATALOG.
3. FORMAT SOURCE INTO FORMATTED CATALOG.
4. EXPAND FORMATTED CATALOG INTO SOURCE.
ENTER NAME OF CATALOG TO BE MODIFIED
ENTER NAME OF MAINTENANCE FILE
ENTER INDEX OF MERGE TYPE
0. DO NOT MERGE.
1. HELP.
2. BY LINE NUMBER.
3. BY SET/MESSAGE NUMBER.
SAVE COLLISIONS?
ENTER "YES" OR "NO"
YES
ENTER NAME OF COLLISION FILE
NO
ENTER NAME OF NEW SOURCE CATALOG FILE
MODIFYING SOURCE...
ENTER NAME OF SOURCE FILE TO BE FORMATTED
FORMATTING...
ENTER NAME FOR NEW FORMATTED FILE
TOTAL NUMBER OF SETS FORMATTED = __
TOTAL NUMBER OF MESSAGES FORMATTED = __
ENTER NAME OF FORMATTED CATALOG TO EXPAND
ENTER NAME OF NEW SOURCE FILE
EXPANDING...
TOTAL NUMBER OF SETS EXPANDED = __
TOTAL NUMBER OF MESSAGES EXPANDED = __
– INDICATES USER INFORMATION DISPLAYED

**Figure 2-2. GENCAT Functions**

## $LANG Records

A $LANG record specifies the language of the message that follows. It is used primarily with 16-bit languages to tell GENCAT that the messages will be in two-byte character formats. $LANG is not required for 8-bit languages.

The format of a $LANG record, where *xxx* is a valid *langnum*, is:

```
$LANG xxx[comment] or $lang xxx[comment]
```

A $LANG record can contain comments as an optional character string. If there is not at least one blank between *xxx* and a comment, GENCAT will issue an error message and terminate the formatting.

$LANG records must begin in column 1. For example, to indicate the message catalog contains characters in Simplified Chinese, the user will indicate:

```
$LANG 201 Simplified Chinese Language
$SET 1
1 This message is in Simplified Chinese.
2 This message is in USASCII.
3 This message is a mix of Chinese and USASCII.
```

## Message Records

Message records consist of a message number followed by the message text. This may be an error message, prompt, or any text which may change with the language or country where the program will be used. Message records:

- Identify message locations within a set.
- Must be in ascending sequence and unique within the set that contains them.
- Do not need to be consecutive.

For example, within a set, one can have messages 1-25, 101, 300-332, and 32766. All of these message numbers can be used again in another set. The format for a message record where *xxxxx*, an integer, is the required message number is:

```
xxxxx [the text of the message].
```

Text is an optional character string which, if present, follows the message number. If the text is not preceded by a blank, GENCAT will replace the character immediately following the message number with a blank. The user will be informed that a blank has replaced the character. An exception is made if one of two special characters, "%" or "&," follow the message number. These characters will not be replaced by a blank. Their meaning is explained in the following section.

## Message Record Special Characters

When CATREAD is writing a message to a file, the percent (%) instructs CATREAD to post a carriage return-line feed before writing the next record. For example, a message in set 4:

```
3 AN ERROR OCCURRED DURING THE LOADING %
OF THE DATA BASE.
```

The execution of CATREAD `(catindex,4,3);` results in a display of:

```
AN ERROR OCCURRED DURING THE LOADING
OF THE DATA BASE.
```

The ampersand (&) indicates that the statement is continued on the next line. Message 98 in set 67 is:

```
98 THE NUMBER OF FILES &
DOES NOT MATCH THE &
SYSTEM'S CALCULATIONS.
```

The execution of CATREAD `(catindex,67,98, ...);` results in a display of:

```
THE NUMBER OF FILES DOES NOT MATCH THE SYSTEM'S CALCULATIONS.
```

Note the use of blanks as separators preceding the ampersand. Message records must begin in column 1 and may have leading zeros. For example, the format of message number 3 in some set is:

```
0003 PLEASE ENTER YOUR NAME.
```

The tilde (~) is used as a literal character. It instructs CATREAD to treat the character which follows it as a literal part of the message (even if it is a special character). For example, two tildes in a row will put one tilde into the message.

The exclamation mark (!) is discussed in "Parameter Substitution" in this section.

## Comment Records

Comments are used throughout the catalog to document sets and messages, and to make them easier to read. The format of a comment record, where comment is an optional string of characters is:

*$* [*comment*] .

A blank between *$* and [*comment*] is necessary only when the comment is a $SET or $DELSET record.

## Sample Source Catalog

Notice the directives $, ($SET numbers), message numbers, message comments, and the use of blanks in the sample source catalog:

```
$ This catalog is for development only.  Messages will be
$ added as needed.
$**
$SET 1  Prompts
1 ENTER FIRST NAME
2 ENTER LAST NAME
$
$**
$LANG 0 ASCII (NATIVE-3000)
$SET 2  Error messages
1 NAME NOT ON DATA BASE
2 ILLEGAL INPUT
95 OPERATION IS %
INCONSISTENT WITH ACCESS TYPE
$
$CHANGE THE LANGUAGE TO JAPANESE
$LANG 221
100 JAPANESE MESSAGE
$LANG 0 SET LANGUAGE TO ASCII (NATIVE-3000)
```

# Parameter Substitution

Parameter substitution can often be used with messages. An exclamation mark (!) is used within a message to indicate where a parameter is to be inserted using CATREAD. The user must choose positional or numerical parameter substitution. Mixing these two types within a message is not allowed.

## Positional Parameter Substitution

Positional parameter substitution simply means that each of the parameters in the CATREAD parameter list is to be inserted into the message at each successive "!". A maximum of 5 parameter substitutions is allowed in one message. The following example is used to illustrate the use of positional parameter substitution:

```
SPL STATEMENT

    CATREAD (catindex, 13, 400, error,,,user, term);

PARAMETERS

    BYTE ARRAY user (0:8):="MARY.KSE", 0;
    BYTE ARRAY term (0:5):="THREE", 0;
```

Message 400 in set 13 is:

```
400 ILLEGAL INPUT FROM USER ! ON TERMINAL NUMBER !
```

The execution of the SPL statement in Figure 2-4, with the parameters given, results in the following message:

```
ILLEGAL INPUT FROM USER MARY.KSE ON TERMINAL THREE.
```

## Numerical Parameter Substitution

Numerical parameters allow the user to decide where the parameters are to be placed within the message. The exclamation mark (!) is immediately followed by a number in the range 1-5. The following example is used to illustrate the use of numerical parameter substitution:

```
SPL STATEMENT

    CATREAD (catindex, 7, 4, error,,,fourstr, fivestr)

PARAMETERS

    BYTE ARRAY fourstr (0:4):="FOUR", 0;
    BYTE ARRAY fivestr (0:4):="FIVE", 0;
```

A message in set 7 is:

```
4 EOF DETECTED AFTER RECORD !1 IN FILE !2
```

The execution of the SPL statement in Figure 2-5, with the parameters given, results in the following message:

```
EOF DETECTED AFTER RECORD FOUR IN FILE FIVE.
```

Message 5 in set 7 is:

```
5 EOF DETECTED AFTER RECORD !2 IN FILE !1
```

A change in the call results in a different message:

```
CATREAD (catindex, 7, 5, error,,,fourstr, fivestr)
```

Message:

```
EOF DETECTED AFTER RECORD FIVE IN FILE FOUR.
```

Mixing numerical and positional parameter substitution characters is not allowed and will be flagged as an error:

```
EOF DETECTED AFTER RECORD ! IN FILE !1.
```

Numeric parameter substitution can be used only with GENCAT and the CATREAD intrinsic. CATREAD interprets the character tilde (~) as a literal character. If a character is preceded by a tilde (~), that character is taken literally. For example, if set 7 also contains the following message:

```
6 ERROR ! IN INPUT !
```

When the SPL statement, `CATREAD (catindex,7,6,error,,,seventeen)`, is executed, the resulting output is:

```
ERROR 17 IN INPUT!
```

The second exclamation mark would not be used for parameter substitution because it is preceded by a tilde (~).

## Catalog Naming Convention

Catalogs are MPE files accessed by application programs via the CAT intrinsics. An application that has been localized into more than one language will typically have a separate message catalog for each language. A naming convention facilitates using different localized versions of files required by an application program.

A catalog filename can be identified with a maximum of five characters. Each native language supported by NLS has a language ID number (*langnum*). A three-digit language ID number can be appended to the catalog filename to identify each localized catalog.

For example, an original unlocalized message catalog is APCAT000. The message catalog in German would be APCAT008. A Spanish version would be APCAT012. Refer to Appendix B, "Supported Languages and Character Sets," for a complete list of native languages and their corresponding language ID numbers. When the language ID number has been selected, the NLAPPEND intrinsic may be used to form the catalog filename. At run time the application program is responsible for determining which catalog to open with the CATOPEN intrinsic.

## Maintaining a Message Catalog

Maintenance functions can include addition, deletion, and modification of records in the source file. The input for merging consists of two files, the source file and the maintenance file. The maintenance file is merged against the source file, either by line numbers or by $SET and message numbers. If the user does not know the line numbers, the $SET and message numbers can be used successfully. The context of the $SET and message records in the maintenance file determines the type of maintenance performed on the source. Changes made to a source during a maintenance merge may be kept in a collision filenamed by the user. Collision files are created at the option of the user. Figure 2-3 illustrates how the collision file may be merged against the modified source catalog to recreate the original source.

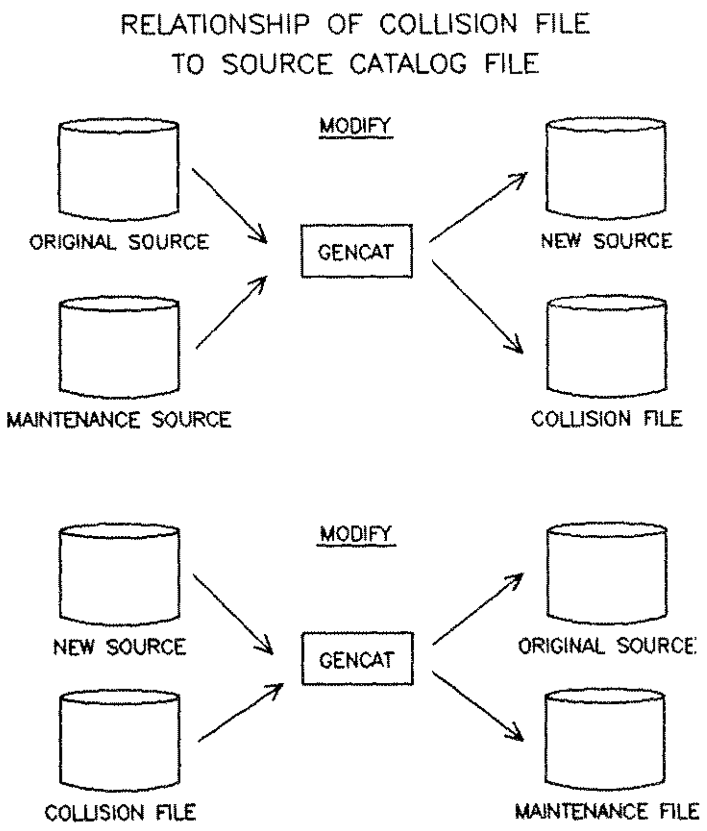


Figure 2-3. Collision Files

## Merging Maintenance Files by Line Numbers

Merging a maintenance file against a source catalog file by line numbers may include modifying, adding, or deleting records.

### Modifying a Record

If the maintenance file's line number is common to the source file's, the source's record is overwritten by the maintenance record.

### Adding a Record

If the line number in the maintenance file does not exist in the source, the record represented by that line number from the maintenance file is added to the source at that line number.

### Deleting a Record

The directives $EDIT and $EDIT VOID=*xxxxxxx* are used to delete records from the source file. If $EDIT VOID= is used, the records beginning with and including the record number of the $EDIT VOID= record to record *xxxxxxx* will be deleted (line number *xxxxxxx* represents the line number *xxxxx.xxx* of the source file).

## Merging Maintenance Files by $SET and Message Number

When GENCAT reads a $SET record from the maintenance file, all records following the $SET record are considered to be message records or comment records within that set until GENCAT reads another $SET record or exhausts the maintenance file. Set numbers must be in ascending order, and message numbers must be in ascending order within each set.

The first record GENCAT expects to read, from the maintenance file, is a $SET, $DELSET, or a comment record. GENCAT will continue to read and evaluate the maintenance file records until an error is encountered or the maintenance file is exhausted. After GENCAT reads a maintenance file record, it is evaluated according to a set of rules, and a copy of the source is modified as necessary. The following rules for evaluation apply to set numbers, message numbers, comment records, and the $DELSET directive.

### Set Numbers

New set numbers are added to the source catalog file. All message numbers and messages following the set record are assumed to be new and will be added to the source file.

Set numbers, if already present, signify changes to the set of messages currently in the source catalog. All message numbers and messages following this set are to be evaluated according to the rules for message numbers.

Set numbers in a $DELSET record indicates that the entire set of messages in the source is to be deleted.

### Message Numbers

New message numbers within a $SET are added to the new source. Message numbers that are already present are deleted if no text follows the message number. If new text is supplied, the existing message will be updated.

### Comment Records

Comment records are written to the new source file or maintenance file as they are encountered.

### The $DELSET Directive

The $DELSET directive is allowed only in the maintenance file. It instructs GENCAT to delete the entire set of messages denoted by *xxx*. Optional text may follow *xxx*, providing it is preceded by at least one blank. The $DELSET directive is not written to the new file.

$DELSET records must begin in column 1. The format of a $DELSET record, where *xxx* is an existing set number in the source catalog is:

$DELSET *xxx* [*text*]

The directives $SET and $DELSET may be either in uppercase or lowercase ($set and $delset). Mixed cases are not allowed (e.g., $Set or $deLseT).

When one of the directives is encountered at the beginning of the maintenance file, it supercedes the corresponding directive (if any) in the master file.

## User Dialog

The user may modify a source file, format a source catalog, or expand a formatted catalog as shown in the following dialog. Figure 2-4 illustrates the process of maintaining a GENCAT source file.

```
:RUN GENCAT.PUB.SYS

HP32414A.00.00 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION

0.  EXIT.
1.  HELP.
2.  MODIFY SOURCE CATALOG.
3.  FORMAT SOURCE INTO FORMATTED CATALOG.
4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>>2

ENTER NAME OF CATALOG SOURCE FILE TO BE MODIFIED

>>APCAT000

ENTER NAME OF MAINTENANCE FILE

>>CATMANNT
```

If the name of a nonexistent file is entered, an error message is displayed.

```
NONEXISTENT PERMANENT FILE (FSERR 52)

EXPECTED AN EXISTENT FILE AS INPUT (GCERR 15)

ENTER NAME OF MAINTENANCE FILE

>>CATMAINT

ENTER INDEX OF MERGE TYPE

0.  DO NOT MERGE.
1.  HELP.
2.  BY LINE NUMBER.
3.  BY SET/MESSAGE NUMBER.

>>3
```

Entering 0 or Return aborts the maintenance function and returns to the main menu.

The user has the option of saving all the modifications, from the merge, in a collision file:

```
SAVE COLLISIONS?  ENTER "YES" OR "NO"

>>YES

ENTER NAME OF COLLISION FILE

COLCAT
```

If the name of an existing file is entered, the prompt is repeated. A Return continues the merging without saving the collisions.

GENCAT merges the source and maintenance files into a temporary file, and will prompt for the name of a permanent file:

```
ENTER NAME OF NEW SOURCE CATALOG FILE

>>NEWCAT
```

This prompt is repeated until a unique filename or a Return is entered. The temporary file is copied to the new permanent file. If a Return is entered the merging is aborted.

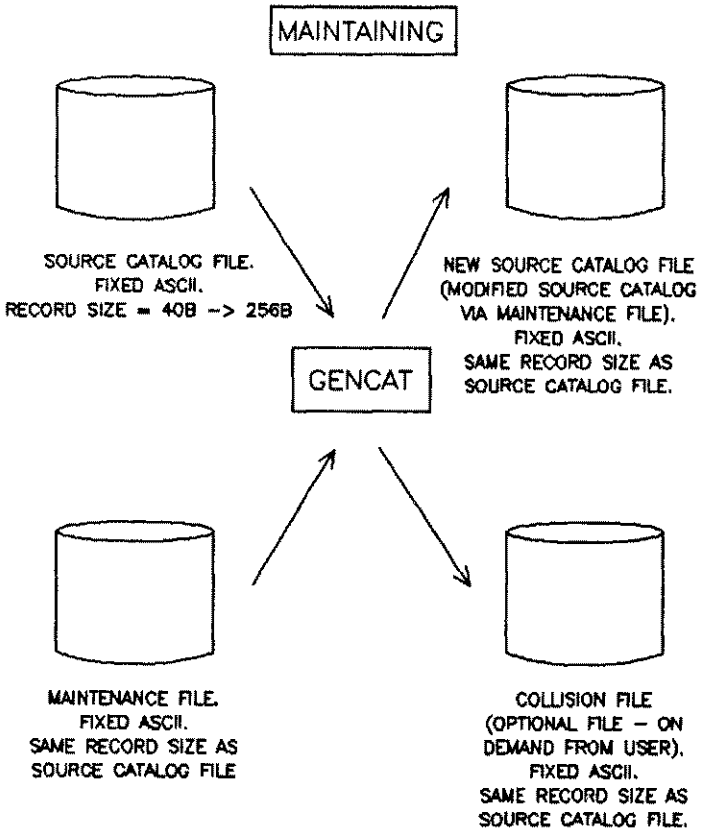


**Figure 2-4. Maintaining a GENCAT Source File**

## Formatting a Source Catalog

It is necessary to format the source catalogs so the CAT intrinsics can access them. GENCAT formatted files are binary and cannot be edited. Formatting compacts files and creates a directory, which saves disc space and reduces access time.

During the formatting process, GENCAT verifies that:

- All directives are legal and used correctly.
- Set numbers are in ascending order.
- Set numbers are greater than 0 and less than or equal to 255.
- Message numbers are in ascending order within each set.
- Message numbers are greater than 0 and less than or equal to 32766.
- Continuation and concatenation characters are correct.
- Parameter substitution characters are used correctly.

The following dialog is used for formatting a source catalog:

```
:RUN GENCAT.PUB.SYS

HP32414A.00.00 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION

0.  EXIT.
1.  HELP.
2.  MODIFY SOURCE CATALOG.
3.  FORMAT SOURCE INTO FORMATTED CATALOG.
4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>> 3

ENTER NAME OF SOURCE FILE TO BE FORMATTED

>>NEWCAT

FORMATTING...

ENTER NAME FOR NEW FORMATTED FILE

>>FORMCAT

TOTAL NUMBER OF SETS FORMATTED = 6
TOTAL NUMBER OF MESSAGES FORMATTED = 167

FORMATTING SUCCESSFUL
```

## Expanding a Formatted Catalog

GENCAT contains a function to recreate the original source catalog file by expanding the formatted catalog. The result is a new source catalog that can be edited and then converted to a formatted catalog. Figure 2-5 is an example of the user dialog for expanding a formatted catalog. The following dialog is used for expanding a formatted catalog:

```
:RUN GENCAT.PUB.SYS

HP32414A.00.00 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION

0.  EXIT.
1.  HELP.
2.  MODIFY SOURCE CATALOG.
3.  FORMAT SOURCE INTO FORMATTED CATALOG.
4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>> 4

ENTER NAME OF FORMATTED CATALOG TO EXPAND

>>FORMCAT

ENTER NAME OF NEW SOURCE FILE

>>NCATSOUR

EXPANDING...

TOTAL NUMBER OF SETS EXPANDED = 6
TOTAL NUMBER OF MESSAGES EXPANDED = 167

EXPANSION SUCCESSFULLY COMPLETED
```

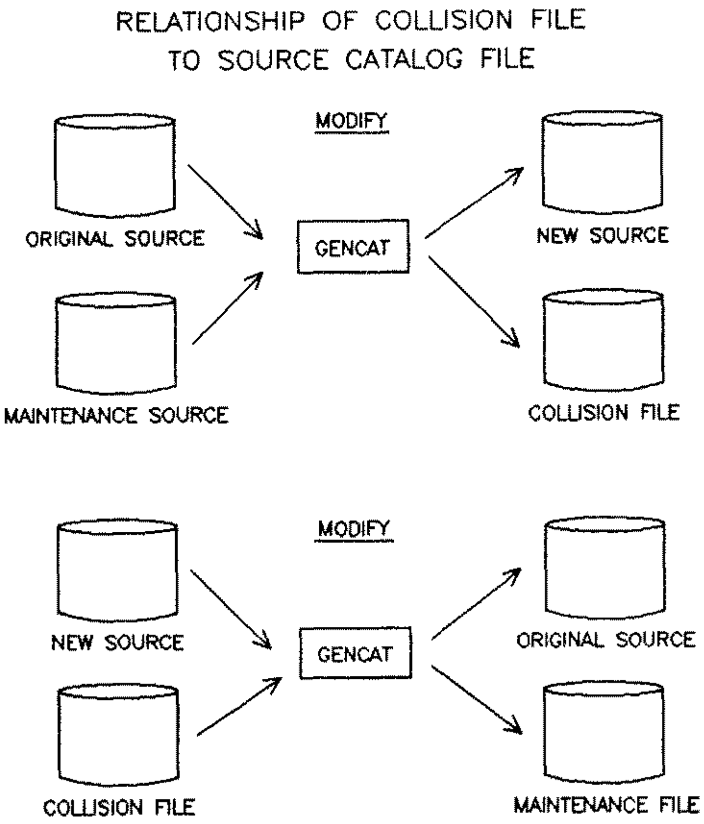


**Figure 2-5. Formatting/Expanding GENCAT Source Files**

## GENCAT JCWs

GENCAT uses three Job Control Words (GCMAINT, GCFORMAT, and GCEXPAND) to indicate the status of the function performed. GENCAT initializes all three JCWs to zero upon entry and sets GC-MAINT, GCFORMAT, or GCEXPAND at the end of a maintenance, formatting, or expanding function, respectively. If the function succeeds, the appropriate GENCAT JCW remains set to zero. If the function fails, the appropriate JCW is set to the GENCAT error number describing the failure. For example, if a formatting function fails with error number 10 (GCERR 10), GCFORMAT is set to 10. If the process completes unsuccessfully, the system JCW is set to FATAL; the status of the GENCAT JCW is not important.

## GENCAT in Batch Mode

GENCAT can be invoked interactively or in batch mode. GENCAT will abort a job in batch mode if an error is encountered while formatting, expanding, or modifying.

## GENCAT Help Facility

With the GENCAT online HELP facility, the user can enter the index number for HELP from the menu or a "?" in response to any prompt that does not have a menu selection for HELP. The following is an example of the GENCAT HELP Facility dialog:

```
:RUN GENCAT.PUB.SYS

HP32414A.00.00 GENCAT/3000 (C) HEWLETT-PACKARD., 1983

ENTER INDEX OF DESIRED FUNCTION

0.  EXIT.
1.  HELP.
2.  MODIFY SOURCE CATALOG.
3.  FORMAT SOURCE INTO FORMATTED CATALOG.
4.  EXPAND FORMATTED CATALOG INTO SOURCE.

>>1
```

This is the driver menu for GENCAT.

Input consists of a numeric index, 0 through 4. Each index denotes a function for GENCAT to perform.

```
0 - Will exit GENCAT and return you to MPE.
1 - Will display this message.
2 - Will direct GENCAT to begin the maintenance function.
3 - Will direct GENCAT to begin the formatting function.
4 - Will direct GENCAT to begin the expansion function.
```

For each prompt, an input of an index for HELP or a "?" (depending upon the type of prompt) will display instruction for that prompt.

Formatting is the creating of an internal representation of a source message catalog into a form used by the CAT*xxxx* intrinsics. Maintenance is modifying the source message catalog by merging a maintenance file against it. The merge may be by line numbers set and message numbers. Expansion is converting the formatted file back into a source message catalog.

Pressing Return exits GENCAT and returns to MPE.

## Error Messages

GENCAT error messages are listed in Table 2-1.

Table 2-1. GENCAT Error Messages

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 1 | FREAD ERROR ON SOURCE FILE. | A failure by FREAD when reading a source message catalog. | Recreate the source message catalog. |
| 2 | INPUT FILE MUST HAVE AT LEAST ONE RECORD. | The file has an EOF of zero (0). | Place at least one record in the file. |
| 3 | INPUT FILE MUST CONTAIN FIXED LENGTH RECORDS ONLY. | File does not have a fixed record length. | Create the file with a fixed record length. |
| 4 | INPUT FILE MUST BE USASCII FILE ONLY. | Source and maintenance files must have records that are in USASCII format. | Create the source and maintenance files with USASCII format. |
| 5 | INPUT FILE RECORD SIZE MUST BE BETWEEN 40 AND 256 BYTES. | The record size of a source or maintenance file is greater than 256 bytes (128 words) or less than 40 bytes (20 words). | Create a source and maintenance file with a record size greater or equal to 40 bytes or less than or equal to 256 bytes. The record length includes any line numbers in the file. |
| 6 | SET NUMBERS MUST BE BETWEEN 1 AND 255. | A set number in a maintenance or source file is not greater than or equal to 1, or not less than or equal to 255. The set number may not be positive or numeric. | Change set number to a value between 1 and 255 inclusive. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 8 | SET NUMBERS MUST BE IN ASCENDING SEQUENCE. | A set number is less than or equal to the previous set number in the source file. Error can be detected at format time or during a maintenance function. | Change numbers to strict ascending sequence. |
| 9 | MESSAGE NUMBERS MUST BE BETWEEN 1 AND 32766. | A message number value is not between 1 and 32766 inclusive. | Change the message number value to a value between 1 and 32766 inclusive. |
| 10 | MESSAGES MUST EITHER CONTAIN ALL NUMBERED OR ALL POSITIONAL PARAMETER SUBSTITUTION CHARACTERS. MIXES NOT ALLOWED. | GENCAT detected a mix of parameter substitution characters during the message scan. For example, a message contained numeric substitution characters as well as positional substitution characters. | Change the parameter substitution characters either to all numeric or all positional substitution characters (for each message only). |
| 11 | MESSAGE NUMBERS MUST BE IN ASCENDING SEQUENCE. | A message number was processed that is less than or equal to the previous message number. The message numbers within a set are not in ascending sequence. | Rearrange the messages, within the set, to strict ascending order. |
| 12 | MESSAGE CONTAINS NON-BLANK CHARACTER IMMEDIATELY FOLLOWING MESSAGE NUMBER. NON-BLANK CHARACTER ASSUMED TO BE A BLANK. | GENCAT detected a non-blank character immediately following the message number in a message. GENCAT replaces this character with a blank. | Insert a blank between the message number and the message text. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 13 | EXPECTED ONE OF THE FOLLOWING INPUTS:<br>0, 1, 2, 3, 4, OR RETURN. | GENCAT detected an incorrect input in response to the menu (prompts for a function). | Respond with 0, 1, 2, 3, 4, or Return only. |
| 14 | EXPECTED ONE OF THE FOLLOWING INPUTS:<br>0, 1, 2, 3, OR A RETURN. | GENCAT detected an incorrect input in response to the menu (prompts for the type of merging it is to perform). | Respond with 0, 1, 2, 3, or Return only. |
| 15 | EXPECTED AN EXISTENT FILE AS INPUT. | The file does not exist on the system. | Either create the file or input the name of a file that exists on the system. |
| 16 | EXPECTED A UNIQUE, NON-EXISTENT FILE NAME AS INPUT. | The file already exists on the system. The name of the file should be one that does not exist on the system. | Purge the file or input the name of a file that does not exist on the system. |
| 17 | EXPECTED A RESPONSE OF "YES" OR "NO" AS INPUT. | GENCAT requires a response of either YES, yes, NO, or no to the prompt of "SAVE COLLISIONS?" Enter YES or NO. | Respond with YES, yes, NO, or no. |
| 18 | INPUT FILES MUST HAVE EQUAL RECORD SIZES FOR THIS FUNCTION. | Source and maintenance files must have equal record sizes if the maintenance file is to modify the source file. | Create a maintenance file that has a record size equal to the record size of the source file. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 20 | THE CONSTRUCT OF $DELSET IS NOT ALLOWED IN THE SOURCE. | The construct $DELSET, which may be used in a maintenance file, was detected in a source file during a maintenance function. | Remove $DELSET construct from the source file. |
| 21 | ONLY FIVE (5) POSITIONAL PARAMETER SUBSTITUTIONS ALLOWED PER MESSAGE. | More than five (5) parameter substitution characters were detected in one message. Up to five parameter substitution characters are allowed per message. | Fewer than or equal to 5 parameter substitution characters per message only are allowed. |
| 22 | MAINTENANCE FILE MUST BE NUMBERED FOR LINE-NUMBER MERGES. | The maintenance file is an unnumbered file. The maintenance file must be a numbered file if it is to be used in a line-number merge. | Number the maintenance file if the file is to be used in a line-number merge. |
| 23 | SOURCE FILE MUST BE NUMBERED FOR LINE-NUMBER MERGES. | The source file is an unnumbered file. The source file must be a numbered file if it is to be used in a line-number merge. | Number the source file if the file is to be used in a line-number merge. |
| 24 | SOURCE FILE CANNOT CONTAIN FORMS OF $EDIT. | The source file was examined for $EDIT and $EDIT VOID= constructs. These are not allowed (for example, if collision files are used, an ambiguity would exist if the $EDIT and $EDIT VOID= were left in the source file). | Remove all occurrences of $EDIT and $EDIT VOID= from the source file. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 25 | SEQUENCE NUMBER IN $EDIT VOID RECORD CONTAINS TOO MANY DIGITS. EIGHT IS THE MAXIMUM. | The value following the $EDIT VOID= may have a maximum of eight place holders. | Reevaluate the value and correct it, it must represent a line number. |
| 26 | FILE IS NOT A FORMATTED FILE. | Formatted catalogs only can be expanded (for example, files formatted by GENCAT). | Format the file using GENCAT. |
| 27 | SET RECORD IS REQUIRED BEFORE A MESSAGE RECORD IS FORMATTED. | A message was found before set number was defined. | Place the message in a set or place a set number before the message. |
| 28 | VALUE IN RIGHT BYTE OF KANJI CHARACTER IS INVALID. | The message contains special escape sequences provided by Hewlett-Packard that are used for research and development activities. These special escape sequences are not supported and Hewlett-Packard assumes no responsibility for their use. | Consult your Hewlett-Packard representative, or remove all occurrences of the form esc$<terminator> or ESC(<terminator> from the message catalog. Where ESC is the escape character and <terminator> is a or A - Z. |
| 29 | SCAN COMPLETED WITH NO CLOSING KANJI ESCAPE SEQUENCE. EXPECTS A CLOSING KANJI ESCAPE SEQUENCE TO TERMINATE KANJI CHARACTER SEQUENCE. | The message contains special escape sequences provided by Hewlett-Packard that are used for research and development activities. These special escape sequences are not supported and Hewlett-Packard assumes no responsibility for their use. | Consult your Hewlett-Packard representative, or remove all occurrences of the form esc$<terminator> or ESC(<terminator> from the message catalog. Where ESC is the escape character and <terminator> is a or A - Z. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 30 | INCOMPLETE KANJI CLOSING ESCAPE SEQUENCE DETECTED. | The message contains special escape sequences provided by Hewlett-Packard that are used for research and development activities. These special escape sequences are not supported and Hewlett-Packard assumes no responsibility for their use. | Consult your Hewlett-Packard representative, or remove all occurrences of the form esc$<terminator> or ESC(<terminator> from the message catalog. Where ESC is the escape character and <terminator> is @ or A - Z. |
| 31 | VALUE IN LEFT-BYTE OF KANJI CHARACTER IS INVALID. | The message contains special escape sequences provided by Hewlett-Packard that are used for research and development activities. These special escape sequences are not supported and Hewlett-Packard assumes no responsibility for their use. | Consult your Hewlett-Packard representative, or remove all occurrences of the form esc$<terminator> or ESC(<terminator> from the message catalog. Where ESC is the escape character and <terminator> is @ or A - Z. |
| 32 | VALUE IN PARAMETER SECTION OF KANJI ESCAPE SEQUENCE IS INVALID. EXPECTED A STRING OF DIGITS. | The message contains special escape sequences provided by Hewlett-Packard that are used for research and development activities. These special escape sequences are not supported and Hewlett-Packard assumes no responsibility for their use. | Consult your Hewlett-Packard representative, or remove all occurrences of the form esc$<terminator> or ESC(<terminator> from the message catalog. Where ESC is the escape character and <terminator> is @ or A - Z. |
| 33 | BLANK RECORDS THAT ARE NOT CONTINUATION RECORDS ARE NOT ALLOWED. | A blank record was detected in the source catalog and it is a continuation record for the previous record. | Remove the record from the source file, or modify the record before it; end the record with a % or & character. |

Table 2-1. GENCAT Error Messages (cont.)

| ERROR # | MESSAGE | MEANING | ACTION |
|---|---|---|---|
| 34 | INTERNAL GENCAT FILE HAS BEEN EXHAUSTED.<br>THE FILE "DATAM" HAS BEEN EXHAUSTED. FOR AN IMMEDIATE SOLUTION JUST REDO YOUR FUNCTION AGAIN, THE PROGRAM INCREASED THE LIMITS FOR YOU. FOR STREAM JOBS USE THE FILE EQUATION BELOW:<br>(see note below)*<br>RUN THE GENCAT PROGRAM. (INCREASE THE FILE SIZE UNTIL YOU GET RID OF THE PROBLEM.)<br>PLEASE INFORM HEWLETT-PACKARD OF THIS PROBLEM. | The file DATAM, used by GENCAT internally, is full. | Return to the main menu and redo the formatting function without exiting the program. |
| 35 | $LANG COMMAND SPECIFIED A LANGUAGE NOT CURRENTLY CONFIGURED. | The language requested is not configured in the system. | If you are not using Asian text, remove the $LANG record. If you are using an Asian language, request your System Manager to install the language in the system. |

* The file equation for error #34 above is:

```
:FILE DATAM=DATAM;REC=-256,32,F,ASCII;DISC=20000,32,32;BUF=4;TEMP
```

Table 2-2. MAKECAT/GENCAT Comparison

| FEATURES | MAKECAT | GENCAT |
|---|---|---|
| Access Methods | FOPEN, GENMESSAGE, and FCLOSE intrinsics open, access, and close formatted MAKECAT catalogs. | CATOPEN, CATREAD, and CATCLOSE intrinsics open, access, and close formatted GENCAT catalogs. |
| Formatting | Places an internal directory in the file's user labels. The file is formatted in place without creating a new file. | A source message file is formatted into another file, leaving the original source intact. The application uses the formatted file. The original source file can be purged. The formatted file can be expanded to restore the original source file. |
| Function | Converts or formats HELP and message files into catalogs. Installs system message catalog, using the BUILD entry point. | Formats application message catalogs. Provides a maintenance facility to modify existing source catalogs and the capability of expanding a formatted file into the original source file. |
| Input | The name of a file must be entered in a file equation. :FILE INPUT=<*your file*>. | GENCAT prompts the user for the name of a file. |
| Literal Character | Not supported. | The tilde (~) serves as a literal character, causing the character which immediately follows it to be treated as text. |
| Messages | The message number range per set is 1-255. | The message number range per set is 1-32766. |
| Numerical Parameters | Not supported. | Up to 5 numerical parameters can be contained in a message. |
| Output | Saves the formatted file as a temporary file with the name CATALOG. | Prompts the user for the name of the formatted file. The file is saved as a permanent file. |
| Processing | Formats more quickly than GENCAT. | Verifies each message for correct parameter substitution characters. Manipulates two temporary files while formatting the source file. |

Table 2-2. MAKECAT/GENCAT Comparison (cont.)

| FEATURES | MAKECAT | GENCAT |
|---|---|---|
| Record Format | Accepts source files of any size, but the file it saves has a record size of 80 bytes. The system message catalog is fixed binary. An application catalog is fixed ASCII. | Accepts source catalog files with record sizes from 40 to 256 bytes. The formatted file has a record size of 128 words, and is fixed binary. When a formatted catalog is expanded into a source catalog, the new source catalog is fixed ASCII with a record size identical to the original source catalog.<br>When maintenance is being performed, both the source file and the maintenance file must be of equal lengths in fixed ASCII. The resulting source and collision files (if specified) will be fixed ASCII, and their record sizes will equal the record size of the original source file. |
| Sets | The set directive is $SET. The set number range for a catalog is 1-63. | The set directive can be $SET or $set. The set number range for a source catalog is 1-255. |
| User Interface | The user must know which entry points to use and when to use them. Files are input via file equations. Error messages require user interpretation. | Menu-driven, originating from a catalog. Each prompt has HELP text associated with it. Error messages are self-explanatory. |

# NLS in MPE Subsystems 3

Native Language Support (NLS) supplies the applications designer with the tools to support native language data and local custom formats. NLS provides support features in FCOPY/3000, IMAGE/3000, KSAM/3000, QUERY/3000, SORT-MERGE/3000, VPLUS/3000, and RAPID/3000. COBOLII access to native language collating sequences is included in the SORT-MERGE/3000 subsection discussion.

The emphasis of NLS in the subsystems is on providing the end-user, rather than the application designer, with local language data and formats. User interfaces (prompts, commands, and messages) of the subsystem utility programs, for example, FORMSPEC or DBUTIL, are not localized.

This reference material is intended to be used as addenda to the subsystems manuals. Refer to the SORT-MERGE/3000, KSAM/3000, FCOPY/3000, QUERY/3000, IMAGE/3000, VPLUS/3000, and RAPID/3000 manuals for complete documentation.

## FCOPY/3000

Native Language Support (NLS) features in FCOPY/3000 can be accessed by adding a LANG= parameter to the existing options:

```
:FCOPY FROM=A; TO=B; LANG=GERMAN; UPSHIFT
```

If the LANG= parameter is omitted, FCOPY/3000 obtains the current data language with NLGETLANG (mode 2) and functions as it did before the introduction of NLS.

### Options

The FCOPY/3000 options affected by language dependency are character printing, translating, upshifting, and updating KSAM/3000 files.

### CHAR Option

Character codes not represented by symbols are displayed as periods. The TO= file can be a line printer, a keyboard display terminal, or an intermediate disc file to be listed at a later time.

CHAR No LANG= The NATIVE-3000 processing scheme will be retained.

CHAR LANG= The character definition table associated with the language will be used. Characters of type 3 (undefined graphic character) and 5 (control code) as in NLINFO item 12, are replaced by periods. Refer to Chapter 4, "Native Language Intrinsics," for more information.

### Character Translate Options

These options translate data for ASCII-to-EBCDIC and EBCDIC-to-ASCII conversions.

EBCDICIN/ EBCDICOUT Input of the LANG= parameter will result in the translation table associated with the language being used.

For example, using an EBCDIC-to-ASCII conversion table, FCOPY/3000 converts data from GERMAN EBCDIC to ROMAN8:

```
>FROM=MYGEBCFL; TO= MYROM8FL; LANG=GERMAN; EBCDICIN
EOF FOUND IN FROMFILE AFTER RECORD 29

30 RECORDS PROCESSED *** 0 ERRORS
```

### NOTE

This option is not available for 16-bit languages.

### UPSHIFT Option

The UPSHIFT option converts lowercase alphabetic characters to their corresponding uppercase characters as part of the copying operation.

| | | |
|---|---|---|
| UPSHIFT | No LANG= | Any character belonging to USASCII or to one of the extensions will be upshifted as it would have been before the introduction of NLS. |
| UPSHIFT | LANG= | All characters will be upshifted according to the specified language upshift definition. |

### FCOPY/3000 and KSAM/3000 Files

To change the language of an existing file, a new KSAM/3000 file must be built with the new language attribute, and the old file copied into the new. If FCOPY/3000 copies an existing KSAM/3000 file to a new KSAM/3000 file the same language attribute is assigned to the new file. The LANG= option of FCOPY/3000 cannot be used to change the language of a KSAM/3000 file.

### Combined Use of Options

Using LANG= without another relevant option such as UPSHIFT or EBCDICIN usually results in a warning message:

```
<<966>> LANG OPTION NOT RELEVANT
```

The user can continue without affecting the outcome of the operation. The LANG= option is ignored. The following combinations are flagged as an error:

```
BCDICIN;LANG=xxx
BCDICOUT;LANG=xxx
EBCDIKIN;LANG=xxx
EBCDIKOUT;LANG=xxx
KANA;LANG=xxx
```

For example:

```
>FROM=DEUTSCH; TO=DANSK; LANG=GERMAN; EBCDICIN
*57*SYNTAX ERROR: ILLEGAL COMBINATION OF OPTIONS
0 RECORDS PROCESSED *** 1 ERROR
```

## Error Messages

Table 3-1 lists the error messages for FCOPY/3000.

Table 3-1. FCOPY/3000 Error Messages

| ERROR # | MESSAGE | CAUSE | ACTION |
|---|---|---|---|
| 960 | LANGUAGE NOT CONFIGURED. | The language requested is not configured on the system. | Verify spelling of language name. Ask the System Manager to configure the language on the system. |
| 961 | NLS NOT CONFIGURED. | No native languages are configured on the system. | Ask the System Manager to configure the native language on the system. |
| 966 | LANG OPTION NOT RELEVANT. | The LANG option is not relevant to the command last entered. | Check command for correct options. You are given the choice whether or not to continue the operation. |

## Performance Issues

The implementation of CHAR, UPSHIFT, and EBCDICIN/EBCDICOUT using NLS intrinsics and language definition tables requires additional time for the conversion process.

# IMAGE/3000

Native Language Support (NLS) in IMAGE enables the user to assign a language attribute to a database. This language attribute determines the collating sequence used to insert an entry with a sort item of type X or U in a sorted chain. It also determines the operation of comparisons for entry level DBLOCK calls. In order to use NLS with IMAGE/3000, this language attribute will have to be specified by the user either at schema processing time or through the SET command in DBUTIL.

## Utility Programs

NLS features in IMAGE/3000 can be requested in four utilities: DBSCHEMA, DBUTIL, DBUNLOAD, and DBLOAD.

### DBSCHEMA

The optional language attribute will be specified:

BEGIN DATA BASE *databasename* [,LANGUAGE:*language*];

The language name or ID number can be used for *language*. If no LANGUAGE is specified, the database will use NATIVE-3000 as a default.

The names of data items and data sets are restricted to certain USASCII characters. This allows schemas to be valid internationally, for all Hewlett-Packard 8-bit character sets. It also allows the sources of application programs which call IMAGE/3000 intrinsics to be entered from and displayed on all 8-bit and 7-bit (USASCII) terminals.

### DBUTIL

DBUTIL includes the SET, HELP, and SHOW commands:

SET: SET LANGUAGE= *language*. This command can be issued only on a virgin root file or an empty database (where <*language*> is the language name or language ID number).

HELP: HELP SHOW and HELP SET will display the syntax for SHOW and SET commands with the LANGUAGE option.

SHOW: SHOW *databasename* [/*maintword*] LANGUAGE. The language attribute of the database is displayed.

### DBUNLOAD/DBLOAD

DBUNLOAD copies the data to specially formatted tapes or disc volumes. The language ID number of the database is stored along with the data.

DBLOAD warns the user, who tries to load data, when the language attribute of the database on disc and the database on tape are incompatible:

```
THE LANGUAGE OF THE DATA BASE IS DIFFERENT
FROM THE LANGUAGE FOUND ON THE DBLOAD MEDIA.
```

If the user is running DBLOAD in a session, the user may choose to continue:

```
CONTINUE DBLOAD OPERATION ? (Y/N)
```

In case of a job execution of DBLOAD, or a negative answer (N) to the previous question, the DBLOAD operation is prematurely terminated.

## Intrinsics

The language attribute of the IMAGE/3000 database enables the IMAGE/3000 intrinsics to utilize native language features.

### DBOPEN

DBOPEN checks the language attribute of the database. When the language attribute of the database is not supported by the current configuration of the system, an error code of -200 is returned:

```
DATA BASE LANGUAGE NOT SYSTEM SUPPORTED.
```

### DBPUT

The position of a new entry with a type X or U item in a sorted chain is determined according to the collating sequence of the language attribute of the database.

If the database language attribute is NATIVE-3000, the insertion of a new entry in the sorted chain is determined by the result of a BYTE COMPARE between the key of the new record and the keys of the entries already in the chain.

If the database has a language attribute other than NATIVE-3000, the collating sequence definition of the native language is used via a system version of the NLCOLLATE intrinsic to determine where to insert the new entry.

## DBINFO

DBINFO provides additional information about the language attribute of the database:

| | |
|---|---|
| Mode: | 901 |
| Purpose: | Obtain language attribute of the database. |
| Qualifier: | Ignored |
| Buffer Array Contents: | Word 1 contains the language ID number. |

## DBLOCK

If a lock item is of type U or X, and a lock specifies an inequality (range), the collating sequence for the language of the database will be used.

## Changing The Language Attribute of an IMAGE/3000 Database

This change cannot be done with a single command. Once data has been stored in an IMAGE/3000 database with a native language attribute, changing the language attribute requires reorganizing data along any sorted chains according to the collating sequence of the new language.

The procedure is:

1. DBUNLOAD the database.
2. Purge the database using PURGE in DBUTIL.
3. Modify the schema with the language attribute set by the LANGUAGE: parameter and create a new root file with the schema processor.
4. Create the database using CREATE in DBUTIL.
5. Run DBLOAD in session mode. A warning message is issued because the language has been changed and a prompt is displayed:

```
CONTINUE DBLOAD OPERATION? (Y/N)
```

Enter Y to complete the change of the language attribute.

### NOTE

All IMAGE/3000 databases created before NLS are considered to have NATIVE-3000 as a language attribute.

## Error Messages

The three types of error messages used in IMAGE/3000 are listed in the following tables. Table 3-2 lists Utility Program Conditional Messages, Table 3-3 lists Library Procedure Calling Errors, and Table 3-4 lists Schema Syntax Errors.

Table 3-2. IMAGE/3000 Utility Program Conditional Messages

| MESSAGE | MEANING | ACTION |
|---|---|---|
| DATA BASE LANGUAGE NOT SYSTEM SUPPORTED. | The database language is not currently configured on your system. | Ask the System Manager to configure the native language on your system, or provide a valid language. |
| ERROR READING ROOT FILE RECORD. | DBUTIL is unable to read a root file record. | Contact your Hewlett-Packard support representative. |
| ERROR WRITING ROOT FILE RECORD. | DBUTIL has detected an error while writing a root file record. | Contact your Hewlett-Packard support representative. |
| INVALID LANGUAGE. | The language name or number contains invalid characters. | Retype the correct language name. |
| LANGUAGE MUST NOT BE LONGER THAN 16 CHARACTERS. | The language name is too long and must be incorrect. | Retype the correct language name. |
| LANGUAGE NOT SUPPORTED. | The language specified is either not supported on your system or is not a valid language name or ID number. | Contact the System Manager for configuration of that language or provide a valid language. |
| NLINFO FAILURE. | An error was returned by MPE NLS. | Contact your Hewlett-Packard support representative. |
| NLS RELATED ERROR. | An error was returned by MPE NLS on a DBOPEN on the database. | Contact your Hewlett-Packard support representative. |

Table 3-2. IMAGE/3000 Utility Program Conditional Messages (cont.)

| MESSAGE | MEANING | ACTION |
|---|---|---|
| THE LANGUAGE OF THE DATA BASE IS DIFFERENT FROM THE LANGUAGE FOUND ON THE DBLOAD MEDIA. | The user has changed the language attribute of the database between DBUNLOAD and DBLOAD. DBLOAD wants the user to be aware of potential differences in sorted chains in the collating sequence of the two languages (the language of the database on disc and tape are different). In session mode the question "CONTINUE DBLOAD OPERATION?" is asked. In job mode, DBLOAD will terminate execution. | After noting the information returned by DBLOAD, and the result on eventual sorted chains in the database, proceed with the operation by answering YES. |

Table 3-3. IMAGE/3000 Library Procedure Calling Errors

| CCL | CONDITION | MEANING | ACTION |
|---|---|---|---|
| -200 | DATA BASE LANGUAGE NOT SYSTEM SUPPORTED. | DBOPEN attempted to open the database and found that the language of the database is not currently configured. The collating sequence of the language is unavailable; DBOPEN cannot open the database. | Ask the System Manager to configure the language on your system. |
| -201 | NATIVE LANGUAGE SUPPORT NOT INSTALLED. | NLS internal structures have not been built at system startup. The collating sequence table in the language of the database is unavailable; DBOPEN cannot open the database. | Ask the System Manager to install NLS. |
| -202 | MPE NATIVE LANGUAGE SUPPORT ERROR #1 RETURNED BY NLINFO. | The error number given was returned by MPE NLS on a NLINFO call in DBOPEN. | Ask the System Manager to install NLS. |

Table 3-4. IMAGE/3000 Schema Syntax Errors

| MESSAGE | MEANING | ACTION |
|---|---|---|
| BAD LANGUAGE. | The language name contains an invalid character or language number is not a valid integer. | Examine schema to find incorrect statement, edit, and run Schema Processor again. |
| DATA BASE NAME TOO LONG. | The database name contains more than six characters. | Examine schema to find incorrect statement, edit, and run Schema Processor again. |
| LANGUAGE EXPECTED. | The schema processor expected, at this point, to find a LANGUAGE statement after the comma following BEGIN DATA BASE name statement. | Examine schema to find incorrect statement, edit, and run Schema Processor again. |
| LANGUAGE NOT SUPPORTED. | Language specified is not currently supported on your system or is not a valid language. | Examine schema to find incorrect statement, edit, and run Schema Processor again. |
| NATIVE LANGUAGE SUPPORT ERROR. | An error was returned by MPE NLS. | Contact your Hewlett-Packard support representative. |

## KSAM/3000

The Keyed Sequential Access Method (KSAM/3000) organizes records in a file according to the content of key fields within each record.

Native Language Support (NLS) in KSAM/3000 provides the resources to create files whose keys of type BYTE are sorted according to a native language collating sequence. All BYTE keys in the file will be sorted using the collating sequence table of the specified language. Keys, as well as data in the record, may contain 8-bit character data.

A file language attribute may be supplied when a KSAM/3000 file is created to provide a key file organized according to the collating sequence of a native language. The language attribute is provided when the file is created. All KSAM/3000 files created before NLS was introduced are considered to have NATIVE-3000 as a language attribute.

A KSAM/3000 file can be built with KSAMUTIL, or programmatically using FOPEN.

## Creating KSAM/3000 Files with KSAMUTIL

When using KSAMUTIL, the parameter LANG=*langname* or LANG=*langnum* may be supplied on the BUILD command, as shown in the dialog below. NATIVE-3000 is used as the default language attribute if no language is specified.

The language specified in the LANG= parameter must be installed on the system when the command is issued for KSAMUTIL to build the file. If the language is not installed, an error message is returned and the file is not built.

The following dialog indicates Danish as the specified language and the language attribute of the KSAM/3000 file is to be checked by the VERIFY command (mode 3):

```
:RUN KSAMUTIL.PUB.SYS

HP32208A.03.13  THU, FEB 16, 1984,  8:54 AM    KSAMUTIL VERSION:A.03.13
>BUILD TEST;REC=-80,3,F,ASCII;KEY=B,1,4;KEYFILE=TESTK;LANG=DANISH
>VERIFY

WHICH (1=FILE INFO, 2=KSAM PARAMETERS, 3=KSAM CONTROL, 4=ALL)? 4

TEST.LORO.NLS            CREATOR=SLORO
FOPTIONS(004005)=KSAM, :FILE, NOCCTL, F, FILENAME, ASCII, PERM
AOPTIONS(000400)=DEFAULT, NOBUF, DEFAULT, NO FLOCK, NO MR, IN
RECSIZE:SUB:TYP:LDNUM:DRT:UN.:  CODE:LOGICAL PTR: END OF FILE:FILE LIMIT
    -80:  9:  0:    3: 89:  2:     0:           0:           0:     1023
 LOG. COUNT:PHYS. COUNT:BLK SZ:EXT SZ:NR EXT: LABELS:LDN:   DISCADDR:
  0:          0:  -240:    43:     8:      0:  3:00000234251:

KEY FILE=TESTK    KEY FILE DEVICE=4        SIZE=       114  KEYS=     1
FLAGWORD(000020)=RANDOM PRIMARY, FIRST RECORD=0, PERMANENT
KEY TY LENGTH   LOC. D KEY BF  LEVEL
  1  B      4      1 N    168      1

DATA FILE = TEST    VERSION= A.3.13
KEY CREATED= 47/'84  9: 0: 7.6     KEY ACCESS=  47/'84  9: 0:19.2
KEY CHANGED= 47/'84  9: 0: 8.5     COUNT START= 47/'84  9: 0: 8.6
DATA RECS  =      0 DATA BLOCKS=     0 END BLK WDS=         0
DATA BLK SZ=    120 DATA REC SZ=    80 ACCESSORS=           0
FOPEN             1 FREAD            0 FCLOSE               1
FREADDIR          0 FREADC           0 FREADBYKEY           0
FREMOVE           0 FSPACE           0 FFINDBYKEY           0
FGETINFO          1 FGETKEYINFO      0 FREADLABEL           0
FWRITELABEL       0 FCHECK           0 FFINDN               0
FWRITE            0 FUPDATE          0 FPOINT               0
FLOCK             0 FUNLOCK          0 FCONTROL             0
FSETMODE          0 FREE KEYBLK      0 FREE RECS            0
KEYBLK READ       2 KEYBLK WRITTEN   0 KEYBLK SPLIT         0
KEY FILE EOF     10 FREE KEY HD      0 SYSTEM FAILURE       0
MIN PRIME         0 MAX PRIME        0 RESET DATE
DATA FIXED     TRUE DATA B/F         3 TOTAL KEYS           1
FIRST RECNUM      0 MIN RECSIZE      4 LANG            DANISH
WHICH (1=FILE INFO, 2=KSAM PARAMETERS, 3=KSAM CONTROL, 4=ALL)?[Return]

>E

END OF PROGRAM
:
```

## Error Messages

KSAMUTIL error messages are listed in Table 3-5.

Table 3-5. KSAMUTIL Error Messages

| ERROR # | MESSAGE | CAUSE | ACTION |
|---|---|---|---|
| 1070 | 'LANG' NOT FOLLOWED BY '=' OR HAS TOO MANY PARAMETERS. | Improper syntax was used in specifying the language name. | Enter the language name using the correct syntax. |
| 1071 | 'LANG' LANGUAGE VALUE TOO LONG OR ABSENT. | The language name is too long or missing a parameter. | Enter the correct language name. |
| 1072 | 'LANG' LANGUAGE NUMBER VALUE INVALID. | The language number contains invalid characters. | Enter the correct language number. |
| 1073 | 'LANG' LANGUAGE NOT SUPPORTED. | The language specified is not configured on your system, or not a valid language name or number. | Ask the System Manager to configure the language on your system. |
| 1074 | NATIVE LANGUAGE SUPPORT IS NOT INSTALLED. | NLS is not installed on your system. | Ask the System Manager to configure the language on your system. |
| 1075 | NATIVE LANGUAGE SUPPORT LANGUAGE NOT SUPPORTED. | An NLS MPE error occurred. No language table exists for the language specified. | Ask the System Manager to configure the language on your system. |
| 1076 | NATIVE LANGUAGE SUPPORT RELATED ERROR. | An NLS MPE error occurred. | Ask the System Manager to configure the language on your system; if it is already configured, contact your Hewlett-Packard support representative. |

Refer to Appendix A of the KSAM/3000 Manual (30000-90079) for more information on error messages.

## Creating KSAM/3000 Files Programmatically

The user must provide the *langnum* when calling FOPEN to build a KSAM/3000 file. The *langnum* is stored in word 10 of the KSAMPARAM array. The FOPEN intrinsic checks each time a KSAM/3000 file is opened to determine whether the language used is configured on the system. For backward compatibility, bit 11 in the flagword (word 15) must be set to 1 if a language other than 0 (NATIVE-3000) is used, to denote that word 10 contains valid information.

If bit 11 of the flagword is 0, the default language (NATIVE-3000) is used and the data in word 10 is ignored. If the language is not configured, condition code CCL is returned by FOPEN.

The file system error messages listed in Table 3-6 have been included with NLS:

Table 3-6. KSAM/3000 File System Error Messages

| ERROR # | MESSAGE | CAUSE | ACTION |
|---|---|---|---|
| 196 | LANGUAGE NOT SUPPORTED. | The language name or number specified for FOPEN is not configured on your system, or is not a valid language name or number. | Ask the System Manager to configure the language on your system. |
| 197 | NATIVE LANGUAGE SUPPORT RELATED ERROR. | An NLS MPE error occurred on an FOPEN call. | Contact your Hewlett-Packard support representative. |

Refer to Appendix A in the *KSAM/3000 Manual* (30000-90079) for a complete list of KSAM/3000 file system errors.

## Modifying KSAM/3000 Files

Every record added or updated in a KSAM/3000 file has its new keys of type BYTE inserted in the key file according to the collating sequence of the language defined for that KSAM/3000 file. That function is handled internally by a system version of the NLCOLLATE intrinsic when the language attribute of the file is different from NATIVE-3000. A new key in a file with a NATIVE-3000 language attribute will be ordered according to the result of a BYTE COMPARE between the key of the new record and the keys of the records already in the key file.

## Generic Keys

NLS collating sequences differ from the USASCII collating sequences, and the differences must be considered when performing generic key searches. Refer to Appendix C, "Collating in European Languages," for more information.

The description of a generic key search in a KSAM/3000 file with a native language attribute is presented from an application point of view.

Keys matching a certain generic key may not be in consecutive order in the key file because the keys are sorted according to a native language collating sequence. The key sequence in Figure 3-1 illustrates this with a French KSAM/3000 file; *keylength* is 4, the generic *keylength* is 2. The partial key "**aa**" appears in non-consecutive keys (with a result of 0 in the last column of the figure). Records containing partial keys (such as "AA" or "Aa") are intermixed according to the French collating sequence. These keys have a result of 1 listed.

If a generic key search is performed in a KSAM/3000 file with a language attribute other than NATIVE-3000, the application program must determine whether the retrieved record matches the generic key and, even if it does not, whether subsequent records might still match it.

The codes returned by NLKEYCOMPARE are shown in Table 3-7. Refer to Chapter 4, "Native Language Intrinsics," for a complete discussion of the NLKEYCOMPARE intrinsic.

Table 3-7. Results returned by the NLKEYCOMPARE Intrinsic

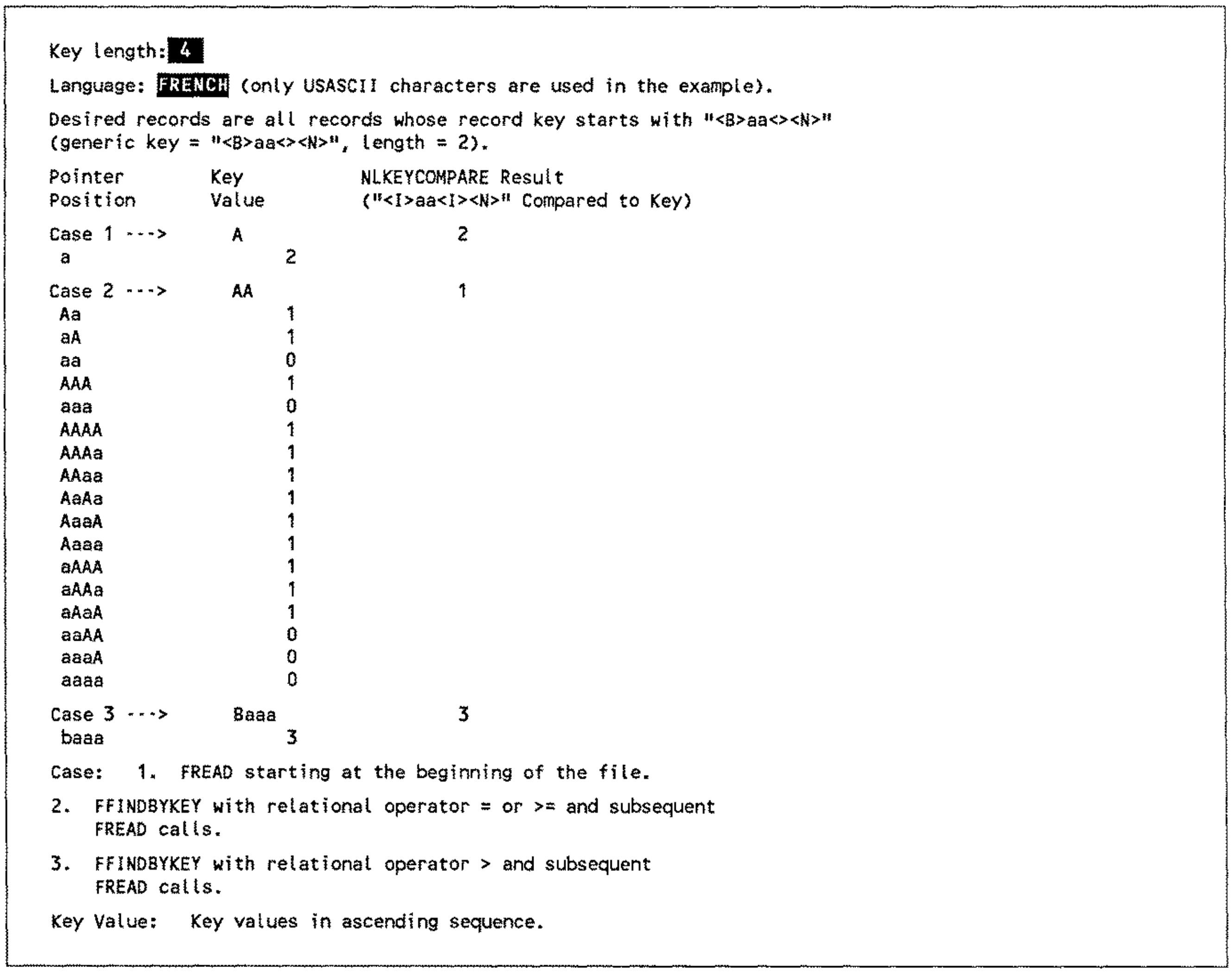
| RESULT | MEANING |
|---|---|
| 0 | The retrieved key matches the generic key exactly. |
| 1 | The retrieved key does not match the generic key.<br>Uppercase/lowercase priority or accent priority is different. |
| 2 | The retrieved key value is less than the generic key.<br>It precedes the designated key in the collating sequence. |
| 3 | The retrieved key is greater than the generic key. |

The generic key search sequence is:

1. After FFINDBYKEY has been called with >= as relational operator (relop), the logical record pointer points to the data record indicated by the arrow labeled "Case 2".
2. The subsequent FREAD call will retrieve the data record. When the partial key "AA" is compared to the generic key "aa" they are found to be different.
   This comparison is done by calling the intrinsic NLKEYCOMPARE using the generic key and the key found in the record. The result returned by NLKEYCOMPARE tells the application whether the FREAD delivered a record:
   a. Before the desired range (result 2).
   b. In the desired range with an uppercase/lowercase or accent priority difference (result 1).
   c. With an exact match (result 0).
   d. After the desired range (result 3).
3. To get all records whose key match the generic key exactly, the FREAD calls and subsequent NLKEYCOMPARE calls should continue until a result of 3 is returned.

When performing a generic key search in a KSAM/3000 file with a native language attribute other than NATIVE-3000 use the NLKEYCOMPARE intrinsic to compare partial keys and generic keys.

Refer to programs I and J in Appendix H, "Example Programs," for generic key searches in KSAM/3000 files with native language attributes.



```
Key length: 4

Language: FRENCH (only USASCII characters are used in the example).

Desired records are all records whose record key starts with "<B>aa<><N>"
(generic key = "<B>aa<><N>", length = 2).

Pointer        Key           NLKEYCOMPARE Result
Position       Value         ("<I>aa<I><N>" Compared to Key)

Case 1 --->     A                      2
 a                   2

Case 2 --->     AA                     1
 Aa                  1
 aA                  1
 aa                  0
 AAA                 1
 aaa                 0
 AAAA                1
 AAAa                1
 AAaa                1
 AaAa                1
 AaaA                1
 Aaaa                1
 aAAA                1
 aAAa                1
 aAaA                1
 aaAA                0
 aaaA                0
 aaaa                0

Case 3 --->     Baaa                   3
 baaa                3

Case:   1.  FREAD starting at the beginning of the file.

2.  FFINDBYKEY with relational operator = or >= and subsequent
    FREAD calls.

3.  FFINDBYKEY with relational operator > and subsequent
    FREAD calls.

Key Value:   Key values in ascending sequence.
```

**Figure 3-1. Generic Key Searches**

## Copying From KSAM/3000 File to KSAM/3000 File

If the KSAM/3000 file already exists (built via KSAMUTIL or programmatically) the keys of type BYTE are put into the new file according to the collating sequence belonging to the language of the TO file. If the file does not exist, a new file is built with the same language attribute as the FROM file.

## Changing the Language Attribute of a KSAM/3000 File

FCOPY/3000 cannot be used to change the language attribute of an existing file. KSAMUTIL must be used to build a new KSAM/3000 file with the new language attribute. Then the data can be copied to this file using FCOPY/3000. Keys of type BYTE in the destination key file will be ordered according to the collating sequence of the new language.

## Moving NLS KSAM/3000 Files To Pre-NLS MPE

Restoring a KSAM/3000 file with a native language attribute other than NATIVE-3000 to a system without NLS installed can result in an incorrect key sequence in the key file for type BYTE keys. Systems without NLS installed do not recognize any collating sequence except NATIVE-3000.

If a file with a native language attribute other than NATIVE-3000 is restored, the first FOPEN on the file will return the same error condition code as if a system failure occurred while the file was opened. KSAMUTIL should be used to build a new KSAM/3000 file. The file with the native language attribute is recovered, and FCOPY/3000 is used to copy the recovered file into the new KSAM/3000 file. Refer to the dialog below for an example of this recovery procedure.

```
:RUN KSAMUTIL.PUB.SYS

HP32208A.03.10 SAT,  SAT, MAY 26,1984, 12:33 PM  KSAMUTIL VERSION:A.03.10

>BUILD NEWDATA;REC=-80,3,F,ASCII;KEY=B,1,4:KEYFILE=NEWKEY
>KEYINFO OLDDATA;RECOVER

>EXIT

:FCOPY FROM=OLDDATA;TO=NEWDATA;KEY=0

:RUN KSAMUTIL.PUB.SYS

HP32208A.03.10 SAT,  SAT, MAY 26,1984, 12:33 PM  KSAMUTIL VERSION:A.03.10
>PURGE OLDDATA
>RENAME NEWDATA,OLDDATA
>RENAME NEWKEY,OLDKEY
>EXIT
```

# QUERY

QUERY provides access to IMAGE databases to allow the following functions to be executed:

- Data entry.
- Data value modification or deletion online.
- Data retrival, meeting selection criteria.
- Data retrival, sort and reporting functions.

QUERY operations are performed by entering commands (English language key words and parameters).

Native Language Support (NLS) features can be accessed in QUERY to retrieve data which meet user-defined selection criteria, and to sort data according to native language collating sequences. The user must know what the native language in QUERY is, how the language is specified, how the language affects the output, and how to determine which language is being used.

IMAGE databases have a language attribute that describes the collating sequence used in sorted chains and locking. This language attribute does not affect the QUERY operation.

Although QUERY commands are in English, the user can expect the output data to be sorted and formatted according to the QUERY user's language. The language of the database may determine the data sequence while using QUERY passively for data retrieval (FIND). When data is being sorted or formatted by QUERY, the user's language will determine the ordering and formatting of the data.

For example, in a French database with a QUERY user's language of Danish, data items in a sorted chain might be retrieved according to the French collating sequence; but the sorting or formatting is done according to Danish criteria.

The user can specify the QUERY user's language by:

- Using a QUERY command:

  >LANGUAGE=*langnum* or >LANGUAGE=*langname.*

  The default is NLUSERLANG. For example, if the user's language is French, the QUERY command is:

  >LANGUAGE=7 or >LANGUAGE=FRENCH

- Using an MPE command:

  :SETJCW NLUSERLANG=*langnum.*

  The default is NATIVE-3000. For example, if the user's language is French, the MPE Job Control Word NLUSERLANG may be used:

  ```
  :SETJCW NLUSERLANG=7
  ```

The >LANGUAGE= command always overrides NLUSERLANG. If neither option is used to specify the user's language, QUERY assumes LANGUAGE=0 (NATIVE-3000). NATIVE-3000 is the default, which ensures backward compatibility. When the user's language is NATIVE-3000, QUERY performs as it did before NLS features were available.

QUERY allows access to more than one database at the same time; more than one database language attribute may be active at the same time. In any case, upshifting, collating, range selection, formatting, or sorting is dependent on the QUERY user's language specified by the user via the JCW NLUSERLANG or the LANGUAGE= command.

## Command Summary

NLS can affect QUERY in upshifting data, range selection, date format, real number conversions, and sorted lists and numeric data editing in REPORT.

### Upshifting Data (Type U Items)

QUERY upshifts commands and the data of type U items. QUERY commands are upshifted according to NATIVE-3000. Data is upshifted according to the user's language to ADD, UPDATE, REPLACE, UPDATE ADD, UPDATE REPLACE, FIND, LIST, MULTIFIND, and SUBSET.

### Range Selection

QUERY collates data according to the user's language in FIND, LIST, MULTIFIND, or SUBSET. The MATCH feature (in FIND and MULTIFIND commands) is no longer valid when LANGUAGE <> 0 (NATIVE-3000). QUERY will display an error message if MATCH is used in an interactive mode, and will abort the session in a batch mode.

### Date Format

DATE is a reserved word in the REPORT command which provides the system date. It is formatted according to the user's language.

### Real Number Conversions

In the commands REPORT and LIST the output is formatted according to the user's language. For example, 123.45 in NATIVE-3000 becomes 123,45 in FRENCH.

### Sorted Lists in Report

QUERY sorts type U or X items in a REPORT according to the collating sequence of the user's language.

### Numeric Data Editing in Report

QUERY converts the data edited using the NATIVE-3000 edit mask (using the period as a decimal point and a comma as thousands separator) to the corresponding characters in the user's language.

The commands listed in Table 3-8 are used to obtain language-dependent information. Refer to the *QUERY Reference Manual* (30000-90042) for a complete description of these commands.

Table 3-8. Commands For Language-Dependent Information

| COMMAND | LANGUAGE-DEPENDENT INFORMATION |
|---|---|
| `>HELP LANGUAGE` | Explains LANGUAGE command function, format and parameters. |
| `>SHOW LANGUAGE` | Displays the QUERY user's language. |
| `>FORM` | Displays the database language attribute. |

## Error Messages

QUERY error messages which support the NLS enhancement are listed in Table 3-9.

Table 3-9. QUERY Error Messages

| MESSAGE | MEANING | ACTION |
|---|---|---|
| DBINFO MODE 901 FAILED. CHECK DATA BASE LANGUAGE ATTRIBUTE AND IMAGE VERSION. | The version of IMAGE on your system does not have NLS features. | This is a warning. The user may wish to update IMAGE/3000 to the same level as QUERY. |
| EXPECTED A LANGUAGE NUMBER OR NAME. | The LANGUAGE command only accepts the name of a language or the number associated with that name. | Enter HELP LANGUAGE for a complete explanation of the command and then re-enter it. |
| INTERNAL QUERY NLS PROBLEM. | The NLS subsystem encountered an error from which it could not recover while attempting to initialize language-dependent information. | Contact your Hewlett-Packard support representative. |
| LANGUAGE INVALID. NATIVE-3000 USED. | Language specified not configured. The default, NATIVE-3000 was used. | Run NLUTIL.PUB.SYS to list the languages and associated numbers available on your system. |
| LANGUAGE NOT CONFIGURED ON THIS SYSTEM. NATIVE-3000 USED. | Languages are configured on each system. Language specified is not available on your system. The default language is NATIVE-3000. | Run NLUTIL.PUB.SYS to list the languages and associated numbers available on your system. |
| MATCH NOT VALID WHEN LANGUAGE <> NATIVE-3000. | QUERY can only allow the matching option for NATIVE-3000. | If possible, change the language to NATIVE-3000 for the match. |
| NLCOLLATE INTRINSIC INTERNAL ERROR. | An unexpected error condition occurred while doing a comparison of the data. | Contact your Hewlett-Packard support representative. |
| NLUTIL INTRINSIC INTERNAL ERROR. | The NLS subsystem encountered an error from which it could not recover while attempting to initialize language-dependent information. | Contact your Hewlett-Packard support representative. |

Table 3-9. QUERY Error Messages (cont.)

| MESSAGE | MEANING | ACTION |
|---|---|---|
| USER LANGUAGE INVALID. | User language not available. Only NATIVE-3000 is available on your system. | Ask the System Manager to configure the desired language on your system. |
| USER LANGUAGE NOT CONFIGURED ON THIS SYSTEM. NATIVE-3000 USED. | Languages are configured on each computer system. Language specified is not available on your system. The default language is NATIVE-3000. | Run NLUTIL.PUB.SYS to list the languages and associated numbers available on your system. |

## SORT-MERGE/3000

SORT-MERGE/3000 organizes records in a file according to the collating sequence of the keys. The default collating sequence for character data is based on the binary values of the characters. EBCDIC and user-defined sequences can also be used. Native Language Support (NLS) in SORT-MERGE/3000 provides the user with the option of collating according to a native language sequence.

SORT-MERGE/3000 can be used as a stand-alone program or programmatically.

### Stand-Alone SORT-MERGE/3000

The key type CHARACTER allows the user to access native language collating sequences. The specific native language collating sequence is assigned by the LANGUAGE command.

C[HARACTER] The collating sequence defined in the LANGUAGE command is used to sort keys of type CHARACTER. Refer to the dialog below for an example of the use of the CHARACTER key type.

| COMMAND | SYNTAX | DESCRIPTION |
|---|---|---|
| LANGUAGE | >L[ANGUAGE] [IS] {*langnum*}<br>{*langname*} | Defines the native language collating sequence to be used to sort keys of type CHARACTER. |

The LANGUAGE command may specify a language ID number (*langnum*) or language name (*langname*). The language specified must be configured on the system. If the LANGUAGE command is not used, the language to be used for collating keys of type CHARACTER defaults to NLDATALANG, the language returned by the NLGETLANG intrinsic (mode 2). In the dialog below, the LANGUAGE command designates Swedish. The VERIFY command will confirm which language collating sequence will be used for the SORT or MERGE stand-alone program:

```
:RUN SORT.PUB.SYS
HP32214C.04.00 SORT/3000 MON, JAN 30, 1984, 1:52 PM
(C) HEWLETT-PACKARD CO. 1983

>INPUT MYFILE
>OUTPUT $STDLIST
>KEY 1,4, CHARACTER
>LANGUAGE IS SWEDISH
>VERIFY

INPUT FILE = MYFILE
RECORD LENGTH = SAME AS THAT OF THE INPUT FILE
OUTPUT FILE = $STDLIST
KEY POSITION     LENGTH     TYPE     ASC/DESC
      1            4       CHAR        ASC       (MAJOR KEY)
LANGUAGE IS SWEDISH
>END
```

## Programmatic SORT-MERGE/3000

To use SORT-MERGE/3000 programmatically with NLS features, the user must designate the collating sequence with the *charseq* parameter in the SORTINIT and MERGEINIT intrinsics.

### Syntax

```
SORTINIT                IA         IA           IV          IV       DV        IV
                      (inputfiles,outputfiles,outputoption,reclen,numrecs,numkeys,
                       IA   IA       LP           P         IA       L        I
                      keys,altseq,keycompare,errorproc,statistics,failure,errorparm,
                           I              IA              O-V
                      spaceallocation,charseq,parm2)

MERGEINIT               IA            P            IA          P            LV
                      (inputfiles,preprocessor,outputfiles,postprocessor,keysonly,
                         IV      IA    IA        LP           P          IA       L
                      numkeys,keys,altseq,keycompare,errorproc,statistics,failure,
                         I              I             IA             O-V
                      errorparm,spaceallocation,charseq,parm2)
```

### PARAMETERS

The following parameters apply:

*numkeys* and *keys*

The *numkeys* parameter is an integer.
The *keys* parameter is an integer array.
These parameters describe the way records are sorted or merged. One of these parameters cannot be specified without the other. The use of *numkeys* and *keys* disallows the use of *keycompare*. The number of keys used during the comparison of records is contained in *numkeys*, and the way records are compared is specified by *keys*. For each key specified, *keys* contains three words:

The first word gives the position of the first character of the key within the record. The second word gives the number of characters in the key. The third word (bits 0-7) gives the ordering sequence of the records (a value of 0 for ascending, 1 for descending). Bits 8-15 of the third word indicate the type of data according to the following convention:

0 = logical or byte (same as type BYTE in interactive mode)
1 = two's complement, including integer and double integer
2 = floating point
3 = packed decimal
4 = Display-Trailing-Sign
5 = packed decimal with even number of digits
6 = Display-Leading-Sign
7 = Display-Leading-Sign-Separate
8 = Display-Trailing-Sign-Separate
9 = character (collating sequence of *charseq* is used)

*charseq* A two-word integer array.
To utilize *charseq*:

Set word 0 to 1.

Set word 1 to the *langnum* of the collating sequence to be used for sorting keys of type 9 (CHARACTER). The language designated must be configured on the system.

Whenever keys of type CHARACTER are compared, and *charseq* has been used to request a native language collating sequence (for example, Dutch, Spanish, Danish), SORT or MERGE will call the NLCOLLATE intrinsic to do a native language comparison.

If NATIVE-3000 has been designated by the user or as a default, SORT-MERGE/3000 will do a direct byte comparison on keys of type CHARACTER. NATIVE-3000 is an artificial language whose collating sequence is based on the binary values of the characters.

Refer to the *SORT-MERGE/3000 Manual* (32214-90002) for other parameter descriptions.

## Error Messages

NLS-specific error messages include those for Programmatic SORT (Table 3-10), Interactive SORT (Table 3-11), Programmatic MERGE (Table 3-12), and Interactive MERGE (Table 3-13).

Table 3-10. Programmatic SORT Error Messages

| | | |
|---|---|---|
| 29 | LIB | SORT LANGUAGE NOT SUPPORTED. |
| 30 | LIB | NLINFO ERROR OBTAINING LENGTH OF COLLATING SEQUENCE TABLE. |
| 31 | LIB | NLINFO ERROR LOADING COLLATING SEQUENCE TABLE. |
| 32 | LIB | INVALID CHARSEQ PARAMETER. |

Table 3-11. Interactive SORT Program Error Messages

| | |
|---|---|
| 40 | INVALID LANGUAGE ID. |
| 41 | THE LANGUAGE SPECIFIED IS NOT SUPPORTED. |

Table 3-12. Programmatic MERGE Error Messages

| | | |
|---|---|---|
| 21 | LIB | SORT LANGUAGE NOT SUPPORTED. |
| 22 | LIB | NLINFO ERROR OBTAINING LENGTH OF COLLATING SEQUENCE TABLE. |
| 23 | LIB | NLINFO ERROR LOADING COLLATING SEQUENCE TABLE. |
| 24 | LIB | INVALID CHARSEQ PARAMETER. |

Table 3-13. Interactive MERGE Program Error Messages

| | |
|---|---|
| 37 | INVALID LANGUAGE ID. |
| 38 | THE LANGUAGE SPECIFIED IS NOT SUPPORTED. |

## Performance Considerations

SORT-MERGE/3000 executes more slowly when keys of type CHARACTER and a native language collating sequence are requested. The complex collating algorithms required by some of the languages may use additional CPU time. The speed of SORT-MERGE/3000 is unchanged when a native language collating sequence is not requested or when NATIVE-3000 is requested.

## COBOLII Sort and Merge

The syntax for the SORT and MERGE verbs has changed slightly for NLS. It is now possible to specify the native language whose collating sequence is to be used. The old syntax allowed only an alphabetic name:

```
[COLLATING SEQUENCE IS alphabet-name]
```

The syntax has been changed to:

```
                       {alphabetname}
[COLLATING SEQUENCE IS {languagename}]
                       {langnum}
```

With the addition of NLS features, *alphabetname* retains the same meaning, *languagename* is an alphanumeric data item containing the name of the language whose collating sequence is to be used, and *langnum* is an integer data item containing the language identification number of the language to be used.

The following demonstrates the use of the SORT verb syntax:

```
002600 WORKING-STORAGE SECTION.
002700 01  AN-LANG-NAME  PIC X(16) VALUE "FRENCH"
002800 01  NUM-LANG-ID   PIC S9(4) COMP VALUE 7.

003300 SORT SORT-FILE
003400      ASCENDING KEY SORT-KEY
003500      COLLATING SEQUENCE IS AN-LANG-NAME
003600      USING IN-FILE
003700      GIVING OUT-FILE.

004000 SORT SORT-FILE
004100      ASCENDING KEY SORT-KEY
004200      COLLATING SEQUENCE IS NUM-LANG-ID
004300      USING IN-FILE
004400      GIVING OUT-FILE.

005000 SORT SORT-FILE
005100      ASCENDING KEY SORT-KEY
005300      USING IN-FILE
005400      GIVING OUT-FILE
```

## VPLUS/3000

The VPLUS/3000 product consists of five major parts: Intrinsics, FORMSPEC, ENTRY, REFSPEC, and REFORMAT.

VPLUS/3000 Native Language Support (NLS) enables an applications designer to create interactive end-user applications which reflect both the user's native language and the local custom for numeric and date information in the supported languages. NLS provides these specific features in VPLUS/3000:

- Native decimal and thousands indicators.
- Native language month names for dates.
- Alphabetic upshifting of native characters.
- Native characters in single value comparisons and table checks.
- Native collating sequence in range checks.

VPLUS/3000 does not support the application design process in native languages. Form names, field identifiers, and field tags support only USASCII characters.

REFSPEC and REFORMAT do not use NLS features. These programs interact with users in NATIVE-3000 only.

### Language Attribute

VPLUS/3000 contains an NLS language attribute option which allows the applications programmer to design an international or language-dependent forms file. If a native language attribute is not specified, the forms file is unlocalized.

The forms file reflects the language characteristics of the application. Each forms file has a global language ID number. The application may be unlocalized, language-dependent, or international. For examples of these applications, see Figures 1-3, 1-4, and 1-5 in Chapter 1, "Introduction to NLS."

#### Unlocalized

If no language ID number is assigned to a forms file, it will default to 0 (NATIVE-3000).

#### Language Dependent

This application only operates in a single language context. The language ID number is assigned when the forms file is designed. If the text needs to be in the native language, unique versions of a forms file are required for each language supported.

#### International

Multinational corporations may need to maintain a business language for commands, titles, and menus in addition to accommodating the language of the end user for the actual data retrieved or displayed. For this application, select "-1" as the language ID number for the forms file. The VPLUS/3000 intrinsic VSETLANG must be called at run time to assign the appropriate language.

## Setting The Language ID Number

The components of a form which can be language-dependent are the text, the initial values of fields, and the field edit rules. The language ID number determines the context for data editing, conversion, and formatting. The FORMSPEC language controls the context when the forms file is designed. The forms file language controls the context when the forms file is executed.

The forms designer sets language ID number values for the forms file via the FORMSPEC Terminal/Language Selection Menu. The forms file language defaults to 0 (NATIVE-3000) if no language ID number is specified for it. NATIVE-3000 is currently the only selection available for the FORMSPEC language. This means that initial values and processing specifications must be defined with the month names and numeric conventions of NATIVE-3000.

The designer can change the forms file language ID number at any time. The value must be a positive number or a zero for a single language application. If the value is acceptable, but the language is not configured, FORMSPEC will issue a warning message. The language ID number will not be rejected. The designer is prompted to confirm the value or change it.

For multiple language applications, the forms designer selects a forms file language ID number value of -1. The international language ID number indicates that the intrinsic VSETLANG will be called at run time to select the language ID number for the forms file. If an application uses an international forms file without calling VSETLANG, it will be executed in the default, NATIVE-3000. If VSETLANG is called for an unlocalized or language-dependent forms file, an error code will be returned.

The designer has three options in designing an application to work effectively with multiple languages:

- Develop several language-dependent forms files.
- Create one international forms file.
- Produce a combination of language-dependent files and an international forms file.

VGETLANG may be used to determine whether a language-dependent forms file or an international forms file is being executed. If VGETLANG indicates an international forms file, VSETLANG must be called to select the actual language. Refer to the VGETLANG and VSETLANG intrinsics at the end of this section.

## Field Edits

NATIVE-3000 must be used to specify date and numeric fields within FORMSPEC. VPLUS/3000 will convert the value when the forms file is executed to be consistent with the native language selected. Single value comparisons (LT, LE, GT, GE, EQ, NE), table checks, and range checks (IN, NIN) specified within FORMSPEC may contain any character in the 8-bit extended character set consistent with the selected language ID number. When the form is executed at run time, the collating table for the native language specified is used to check whether the field is within a range.

### Date Handling

VPLUS/3000 supports several date formats and three date orders: MDY, DMY, YMD. Any format is acceptable as input when the form is executed, provided that the field length can accommodate the format. The forms designer specifies the order for each date-type field. With NLS, the native month names are edited and converted to numeric destinations. The format and the date order are not related to the language of the forms file.

## Numeric Data

Decimal and thousands indicators are language-dependent in the NUM [*n*] and IMP*n* fields. When data is moved between fields and automatic formatting occurs for data entered in any field, recognition, removal, or insertion of these decimal and thousands indicators is language-dependent. The optional decimal symbol in constants is also language-dependent.

---

**NOTE**

VPLUS/3000 edit processing specifications and terminal edit processing statements are separate and are not checked for compatibility. There will be no check that the designer has specified a terminal local edit which is consistent with the language-dependent symbol for the decimal point (DEC TYPE EUR, DEC TYPE US) in the configuration phase.

---

### Native Language Characters

If a native language ID number has been specified in the forms file, the UPSHIFT formatting statement will use native language upshift tables.

Range checks and the single value comparisons LT, LE, GT and GE involve collating sequences. When the form is executed, the native language collating sequence table designated by the language ID number is used to check whether the field passes the edit.

NLS features in VPLUS/3000 do not include support for pattern matching with native characters. MATCH uses USASCII specifications.

## Entry and Language ID Number

The forms file language determines the user language in ENTRY unless the file is international (-1). The ENTRY program uses the intrinsic VGETLANG to identify the language of the forms file selected by the designer.

If the forms file is international, ENTRY calls the NLS intrinsic NLGETLANG (mode 1). If it returns a value of UNKNOWN, the user is prompted for a language ID number. Once a valid language ID number is determined, ENTRY calls the VSETLANG intrinsic to specify the corresponding language.

The batch file does not have a language indicator. Users with different native languages may collect data in the same batch file if the associated forms file is international.

## Error Messages

VPLUS/3000 Error Messages are listed in Table 3-14.

Table 3-14. VPLUS/3000 Error Messages

| NUMBER | MESSAGE | ACTION |
|---|---|---|
| 9001 | NATIVE LANGUAGE SUPPORT SOFTWARE NOT INSTALLED. | ask the System Manager to install NLS software. |
| 9002 | LANGUAGE SPECIFIED IS NOT CONFIGURED ON THIS SYSTEM. | Select another language or ask the System Manager to configure the desired language. |
| 9011 | LANGUAGE NOT CONFIGURED. CHANGE OR HIT "ENTER" TO PROCEED. | Language specified is not configured on the system. Forms file produced can only be executed on a system configured with that language. |
| 9014 | ATTEMPTED SETTING A LANGUAGE DEPENDENT FORMS FILE TO ANOTHER LANGUAGE. | VSETLANG can only be used with international forms files. |
| 9015 | NATIVE-3000 IS CURRENTLY THE ONLY SELECTION AVAILABLE. | FORMSPEC language can only be 0 in this version. |
| 9500 | LANGUAGE OF FORMS FILE IS NOT CONFIGURED ON THIS SYSTEM. | Ask the System Manager to configure the language or use forms file on a system with that language configured. |
| 9998 | LANGUAGE ID MUST BE 0 TO 999 OR -1 FOR INTERNATIONAL FORMS FILE. | Forms file language ID number must be between -1 and 999. |

## VPLUS/3000 Intrinsics

The VGETLANG and VSETLANG intrinsics are used only with the VPLUS/3000 subsystem. Intrinsic calls in VPLUS/3000 are usually in COBOL. Refer to the VGETLANG and VSETLANG sections for examples of calls in other programming languages.

### VGETLANG

This intrinsic returns the language ID number of the forms file being executed. The forms file must be opened before calling VGETLANG. Otherwise, CSTATUS returns a nonzero value.

**Syntax .**

```
CALL "VGETLANG" USING COMAREA,LANGNUM
```

**Parameters .**

COMAREA — The following COMAREA fields must be set before calling VGETLANG if not already set:

LANGUAGE - Set to code identifying the programming language of the calling program.

COMAREALEN - Set to total number of words in COMAREA.

CSTATUS - Set to nonzero value if call is unsuccessful. VGETLANG may set this field.

LANGNUM — Integer variable to which the language ID number of the forms file is returned.

**Examples .**

The following examples illustrate a call to VGETLANG:

| | |
|---|---|
| COBOL | `CALL "VGETLANG" USING COMAREA,LANGNUM.` |
| BASIC | `120 CALL VGETLANG(C(*),L).` |
| FORTRAN | `CALL VGETLANG (COMAREA,LANGNUM).` |
| SPL | `VGETLANG (COMAREA,LANGNUM);.` |

**Special Considerations .**

This intrinsic is used in the VPLUS/3000 subsystem only.

## VSETLANG

This intrinsic sets the language to be used by VPLUS/3000 at run time for an international forms file. The forms file must be opened before calling VSETLANG. Otherwise, CSTATUS returns a nonzero value.

If VSETLANG is called to set the language ID number for a language-dependent or unlocalized forms file, an error code of -1 will be returned to ERROR. For international forms files, both CSTATUS and ERROR return a value of zero and the forms file is processed with the native language ID number specified in LANGNUM.

### Syntax .

```
CALL "VSETLANG" USING COMAREA,LANGNUM,ERROR
```

### Parameters .

COMAREA — The following COMAREA fields must be set before calling VSETLANG (if not already set):

LANGUAGE - Set to code identifying the programming language of the calling language.

COMAREALEN - Set to total number of words in COMAREA.

CSTATUS - Set to nonzero value if call is unsuccessful. VSETLANG may set this field.

LANGNUM — An integer containing the ID number of the language to be used by VPLUS/3000.

ERROR — Integer to which the error code is returned. Zero means the call was successfully completed. A value of -1 is returned if the call is unsuccessful.

### Example .

The following examples illustrate a call to VSETLANG:

| | |
|---|---|
| COBOL | `CALL "VSETLANG" USING COMAREA,LANGNUM,ERROR.` |
| BASIC | `120 CALL VSETLANG(C(*),L,E).` |
| FORTRAN | `CALL VSETLANG (COMAREA,LANGNUM,ERROR).` |
| SPL | `VSETLANG (COMAREA,LANGNUM,ERROR);.` |

### Special Considerations .

This intrinsic is used in the VPLUS/3000 subsystem only.

## RAPID/3000

The Rapid/3000 products differ from other products in that they provide both compile (specification) time and run time support. In order to provide user access to the NLS intrinsics, the products maintain a global native language attribute while they are executing. This global attribute is used for all collating, upshifting, and sorting. The native language is specifiable at either run or compile time.

### Inform Language Attribute

Inform will use the language provided by the NLGETLANG intrinsic as the user language. A prompt in the option menu (appearing after all the other prompts) will provide the ability to change this attribute:

```
NATIVE LANGUAGE (NATIVE-3000) >
```

### REPORT LANG Option

By default, REPORT uses NATIVE-3000 as the language. A parameter for the OPTION statement in REPORT allows the specification of the native language at compile time:

OPTION LANG = *languagename;*

The REPORT program may also allow the user to select the language at run time:

```
OPTION LANG;
```

The user will be prompted with the question:

```
NATIVE LANGUAGE >
```

### Transact SET (LANGUAGE) Verb

A modifier is available on the SET statement in TRANSACT. There are three forms of this verb:

SET(LANGUAGE) : *1<br>
SET(LANGUAGE) *languagename*[ ,STATUS ] ; *2,3<br>
SET(LANGUAGE) *itemname*[ ,STATUS ] ; *2,3

These allow the programmer to specify a change of the native language at run time. The user can either specify a literal language name or ID number (which is checked at compile time) or give the name of and X(16) item which contains the name or number.

*1 - STATUS is set to the OLD language ID.

*2 - STATUS is set to the NEW language ID.

*3 - If the STATUS option is not specified and the language is not defined or configured, an error message is displayed and the language is set to 0 (NATIVE-3000). The specifying STATUS suppresses the error message and results in a negative value for STATUS if an error occurs. In this case, the language is left unchanged.

## Command Summary

### Upshift and Character Tables

The upshift and character type tables previously in the message have been replaced by the tables returned by NLINFO. These tables will be initialized at system startup and reinitialized whenever the language is changed. These tables were previously initialized from RAPIDCAT.

### Input and Output

In processing numeric items for input the thousands's separator will be ignored, provided it is not a delimiter character. For example, the NATIVE-3000 thousands's separator of "," is also a default delimiter. The radix character will be converted to ".". The default delimiters of ",=" will not be changed.

The processing of number items for output has been changed. All occurences of a "," in the resulting string are replaced by the thousands's separator, and all occurances of "." are replaced by the radix character.

### Date and Time

The procedures which print out data and time have been modified to call the native language procedures.

### IF and MATCH Changes

The code that processes IF statements and MATCH register comparisons has been modified to call NLCOLLATE and to do comparisons for native languages. The language in effect at the time of the comparisons is used (regardless of what language was used when the MATCH register was set).

### Native Language Accepting Intrinsics

The calls to intrinsics which accept a native language have been modified to pass in the language ID. This only applies to SORT. The language being used at the time the sort is initiated will be used.

# Native Language Intrinsics 4

The following categories of intrinsics are used by Native Language Support (NLS) and are described, in detail, in this chapter.

Table 4-1. Intrinsic Catagories

| Catagory | Intrinsic | Description |
|---|---|---|
| Information Retrieving | ALMANAC | Returns numeric date information. |
| | NLGETLANG | Returns the current language. |
| | NLINFO | Returns language-dependent information. |
| Character Handling | NLCOLLATE | Compares two character strings. |
| | NLFINDSTR | Searches for a string. |
| | NLJUDGE | Determines whether a character is a one-byte or two-byte Asian character. |
| | NLKEYCOMPARE | Compares strings of different length. |
| | NLREPCHAR | Replaces nondisplayable characters. |
| | NLSCANMOVE | Moves and scans character strings. |
| | NLTRANSLATE | Translates strings from and to EBCDIC. |
| | NLSUBSTR | Returns a substring. |
| | NLSWITCHBUF | Converts a string of characters from phonetic order to screen order and vice versa. |
| Time/Date Formatting | NLCONVCLOCK | Converts the time format. |
| | NLCONVCUSTDATE | Converts the custom date format. |
| | NLFMTCALENDAR | Formats the date. |
| | NLFMTCLOCK | Formats the time. |
| | NLFMTCUSTDATE | Formats the date into custom date format. |
| | NLFMTDATE | Formats date and time. |
| | NLFMTLONGCAL | Formats a long version of the date. |

Table 4-1. Intrinsic Catagories (cont.)

| Catagory | Intrinsic | Description |
|---|---|---|
| Number Formatting | NLNUMSPEC<br>NLCONVNUM<br>NLFMTNUM | Returns information needed for formatting and converting numbers.<br>Converts numbers from native to internal form.<br>Formats an internal number in native form. |
| Application Message Catalog | CATCLOSE<br>CATOPEN<br>CATREAD<br>NLAPPEND | Closes a mesage catalog.<br>Opens a message catalog.<br>Reads information from a message catalog.<br>Concatenates a filename and a language number. |

## NLS Date and Time Formatting Overview

The use of NLS intrinsics provides a variety of date and time formats as shown in Figure 4-1.

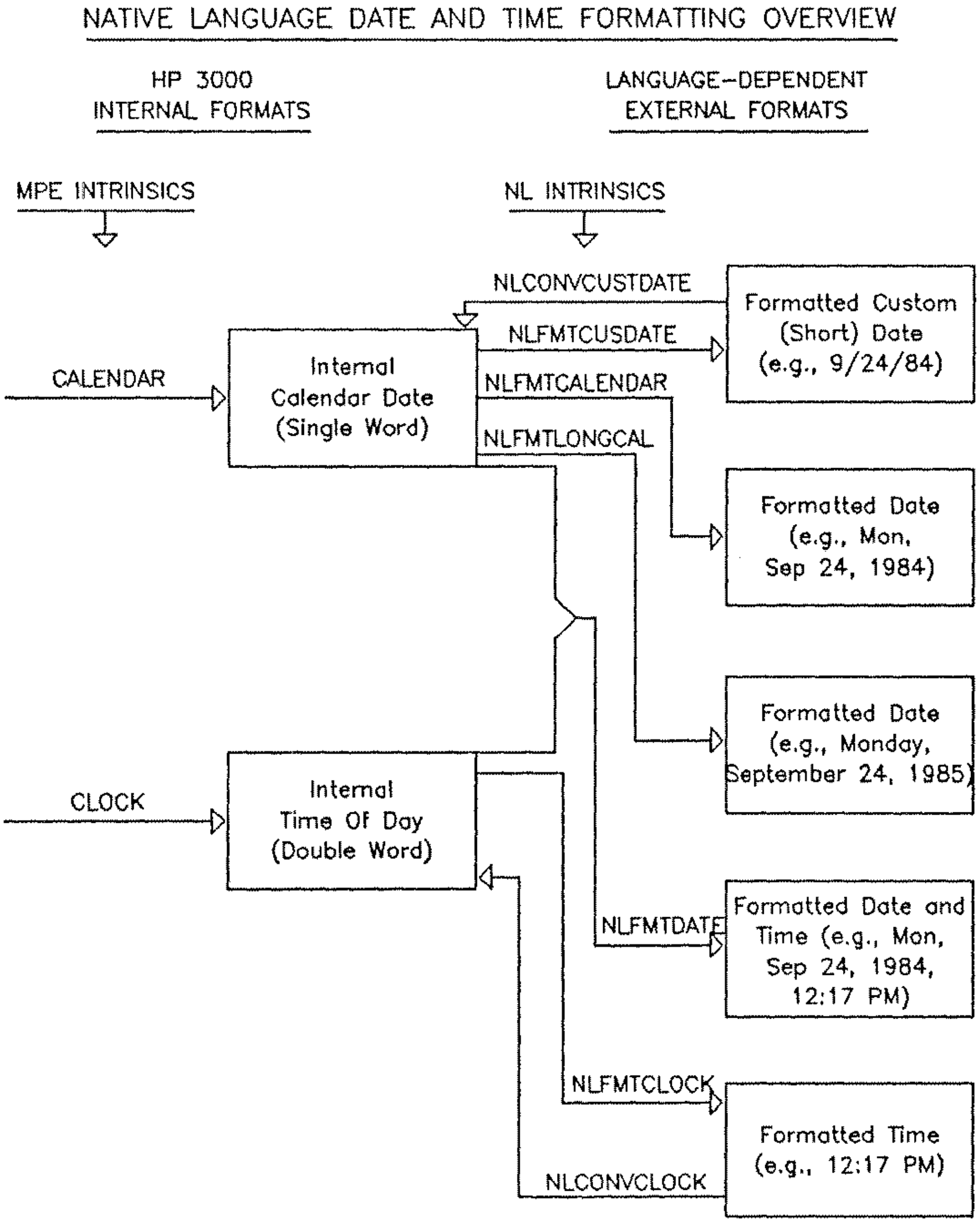

NATIVE LANGUAGE DATE AND TIME FORMATTING OVERVIEW
HP 3000 INTERNAL FORMATS
LANGUAGE-DEPENDENT EXTERNAL FORMATS
MPE INTRINSICS
NL INTRINSICS
NLCONVCUSTDATE
NLFMTCUSDATE
NLFMTCALENDAR
NLFMTLONGCAL
CALENDAR
Internal Calendar Date (Single Word)
Formatted Custom (Short) Date (e.g., 9/24/84)
Formatted Date (e.g., Mon, Sep 24, 1984)
Formatted Date (e.g., Monday, September 24, 1985)
CLOCK
Internal Time Of Day (Double Word)
NLFMTDATE
Formatted Date and Time (e.g., Mon, Sep 24, 1984, 12:17 PM)
NLFMTCLOCK
NLCONVCLOCK
Formatted Time (e.g., 12:17 PM)

**Figure 4-1. Date and Time Formatting Overview**

# ALMANAC

## ALMANAC (Intrinsic Number 406)

This intrinsic returns the numeric date information for a date returned by the CALENDAR intrinsic. The returned information is year of the century, month of the year, day of the month, and day of the week.

### Syntax

```
           LV    LA     I         I         I         I        O-V
ALMANAC (date,error,yearnum,monthnum,daynum,weekdaynum);
```

### Parameters

*date*

*logical by value (required)*
Contains the date in the format:

```
Bits  0               6 7            15
      +---------------+---------------+
      | Year of Century | Day of Year  |
      +---------------+---------------+
```

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 | No parameters available for returning values. |
| 2 | Day of the year out of range. |
| 3 | Year of the century out of range. |

*yearnum*

*integer by reference (optional)*
The year of the century is returned to this integer. For example, 00=1900 and 84=1984.

*monthnum*

*integer by reference (optional)*
The month of the year is returned to this integer. For example, 1=January and 12=December.

*daynum*

*integer by reference (optional)*
The day of the month is returned to this integer.

*weekdaynum*

*integer by reference (optional)*
The day of the week is returned to this integer. For example, 1=Sunday and 7=Saturday.

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

Refer to Programs D and E in Appendix H, "Example Programs" for examples of how this intrinsic is used.

# CATCLOSE

## CATCLOSE (Intrinsic Number 417)

The CATCLOSE intrinsic closes the specified application message catalog and must be used with the application message facility.

### Syntax

```
             D      LA
CATCLOSE (catindex,error)
```

### Parameters

*catindex* — *double by value (required)*
The catalog index returned by the CATOPEN intrinsic.

*error* — *logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 | Close of catalog file failed. |
| 100 | Internal Message facility error. |

## Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to Program L in Appendix H, "Example Programs" for an example of how this intrinsic is used.

## CATOPEN (Intrinsic Number 415)

The CATOPEN intrinsic opens the specified application message file and must be used with the application message facility.

### Syntax

```
    D                          BA         LA
catindex:=CATOPEN (formaldesignator,error);
```

### Functional Returns

A catalog index double (an internal value recognized by the CATREAD and CATCLOSE intrinsics) is returned; this is not a file number.

### Parameters

*formaldesignator* — *byte array (required)*
Contains a string of USASCII characters that identify the catalog file for the system. This string must be terminated by any USASCII special character except a slash or period.

*error* — *logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 | Open failed on catalog file. |
| 2 | Could not access catalog file. |
| 3 | File specified is not a GENCAT formatted catalog. |
| 100 | Internal message facility error. |

### Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to Program L in Appendix H, "Example Programs" for an example of how this intrinsic is used.

# CATREAD

## CATREAD (Intrinsic Number 416)

The CATREAD intrinsic reads the specified catalog and returns the text as indicated; it accesses catalogs opened by the CATOPEN intrinsic only. The CATREAD intrinsic provides access to the application message facility. The NLS application message catalog facility is discussed in Chapter 2, "Application Message Facility."

### Syntax

```
  I                 D         IV        IV      LA   BA     IV
msglen:=CATREAD (catindex,setnum,msgnum,error,buff,buffsize,
                   BA     BA     BA     BA      BA       IV          O-V
                 parm1,parm2,parm3,parm4,parm5,msgdest);
```

### Functional Returns

The length of the message is returned to *msglen*.

### Parameters

*catindex*

*double by value (required)*
An index, returned by CATOPEN, specifying which catalog is to be used.

*setnum*

*integer by value (required)*
A positive integer, no greater than 255, specifying the set number within the catalog.

*msgnum*

*integer by value (required)*
A positive integer, no greater than 32766, specifying the message number within the message set.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 | Invalid *catindex*specified. |
| 2 | Read failed on catalog file. |
| 3 | Set not found. |
| 4 | Message not found. |
| 6 | User buffer overflow. |
| 7 | Write failed to *msgdest*file. |
| 14 | Set < = 0 specified. |
| 15 | Set > 255 specified. |
| 16 | Message number < 0 specified. |
| 17 | Message number > 32766 specified. |
| 18 | Specifies *buffsize* < = 0. |
| 19 | Specifies *msgdest* < 0. |
| 100 | Internal message facility error. |

*buff*

*byte array (optional)*
Where the assembled message is returned.

*buffsize*

*integer by value (optional)*
When specified, this is the buffer length in bytes. If *buff* is not specified, this is the length (in bytes) of the records to be written to the destination file (Default = 72 bytes).

*parm1 - parm5*

*byte arrays (optional)*
Parameters to be inserted into message. These must always point to a character string. The strings must be terminated by a binary zero.

*msgdest*

*integer by value (optional)*
Integer value specifying the destination of the assembled message (0 = $STDLIST, >2 = file number of destination file. Default = $STDLIST if *buff* and no file is specified).

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

Refer to Program L in Appendix H, "Example Programs" for an example of how this intrinsic is used.

# NLAPPEND

## NLAPPEND (Intrinsic Number 412)

The NLAPPEND intrinsic allows an application to designate which of several language-dependent files (for example, application message catalogs or VPLUS/3000 forms files) should be used by appending the language ID number to the filename. (This assumes that the application uses this naming convention for its language-dependent files.)

### Syntax

```
                  BA              IV      LA
NLAPPEND (formaldesignator,langnum,error);
```

### Parameters

*formaldesignator* — *byte array (required)*
Contains a string of USASCII characters interpreted as part of a formal file designator. The filename must end with three blanks.

*langnum* — *integer by value (required)*
The language ID number, specifying which catalog is to be opened.

*error* — *logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid filename. |
| 4 | File name not terminated by three blanks. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

### Special Considerations

Split-stack calls are not permitted.

## NLCOLLATE (Intrinsic Number 402)

The NLCOLLATE intrinsic collates two character strings according to the collating sequence of the specified language ID number. Its purpose is to determine a lexical ordering. It is not intended to be used for searching or matching. To determine if two strings are equal, use the COMPARE BYTES machine instruction.

### Syntax

```
           BA      BA      IV    I       IV     LA      LA      O-V
NLCOLLATE (string1,string2,length,result,langnum,error,collseq);
```

### Parameters

*string1*
*byte array (required)*
The first of two character strings to be collated.

*string2*
*byte array (required)*
The second of two character strings to be collated.

*length*
*integer by value (required)*
The length (in bytes) of the string segments to be collated.

*result*
*integer by reference (required)*
The result of the collated character string:

0 If *string1* collates equal to *string2*.
-1 If *string1* collates before *string2*.
1 If *string1* collates after *string2*.

Result will be 0 if a nonzero error is returned.

*langnum*
*integer by value (required)*
The language ID number, specifying which collating sequence is to be used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid collating table entry. |
| 4 | Invalid *length* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 * | Invalid collation range table. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*collseq*

*logical array (optional)*
An array containing the native language collating sequence table as returned by NLINFO, item 11. This parameter is required for split-stack calls. If this parameter is present, *langnum* will be ignored and this routine will be more efficient.

If the *collseq* parameter is omitted, and *langnum* is specified or defaults to a language which collates by binary encoding, the COMPARE BYTES machine instruction will be used to compare the two indicated strings. If the *collseq* parameter is used, it will determine the string compare operation (this may be a COMPARE BYTES). Refer to the NLINFO intrinsic items 11 and 27.

## Special Considerations

Split-stack calls are permitted.

## NLCONVCLOCK (Intrinsic Number 409)

The NLCONVCLOCK intrinsic checks validity of the string by using the formatting template returned by NLINFO item 3, then converts the time to the general time format returned by the CLOCK intrinsic. This intrinsic is the inverse of NLFMTCLOCK.

### Syntax

```
 D                    BA       IV        IV       LA
time:=NLCONVCLOCK (string,stringlen,langnum,error);
```

### Functional Returns

The intrinsic returns the time in the format:

```
Bits 0          7 8                 15
    +-----------+-----------------+
    | Hour of Day |   Minute of Hour  |
    +-----------+-----------------+
    |   Seconds   | Tenths of Seconds |
    +-----------+-----------------+
```

**NOTE**

Seconds and tenths of seconds will always be zero.

### Parameters

*string*

*byte array (required)*
A character string containing the time to be converted.

*stringlen*

*integer by value (required)*
A positive integer specifying the length of the string (in bytes).

*langnum*

*integer by value (required)*
The language ID number, specifying which custom time format is to be matched by the string.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid time string. |
| 4 | Invalid length. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

### Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to Programs D and E in Appendix H, "Example Programs" for examples of how this intrinsic is used.

## NLCONVCUSTDATE (Intrinsic Number 408)

Checks the validity of a string by using the formatting template returned by NLINFO item 2, then converts the date to the general date format as returned by the CALENDAR intrinsic. This intrinsic is the inverse of NLFMTCUSTDATE.

### Syntax

```
 L                      BA     IV        IV     LA
date:=NLCONVCUSTDATE (string,stringlen,langnum,error);
```

### Functional Returns

The intrinsic returns the date in the format:

```
Bits  0               6 7           15
     +----------------+-------------+
     | Year of Century | Day of Year  |
     +----------------+-------------+
```

### Parameters

*string*

*byte array (required)*
A character string containing the date to be converted. Leading and trailing blanks will be disregarded.

*stringlen*

*integer by value (required)*
A positive integer specifying the length of the string (in bytes).

*langnum*

*integer by value (required)*
The language ID number, specifying which custom date format is to be matched by the string.

*error*

*logical array (required)*

The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid date string. |
| 4 | Invalid string length. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Separator character in *string* does not match separator in the custom date template. |
| 8 | The length of the date string is more than 13 characters (excluding leading and trailing blanks). |
| 9 * | Invalid national special table defined. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

### Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to Programs D and E in Appendix H, "Example Programs" for examples of how this intrinsic is used.

## NLCONVNUM (Intrinsic Number 419)

Converts native language numbers with native decimal and thousands separators (for example, 1.234,56) to an ASCII number with NATIVE-3000 decimal separator (.) and thousands separators (,). As an option, the decimal and thousands separators can be stripped.

### Syntax

```
                IV        BA          IV        BA
NLCONVNUM (langnum,instring,inlength,outstring,
                I         LA      LA       LV        I            O-V
          outlength,error,numspec,fmtmask,decimals);
```

### Parameters

*langnum*
*integer by value (required)*
The language ID number, specifying which numeric formatting rules are to be used in the conversion.

*instring*
*byte array (required)*
Contains the native language formatted number to be converted. Leading and trailing spaces are ignored.

*inlength*
*integer by value (required)*
Length, in characters, of *instring*.

*outstring*
*byte array (required)*
Contains the converted output. The output will be left justified in the buffer and *outlength* will contain the actual length of the converted number. *Outstring* may reference the same address as *instring*.

*outlength*
*integer (required)*
Length, in characters, of *outstring*. After a successful call to NLCONVNUM, *outlength* will contain the actual length of the converted number.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid length specified (*inlength*or *outlength*). |
| 4 | Invalid number specified (*instring*). |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Truncation has occurred (*outstring*is left partially formatted). |
| 8 | Invalid *numspec* parameter. |
| 9 | Invalid *fmtmask* parameter. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*numspec*

*logical array (optional)*
A byte array, returned from NLNUMSPEC, which contains formatting information. If this parameter is present, *langnum* will be ignored, and performance will be improved (refer to the description of NLNUMSPEC in this chapter).

*fmtmask*

*logical by value (optional)*
Specifies how to format the number. The default value is 0 and indicates substitution only.

| Bit # | Description |
|---|---|
| (15:1) | 0 - Convert thousands separators.<br>1 - Strip thousands separators. |
| (14:1) | 0 - Convert decimal separators.<br>1 - Strip decimal separators. |
| (13:1) | 0 - *instring* can contain any character. (No validation will be performed)<br>1 - *instring* contains a number. (Validation will be performed) |
| (0:13) | Reserved. Should always be set to zero. |

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

This intrinsic converts a native language formatted number to an ASCII number with the NATIVE-3000 decimal separator (.) and thousands separator (,) for use in further conversion to INTEGER, REAL, etc. This intrinsic will convert the decimal and thousands separators, or strip them (see *fmtmask*), to the NATIVE-3000 equivalent. For languages using an alternate set of digits (Arabic, HINDI digits only), the intrinsic will convert the digits to ASCII for recognition and use as numeric characters.

# NLFINDSTR

## NLFINDSTR (Intrinsic Number 429)

This intrinsic searches *string2* for *string1*, and returns an integer value indicating the offset in *string2* where *string1* was found.

### Syntax

```
I                    IV      BA      IV      BA
offset:=NLSUBSTR (langnum,string1,length1,string2,
                   IV     LA     LA      O-V
                  length2,error,charset);
```

### Functional Returns

A -1 is returned if *string1* is not found in *string2*.

### Parameters

*langnum* — *integer by value (required)*
The language ID number.

*string1* — *byte array (required)*
The string of characters to be searched. It can contain one-byte and two-byte Asian characters.

*length1* — *integer by value (required)*
Length, in characters, of *string1*.

*string2* — *byte array (required)*
The character string to be searched for.

*length2* — *integer by value (required)*
Length, in characters, of *string2*.

*error*

*logical array (required)*
In the first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid *length1* parameter. |
| 4 | Invalid *length2* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*charset*

*logical array (optional)*
Contains the character set definition for the language to be used, as returned by NLINFO's item 12.

## Special Considerations

Split-stack calls are not permitted.

# NLFMTCALENDAR

## NLFMTCALENDAR (Intrinsic Number 413)

Formats the date as specified by the language-dependent calendar which is returned by NLINFO item 1.

### Syntax

```
                LV    BA      IV      LA      O-V
NLFMTCALENDAR (date,string,langnum,error);
```

### Parameters

*date*

*logical by value (required)*
Indicates the date, in the format, as returned by the CALENDAR intrinsic:

```
Bits  0                6 7           15
      +----------------+-------------+
      | Year of Century | Day of Year  |
      +----------------+-------------+
```

*string*

*byte array (required)*
A character string in which the formatted date is returned. This string will be 18 characters long, padded with blanks if necessary.

*langnum*

*integer by value (required)*
The language ID number, specifying which calendar template is to be used. A *langnum* of 0 will return the date formatted as though FMTCALENDAR were used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid date value. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

### Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to Programs D and E in Appendix H, "Example Programs" for examples of how this intrinsic is used.

## NLFMTCLOCK (Intrinsic Number 410)

The NLFMTCLOCK intrinsic formats the time of day, as returned by the CLOCK intrinsic, to the custom time of day format specified for the native language. The template returned by NLINFO item 3 will be used.

### Syntax

```
             DV     BA      IV       LA
NLFMTCLOCK (time,string,langnum,error);
```

### Parameters

*time*

*double by value (required)*
A double word value, containing the time, in the format returned by the CLOCK intrinsic:

```
Bits 0           7 8                 15
    +-------------+-------------------+
    | Hour of Day |   Minute of Hour  |
    +-------------+-------------------+
    |   Seconds   | Tenths of Seconds |
    +-------------+-------------------+
```

*string*

*byte array (required)*
An eight-character byte array, containing the formatted time of day which is returned.

*langnum*

*integer by value (required)*
The language ID number, specifying which format is to be used. A *langnum* of 0 will return the time formatted as though FMTCLOCK were used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid time format. |
| 4 * | NLS internal error. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

Refer to Programs D and E of Appendix H, "Example Programs" for examples of how this intrinsic is used.

## NLFMTCUSTDATE (Intrinsic Number 407)

The NLFMTCUSTDATE intrinsic formats the date, as returned by the CALENDAR intrinsic, to the custom date format for the specified native language. The template NLINFO item 2 will be used.

### Syntax

```
              LV    BA     IV    LA
NLFMTCUSTDATE (date,string,langnum,error);
```

### Parameters

*date*

*logical by value (required)*
A logical value, containing the date, in the format returned by the CALENDAR intrinsic:

```
Bits 0              6 7           15
     +----------------+-------------+
     | Year of Century | Day of Year |
     +----------------+-------------+
```

*string*

*byte array (required)*
A thirteen-character byte array, containing the formatted date which is returned.

*langnum*

*integer by value (required)*
The language ID number, specifying which custom date template is to be used for formatting. A *langnum* of 0 will return the time formatted as though FMTCLOCK were used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid date value. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

### Special Considerations

Split-stack calls are not permitted.

### Additional Discussion

Refer to examples D and E in Appendix H, "Example Programs" for examples of how this intrinsic is used.

# NLFMTDATE

## NLFMTDATE (Intrinsic Number 414)

The NLFMTDATE intrinsic formats the specified date and time according to the concatenation of the templates returned by NLINFO items 1 and 3.

### Syntax

```
           LV   DV    BA      IV     LA
NLFMTDATE (date,time,string,langnum,error);
```

### Parameters

*date*

*logical by value (required)*
A logical value indicating the date in the format as returned by the CALENDAR intrinsic:

```
Bits  0                6 7           15
     +----------------+-------------+
     | Year of Century | Day of Year |
     +----------------+-------------+
```

*time*

*double by value (required)*
A double word value indicating the time to be formatted. The double word is in the format returned by the CLOCK intrinsic:

```
Bits 0           7 8                15
    +------------+------------------+
    | Hour of Day |  Minute of Hour  |
    +------------+------------------+
    |   Seconds   | Tenths of Seconds |
    +------------+------------------+
```

*string*

*byte array (required)*
A 28-character string in which the formatted date and time are returned.

*langnum*

*integer by value (required)*
The language ID number, specifying which formatting templates are to be used. A *langnum* of 0 will return the date/time string as though FMTDATE were used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid date value. |
| 4 | Invalid time value. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

Refer to Program K in Appendix H, "Example Programs" for examples of how this intrinsic is used.

# NLFMTLONGCAL

## NLFMTLONGCAL (Intrinsic Number 420)

The NLFMTLONGCAL intrinsic formats the supplied date according to the long calendar format. The formatting is done according to the template returned by NLINFO item 30.

### Syntax

```
              LV   BA    IV      LA
NLFMTLONGCAL (date,string,langnum,error)
```

### Parameters

*date*

*logical by value (required)*
A logical value containing a date in the format as returned by the CALENDAR intrinsic:

```
Bits  0               6 7            15
      +---------------+---------------+
      | Year of Century | Day of Year  |
      +---------------+---------------+
```

*string*

*byte array (required)*
A 36 character array to which the formatted long calendar date is returned, padded with blanks if necessary.

*langnum*

*integer by value (required)*
The language ID number, specifying which format is to be used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid date format. |
| 4 * | NLS internal error. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE/3000).

### Special Considerations

Split-stack calls are not permitted.

## NLFMTNUM (Intrinsic Number 421)

The NLFMTNUM intrinsic converts a string, containing an ASCII number (may include NATIVE/3000 decimal separator (.), thousands separator (,), and currency symbol/name ($)), to a language specific format using the currency symbol/name, decimal separator, and thousands separators defined for the native language.

### Syntax

```
                  IV      BA       IV       BA
NLFMTNUM (langnum,instring,inlength,outstring,
                  I       LA     LA      LV      IV
          outlength,error,numspec,fmtmask,decimals);
                                                   O-V
```

### Parameters

*langnum* — *integer by value (required)*
The language ID number, specifying which formatting specifications are to used.

*instring* — *byte array (required)*
A byte array containing the NATIVE-3000 formatted ASCII number to be converted (for example, $-123,456.78). Leading and trailing spaces are allowed.

*inlength* — *integer by value (required)*
Length, in characters, of instring.

*outstring* — *byte array (required)*
A byte array where the language specific formatted number will be returned. The decimal separator, thousands separator, and currency symbol/name are replaced, if present; or are inserted, if specified by *fmtmask*, according to the language definition. The *outstring* may reference the same address as *instring*.

*outlength* — *integer (required)*
Length, in characters, of *outstring*. After a successful call, if *outstring* is returned left-justified (specified by *fmtmask*), *outlength* will return the actual length, in characters, of the formatted number.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid length specified. (*inlength* or *outlength*) |
| 4 | Invalid number specified. (*instring*) |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Truncation has occurred. (*outstring* is left partially formatted) |
| 8 | Invalid *numspec* parameter. |
| 9 | Invalid *fmtmask* parameter. |
| 10 | Invalid *decimals* parameter. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE/3000).

*numspec*

*logical array (optional)*
A byte array, as returned from NLNUMSPEC, containing formatting specifications for the specified language (currency/name, decimal separator, etc.) If this parameter is present, *langnum* will be ignored, and performance will be improved. See description of NLNUMSPEC.

*fmtmask*

*logical by value (optional)*
A logical specifying any formatting to be done on the input. The default value is 0, which means a simple substitution.

| Bit # | Description |
|---|---|
| (15:1) | 0 - Do not insert thousands separator.<br>1 - Insert thousands separators. |
| (14:1) | 0 - Do not insert decimal separators.<br>1 - Insert decimal separators. |
| (13:1) | 0 - Do not insert currency symbol/name.<br>1 - Insert currency symbol/name. |
| (11:2) | 0 - No justification of the output.<br>1 - The output will be left-justified.<br>2 - The output will be right-justified.<br>3 - The output will be left-justified and *outlength* will return the actual length of the formatted number. |
| (0:11) | Reserved. Should always be set to zero. |

*decimals*

*integer by value (optional)*
An integer specifying where to insert the decimal separator. The value is ignored if bit 14 of *fmtmask* is zero, or a decimal separator is present in the number.

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

This intrinsic operates in substitution mode and formatting mode:

### Substitution Mode

If *fmtmask* is omitted or has all bits set to zero, the substitution mode will substitute the native equivalent for ('.') and (','); for Arabic, it will substitute the alternative set of digits for ASCII digits. The input is not validated as a number, and can contain several numbers. No justification takes place, and the output will be left truncated if *outstring* is shorter than *instring* (for example, 1,234.56 -> .234,56).

### Formatting Mode

If any bit 10-15 in *fmtmask* is set to one, the formatting mode will perform the substitution, and format the input according to *fmtmask*. In this mode, input is validated as a number, and only ASCII digits and '.', ',', '-', '+', and '$' are allowed.

Only one sign and one '$' are allowed. They must be the first character(s) in *instring*. Even if insertion (of thousands separators etc.) is specified in *fmtmask*, the thousands and decimal separators are still valid characters in the input. In this case, they will be substituted. If no justification is specified, the output will be right-justified with the same number of trailing spaces as the input. If the output is truncated, it will be left-truncated

**NOTE**

For languages written right to left, trailing spaces in the input will be preserved as leading spaces in the output.

# NLGETLANG

## NLGETLANG (Intrinsic Number 411)

This intrinsic returns a language ID number which characterizes the current user, data, or system. It is intended for use by Hewlett-Packard subsystems (programs, not intrinsics) or by applications programs so they can automatically configure themselves. Refer to "Special Considerations" for a description of where NLGETLANG derives its information.

### Syntax

```
   I                        IV      LA
langnum:=NLGETLANG (function,error);
```

### Functional Returns

The language ID number (*langnum*) of the current user, data, or system. In the event of an error, an integer value of 0 (NATIVE-3000) is always returned to *langnum*.

### Parameters

*function*

*integer by value (required)*
The function number indicating which language ID number should be returned. The possible values are:

1. The user-interface language. This is used to specify the language to be used for communication between the program and the user.

2. The data language. This is an attribute which determines how various language-dependent data manipulation functions (for example, sorting or upshifting) should be performed by the subsystem.

3. The system default language.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 | NLS is not installed. |
| 2 | NLGETLANG found the language requested,<br>but it was not configured on the system. |
| 3 | Invalid *funtion* value. |
| 4 | No language specified for NLGETLANG to access. |

## Special Considerations

Split-stack calls are not permitted.

The NLGETLANG intrinsic will locate the language ID numbers requested by *function* 1 and 2 by referring to the Hewlett-Packard defined Job Control Words (JCWs) NLUSERLANG and NLDATALANG respectively. If the required JCW does not exist, or has a value greater than or equal to FATAL (32768), Error #4 is returned.

## Additional Discussion

For example calls of this intrinsic refer to Program K in Appendix H, "Example Programs."

# NLINFO

## NLINFO (Intrinsic Number 400)

This intrinsic returns language-dependent information.

### Syntax

```
            IV          LA       I     LA
NLINFO (itemnumber,itemvalue,langnum,error);
```

### Parameters

*itemnumber* — *integer by value (required)*
Positive integer which specifies the *itemvalue* to return.

*itemvalue* — *type of variable depends on itemnumber (required)*
Return variable for information requested; or (if *itemnumber* is 22 or 24) the language name or number about which information is requested.

The following is a list of the currently defined *itemnumbers*, and the data types and information returned to *itemvalue*.

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 1 | LA | An 18-character array to which the calendar format is returned. The 18 characters of the string for this definition are interpreted as the format description for that language. The following descriptors are valid: |

| | |
|---|---|
| D | One-character day abbreviation. |
| DD | Two-character day abbreviation. |
| DDD | Three-character day abbreviation. |
| M | One-character month abbreviation. |
| MM | Two-character month abbreviation. |
| MMM | Three-character month abbreviation. |
| MMMM | Four-character month abbreviation. |
| mm | Numeric month of the year. |
| dd | Numeric day of the month. |
| yy | Numeric year of the century. |
| yyyy | Numeric year. |
| Nyy | National year. |
| NPyy | National year which may include a before-period symbol. |
| E | 1-8 of these are to be replaced by that many characters from the Emperor/Country name. |

Valid separators are any special character.

For example, a format may be: `DDD, MMM dd, yyyy`. Using this format in NATIVE-3000 would result in: `FRI, MAY 25, 1984`.

**Item #** **Type** **Description of *itemvalue***

2 LA

A 13-character array to which the custom date format is returned. The 13 characters of the string for this definition are interpreted as the custom date format description.

The following descriptors are valid:

| | |
|---|---|
| mm | Numeric month of the year. |
| dd | Numeric day of the month. |
| yy | Numeric year of the century. |
| yyyy | Numeric year. |
| Nyy | National year. |
| NPyy | National year which may include the before-period symbol. |

Valid separators are any special character. For instance, a date format might be: yy/mm/dd. An example of this format in NATIVE-3000: 81/03/25.

3 LA

An eight-character array to which the clock specification is returned. This eight-character string provides the clock format description (template):

HHSXXYYZ, where:

| | |
|---|---|
| HH | Clock hour specification, either 12 or 24. |
| S | Separator. Valid separators may be any special or alpha character, or 0 if no sparators between hours and minutes should appear. |
| XX | Symbol for AM. |
| YY | Symbol for PM. |
| Z | If blank, supresses leading zeros (hours); if zero (0), prints leading zero. |

In suppression of leading zero, " " (leading zero suppressed) or "0" (leading zero will be printed) are valid. For example, the format "12:AMPM " would yield formatted clock information in the form: 9:06 AM. The leading zero is suppressed.

If the clock specification were changed to"240 0", the formatted clock information for the same time would be:0906. Note the four blanks used as place holders to ensure the correct placement of the leading-zero suppression character.

4 LA

A 48-character array to which the month abbreviation table is returned. Each abbreviation is four characters long, using blank padding where necessary to maintain uniform length in all native language abbreviations. For example, the NATIVE-3000 abbreviations contain three characters plus a blank. The first four characters of the array contain the abbreviation of January.

The month abbreviation table for NATIVE-3000 would be:

```
"JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC"
```

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 5 | LA | A 144-character array in which the month table is returned. Each month's name can be up to 12 characters long. Unused space in each month name is padded with blanks where necessary to equal 12 characters. The table begins with the language-dependent equivalent in the native language specified for January.<br>For example, the month name table for NATIVE-3000 would be:<br>`"JANUARY FEBRUARY MARCH ...DECEMBER"` |
| 6 | LA | A 21-character array in which the day abbreviation table is returned. Each abbreviation is three characters long. The table begins with Sunday.<br>For example, the day abbreviation table for NATIVE-3000 would be:<br>`"SUNMONTUEWEDTHUFRISAT"` |
| 7 | LA | An 84-character array in which the table containing the day of the week is returned. Each day is 12 characters long (with blank padding as needed). The table starts with Sunday.<br>For example, the day name table for NATIVE-3000 would be:<br>`"SUNDAY MONDAY TUESDAY ...SATURDAY"` |
| 8 | LA | A 12-character array to which the YES/NO responses are returned. The first six characters contain the (upshifted) "YES" response; the second six the (upshifted) "NO" response. |
| 9 | LA | A two-character array to which the symbols for decimal separator and thousands indicator are returned. The first character contains the decimal separator, the second contains the thousands indicator.<br>The character for the thousands separator may take a special value: '0' (zero). This value is not to be taken literally as a thousands separator, but signifies the absence of a thousands separator for the language chosen. |

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 10 | LA | A six-character array to which the currency signs are returned. The first character represents the short currency symbol (if any) used for business formats; the second character is a flag that indicates whether the currency symbol precedes or succeeds the number and whether the currency symbol is preceded or succeeded by blanks. The last four characters contain the full currency symbol. The layout of the second character is as follows:<br>bits 0:4 0 The currency symbol has no blanks preceding or succeeding it.<br>1 The currency symbol has a blank preceding it.<br>2 The currency symbol has a blank succeeding it.<br>3 The currency symbol has blanks preceding and succeeding it.<br>bits 4:4 0 The currency symbol precedes the number.<br>1 The currency symbol succeeds the number.<br>2 The currency symbol replaces the decimal separator.<br>3 The currancy symbol precedes the sign (if present). |
| 11 | LA | An array to which the collating sequence table is returned. A call to NLINFO item 27 determines the length of this array based on the length of the table of the native language specified. |
| 12 | LA | A 256-character array to which the character set attribute table is returned. Each character will contain the numeric identification of the character type:<br>0 Numeric character.<br>1 Alphabetic lowercase character.<br>2 Alphabetic uppercase character.<br>3 Undefined graphic character.<br>4 Special character.<br>5 Control code.<br>6 First byte of a two-byte character. |
| 13 | LA | A 256-character array to which the ASCII-to-EBCDIC translation table is returned. |
| 14 | LA | A 256-character array to which the EBCDIC-to-ASCII translation table is returned. |
| 15 | LA | A 256-character array to which the upshift table is returned. |
| 16 | LA | A 256-character array to which the downshift table is returned. |

# NLINFO

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 17 | LA | A logical array to which the language numbers of all configured languages are returned. The first word of this array contains the number of configured languages. The second word contains the language number of the first configured language. The third word contains the language number of the second configured language, etc. (The *langnum* parameter is disregarded.) |
| 18 | L | A logical to which true (-1) is returned if the specified language is supported (configured) on the system. Otherwise, false (0) is returned. |
| 19 | I | An integer to which the character set ID number supporting the specified language is returned. |
| 20 | LA | A 16-character array to which the uppercase name of the character set supporting the specified language is returned. If the name contains fewer than 16 characters, it will be padded with blanks. |
| 21 | LA | A 16-character array to which the uppercase name of the specified language is returned. If the name contains fewer than 16 characters, it will be padded with blanks. |
| 22 | LA | The *itemvalue* is a logical array containing a language name or number (in ASCII digits) terminated by a blank. The array must be at least eight words in length. The associated language ID number will be returned to *langnum.* |
| 23 | L | A logical to which true (-1) is returned if the character set specified is supported (configured) on the system. Otherwise, false (0) is returned. |
| 24 | LA | The *itemvalue* is a logical array containing a character set name or number (in ASCII digits) terminated by a blank. The required length of this array is eight words or more. The associated character set ID number will be returned to *langnum.* |
| 25 | LA | A 16-character array to which the uppercase name of the specified character set is returned. The *langnum* parameter must contain the ID number of the character set. If the name contains fewer than 16 characters, it will be padded with blanks. |
| 26 | I | An integer to which the class number of the specified language is returned. |
| 27 | I | An integer to which the length (in words) of the collating sequence table of the specified language is returned. |
| 28 | I | An integer to which the length (in words) of the national-dependent information table is returned. If no national table exists for the specified language, Error #4 is returned. |

**Item #** **Type** **Description of *itemvalue***

29 LA

A logical array to which the national-dependent information table is returned. To determine the size of this array, the length must first be obtained with a call to NLINFO item 28.

30 LA

A 36 character array to which the long calendar format is returned. It may contain arbitrary text, as well as the following descriptors:

| | |
|---|---|
| D | 1-3 of these are to be replaced by that many characters from the day abbreviation. |
| W | 1-12 of these are to be replaced by that many characters from the day of the week. |
| dd | Numeric day of month. |
| M | 1-4 of these are to be replaced by that many characters from the month abbreviation. |
| O | 1-12 of these are to be replaced by that many characters from the month of the year. |
| mm | Numeric month of the year. |
| yy | Numeric year of the century. |
| yyyy | Numeric year of the century. |
| Nyy | National year. |
| NPyy | National year which may include a before-period symbol. |
| E | 1-8 of these are to be replaced by that many characters from the Emperor/Country name. |

In addition, a special literal character "~" may be used to indicate that the following character should be taken literally in the format, even if it is one of the special characters above.

For example, a format may be: WWWWWWWWW, OOOOOOOOO dd, A.~D.yyyy. Using this format in NATIVE-3000 would result in: WEDNESDAY, NOVEMBER 21, A.D. 1984.

31 LA

A 16 character array to which the currency name is returned.

32 LA

An 8 character array, containing information about an alternative set of digits. (Currently only used by Arabic)

| Byte | Description |
|---|---|
| 0-1 | Alternative digit separator (Integer).<br>0 - No Alternative digits defined.<br>1 - Alternative digits defined. |
| 2 | The Alternative digit '0'. |
| 3 | The Alternative digit '9'. |
| 4 | The '+' used with Alternative digits. |
| 5 | The '-' used with Alternative digits. |
| 6 | The decimal separator used with Alternative digits. |
| 7 | The thousands separator used with Alternative digits. |

# NLINFO

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 33 | LA | A 4 character array, containing information about the direction of the language. |

| Byte | Description |
|---|---|
| 0-1 | Language direction (Integer)<br>0 - Direction is 'left to right'.<br>1 - Direciton is 'right to left'. |
| 2 | The 'right to left' space. |
| 3 | Undefined. |

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 34 | L | A logical value which returns the data ordering of the language. |

| Byte | Description |
|---|---|
| 0 | Keyboard order. |
| 1 | Left-to-right screen order. |
| 2 | Right-to-left screen order. |

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 35 | L | A logical value which returns the size of the character used by the language. |

| Byte | Description |
|---|---|
| 0 | One-byte characters (8-bits). |
| 1 | Two-byte characters (16-bits). |

| Item # | Type | Description of *itemvalue* |
|---|---|---|
| 36 | L | A logical value that returns a true (1) if the language requires suppressing the leading zero or blank in the date format. |

*langnum*

*integer by reference (required)*
The language or character set identification number for the information requested.

*error*

*logical array (required)*
This two-word array contains the error number in the first word. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 * | Specified character set is not configured. |
| 4 | No national table is present. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7-9 | Reserved. |
| 10 | The *itemnumber* is out of range. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

## Special Considerations

Split-stack calls are permitted.

## Additional Discussion

"Alternative digits" exist for the convenience of Arab speaking cultures that use Hindi digits in place of the Arabic digits (0..9), which are more familiar to European and American users. For example calls of this intrinsic refer to Program H in Appendix H, "Example Programs."

# NLJUDGE

## NLJUDGE (Intrinsic Number 427)

This intrinsic judges whether a character is a one-byte or two-byte Asian character. If it is a two-byte character, set *judgeflag* to 1 or 2. If it is a one-byte character, set *judgeflag* to 0.

### Syntax

```
 IV                    IV       BA          IV        BA
N2bytes:=NLJUDGE (langnum,instring,stringlength,judgeflag,
                   LV       LA       O-V
                   error, charset);
```

### Functional Returns

The number of a two-byte Asian character is an integer value that can be used to check if a string of characters contain Asian characters.

### Parameters

*langnum*

*integer by value (required)*
The language ID number.

*instring*

*byte array (required)*
The string of characters to be judged.

*stringlength*

*integer by value (required)*
An integer value specifying the number of bytes in the *instring*.

*judgeflag*

*byte array (required)*
This string will contain the flag values as follows:

| | |
|---|---|
| 0 | One-byte character. |
| 1 | First byte of a two-byte character. |
| 2 | Second byte of a two-byte character. |
| 3 | Invalid Asian character. |

*error*

*logical array (required)*
In the first word, of this two-word array, the error number will be returned. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid string length. |
| 4 | Not returned. |
| 5 * | Bad NLT extra data segment. |
| 6 * | Bad LDST extra data segment. |
| 7 * | Invalid characters found in *instring*. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*charset*

*logical array (optional)*
An array containing the character set definition for the language to be used, as returned by NLINFO's item 12. If present, the *langnum* parameter will be ignored, and this routine will be more efficient.

## NLKEYCOMPARE (Intrinsic Number 405)

Compares two strings of different length. This intrinsic gives the KSAM/3000 user the ability to determine whether the key of a record matches the generic key specified. It should be used when reading a KSAM/3000 file in key sequential order in combination with FREAD, after a FFINDBYKEY call.

The NLKEYCOMPARE intrinsic allows a program to determine whether a generic key search found an exact match. That is, the generic key must exactly equal the beginning of the key, and not almost equal because of priority (for example, uppercase versus lowercase or accent). It also allows the program to determine whether an exactly matching key could be farther along the key sequence.

### Syntax

```
              BA      IV      BA    IV
NLKEYCOMPARE (genkey,length1,key,length2,
              I       IV      LA    LA      O-V
              result,langnum,error,collseq);
```

### Parameters

*genkey*
*byte array (required)*
Contains the generic key to be compared to the keys contained in the record read by FREAD.

*length1*
*integer by value (required)*
The length in bytes of *genkey*, which must be less than *length2*.

*key*
*byte array (required)*
This contains an entire key to which the user wants to compare *genkey*.

*length2*
*integer by value (required)*
The length in bytes of *key*, which must be greater than *length1*.

*result*

*integer by reference (required)*
The result of the compare:

| | |
|---|---|
| 0 | The retrieved key matches the generic key exactly for a length of *length1.* |
| 1 | The retrieved key does not match the generic key: it is different only because of priority (for example, uppercase versus lowercase characters or accent). The FREAD key is still in range. This means that records may follow whose key matches the generic key exactly. |
| 2 | The retrieved key is less than the generic one (its collating order precedes the key specified). It does not match *genkey*. This means the FREAD call found a record which precedes the range requested. Records which match *genkey* may follow. |
| 3 | The retrieved key is greater than the generic key (it collates after the specified key). This means that the FREAD call found a record whose key follows the specified range. No records matching *genkey* follow. |

*langnum*

*integer by value (required)*
The language ID number, specifying the collating sequence to be used for the compare.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid collating table entry. |
| 4 | Invalid *length* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Value of *length1* is not less than *length2.* |
| 8 * | Invalid collation range table. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*collseq*

*logical array (optional)*
An array containing the collating sequence table as returned by NLINFO item 11. This parameter is required for split-stack calls. If this parameter is present, *langnum* will be ignored and this routine will be much more efficient.

# NLKEYCOMPARE

## Special Considerations

Split-stack calls are permitted.

NLKEYCOMPARE is intended for use with the KSAM/3000 subsystem.

## Additional Discussion

For example calls of this intrinsic refer to Programs I and J in Appendix H, "Example Programs."

## NLNUMSPEC (Intrinsic Number 425)

The intrinsic returns the information needed for formatting and converting numbers. It combines several calls to NLINFO in order to simplify the use of native language formatting. By calling NLNUMSPEC once, and passing the obtained information to NLFMTNUM and NLCONVNUM, implicit calls to NLUMSPEC from NLFMTNUM and NLCONVNUM are avoided and performance improved.

### Syntax

```
            IV       LA     LA
NLNUMSPEC (langnum, string, error);
```

### Parameters

*langnum*

*integer by value (required)*
The language ID number.

*string*

*logical array (required)*
A byte array of minimum 60 bytes in which will be returned the following information :

| Byte | Description |
|---|---|
| 00-01 | Language identification number. (Integer) |
| 02-03 | Alternate Digit Indicator. (Integer) |
| | 0 - No Alternate digits exist. |
| | 1 - Alternate digits exist. |
| 04-05 | Language Direction Indicator. (Integer) |
| | 0 - The Language is 'left-to-right'. |
| | 1 - The Language is 'right-to-left'. |
| 06-07 | The Alternate digit range. ('0','9') |
| 08 | Decimal separator. ASCII-digits |
| 09 | Decimal separator. Alternate-digits |
| 10 | Thousands separator. ASCII-digits |
| 11 | Thousands separator. Alternate-digits |
| 12 | '+' Alternate-digits. |
| 13 | '-' Alternate-digits. |
| 14 | 'Right-to-left' space. |
| 15 | Reserved. |
| 16-17 | Currency place. (Integer) |
| | 0 - Currency symbol |
| | 1 - Currency symbol succeeds the number. |
| | 2 - Currency symbol replaces the decimal separator. |
| | 3 - Currency symbol precedes the sign. |
| 18-19 | Length of Currency Symbol. (Integer) (Including any spaces) |
| 20-37 | Currency symbol. (Including any spaces) |
| 38-59 | Reserved. |

*error*

*logical array (required)*

The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid string. |
| 4 | Not returned. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE/3000).

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

The intrinsic combines NLINFO calls with item numbers 9, 10, 31, 32, and 33. The information is formatted where needed (for example, any spaces in the currency symbol/name is included). The currency symbol/name is the shortest non-blank descriptor, as returned from NLINFO, items 10 and 31. Apart from the mentioned formatting, the intrinsic does not provide any information not obtainable with NLINFO. It is included for the convenience of the NLS user. For efficiency, the user of this intrinsic would presumably call it only once, save the result, and then call NLFMTNUM and/or NLCONVNUM multiple times.

## NLREPCHAR (Intrinsic Number 403)

This intrinsic replaces all nondisplayable control characters in the string with the replacement character. Nondisplayable characters are those with attribute 3 (undefined graphic character) or 5 (control code), as returned by NLINFO item 12.

### Syntax

```
            BA    BA       IV          BV
NLREPCHAR (instr,outstr,stringlength,repchar,
             IV       LA     LA       O-V
           langnum,error,charset);
```

### Parameters

*instr*

*byte array (required)*
A byte array in which the nondisplayable characters have to be replaced.

*outstr*

*byte array (required)*
A byte array to which the replaced character string is returned.

*stringlength*

*integer by value (required)*
A positive integer specifying the length (in bytes) of *instring*.

*repchar*

*byte value (required)*
A byte specifying the replacement character to be used.

*langnum*

*integer by value (required)*
The language ID number, specifying which character set is to be used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid replacement character. |
| 4 | Invalid *length* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Invalid charset table entry. |
| 8 | Overlapping strings, *outstring* would overwrite *instring*. |
| 9 * | Invalid two-byte character. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

| | |
|---|---|
| *charset* | *logical array (optional)*<br>Contains the character set definition for the language to be used, as returned in NLINFO item 12. If this parameter is present, *langnum* will be ignored and this intrinsic will be much more efficient. |

## Special Considerations

Split-stack calls are not permitted.

## Additional Discussion

For example calls of this intrinsic refer to Program H in Appendix H, "Example Programs."

## NLSCANMOVE (Intrinsic Number 401)

Moves and scans character strings according to character attributes. The machine instructions (and the SPL constructs) for SCAN and MOVE used for upshifting or in conjunction with the alphabetic, numeric, or special characters will only work for NATIVE-3000. This intrinsic will handle this function in a language-dependent manner.

### Syntax

```
  I                        BA       BA      LV    IV
numchar:=NLSCANMOVE (instring,outstring,flags,length,
                           IV      LA      LA      LA        O-V
                     langnum,error,charset,shift);
```

### Functional Returns

The number of characters acted upon in the SCAN or MOVE operation.

### Parameters

*instring* — *byte array (required)*
A character string which will act as the source string of the SCAN/MOVE.

*outstring* — *byte array (required)*
A character string which will act as the target.

**NOTE**

If *outstring* and *instring* are the same string, this intrinsic will act as SCAN. Otherwise, a MOVE will be performed. (Refer to Error #3.)

*flags*

*logical by value (required)*

A flag defining the options for calling the intrinsic. This parameter always defines the condition for terminating the SCAN/MOVE operation.

| Bits | Description |
|---|---|
| 14:2 | Alphabetic. NLINFO item 12, types 1 (alphabetic lowercase character) and 2 (alphabetic uppercase character).<br>1 - Lowercase.<br>2 - Uppercase.<br>3 - Uppercase or lowercase. |
| 13:1 | Numeric. NLINFO item 12, type 0. |
| 12:1 | Special. NLINFO item 12, types 3 (undefined graphic character), 4 (special character), or 5 (control code). |
| 11:1 | WHILE/UNTIL option. If this bit is zero, then SCAN/MOVE is performed while the condition specified by *flags* (12:4) is true. If this bit is one, SCAN/MOVE is performed until the condition specified by *flags* (12:4) is true. |
| 9:2 | Shift.<br>1 - Upshift.<br>2 - Downshift. |
| 7:2 | 0 or 3 SCAN. For/UNTIL one-byte and two-byte characters.<br>1 - Two-byte mode only.<br>2 - One-byte mode only. |
| 0:7 | Reserved. These bits of the *flags* parameter are reserved and must be zero. |

*length*

*integer by value (required)*

An integer indicating the maximum number of characters to be acted upon during the indicated operation.

*langnum*

*integer by value (required)*

The language ID number, specifying both the character set definitions of character attributes and the language-specific shift.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Overlapping strings; *instring* would have been overwritten by *outstring*. |
| 4 | Invalid *length* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Reserved portion of *flags* is not zero. |
| 8 | Both upshift and downshift requested. |
| 9 | Invalid table element. |
| 10 * | Invalid two-byte character. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*charset*

*logical array (optional)*
An array containing the character set definition for the language to be used, as returned in NLINFO item 12. If present, the *langnum* parameter will be ignored, and this routine will be much more efficient. This parameter is required for split-stack calls in which *flags* (12:4) is not equal to 0 and *flags* (12:4) is not equal to 15.

*shift*

*logical array (optional)*
An array containing shift information for a desired upshift or downshift (for example, as returned in NLINFO items 15 or 16). This parameter will be utilized when bits (9:2) of *flags* is not equal to 0. If present, the *langnum* parameter will be ignored, and this routine will be much more efficient. In split-stack calls this parameter is required if bits (9:2) of *flags* is not equal to 0.

## Special Considerations

Split-stack calls are permitted.

See NLINFO's item 35, the *judgeflag* will return zero's.

# NLSUBSTR

## NLSUBSTR (Intrinsic Number 428)

This intrinsic is used to extract *Length-to-Move* bytes from the *Instring* to the *Outstring*.

### Syntax

```
              BA        IV        BA        I
NLSUBSTR (instring,inlength,outstring,outlength,
            IV                IV         IV
          start'position,length-to-move,langnum,
           IV   LA     LA                     O-V
          flags,error,charset);
```

### Parameters

*instring*

*byte array (required)*
The string from which the substring will be extracted. The string can contain both one-byte and two-byte Asian characters.

*inlength*

*integer by value (required)*
The length, in characters, of *instring*.

*outstring*

*byte array (required)*
Indicates where *substring* will be placed.

*outlength*

*integer (required)*
Length, in characters, of *outstring*. After a successful call, *outlength* will return the actual length of the substring moved to *outstring*.

*start'position*

*integer by value (required)*
The offset into *instring* where the substring starts. A value of zero is the beginning point.

*length-to-move*

*integer by value (required)*
Length, in characters, of the substring.

*langnum*

*integer by value (required)*
The language ID number.

*flags*

*integer by value (required)*
This flag word is used primarily with Asian languages. It is meaningless with one-byte oriented languages. *Flags* is used to indicate the treatment of the case when the first character of the substring is the second byte of a two-byte Asian character and in the case where the last character in a substring is the first byte of a two-byte Asian character.

*Flags.(12:4)* are for the treatment if the first character is the second byte of an Asian character:

0000 : Return an error condition.
0001 : Start from *start'position* +1.
0010 : Start form *start'position* -1.
0011 : Start from *start'position*, but replace the character
       with a blank in *outstring.*
0100 : Start from *start'position* regardless.

*Flags.( 8:4)* are for the treatment if the last character is the first byte of an Asian character:

0000 : Return an error condition.
0001 : Move until *length-to-move* +1.
0010 : Move until *length-to-move* -1.
0011 : Move until *length-to-move*, but replace the character
       with a blank in *outstring.*
0100 : Move until *length-to-move* regardless.

*Flags.( 0:8)* are reserved. These bits must be set to zero.

# NLSUBSTR

*error*

*logical array (required)*
In the first word of this two-word array, the error number will be returned. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS not installed. |
| 2 * | Specified language is not configured. |
| 3 | Not returned. |
| 4 | Not returned. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 | Invalid *source'length.* |
| 8 | Invalid *start'position.* |
| 9 | Invalid *legth-to-move.* |
| 10 | Reserved portion of *Flags,* not zero. |
| 11 | Invalid value for *Flags.( 8:4).* |
| 12 | Invalid value for *Flags.(12:4).* |
| 13 * | The start position is the first byte of an Asian character,<br>or an underflow condition occured due to *Flags.* |
| 14 * | The end position is the second byte of an Asian character,<br>or an overflow condition occured due to *Flags.* |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*charset*

*logical array (optional)*
An array containing the character set definition for the language to be used, as returned by NLINFO's item 12.

## Additional Discussion

Split-stack calls are not permitted.

## NLSWITCHBUF (Intrinsic Number 426)

Converts a string of characters from phonetic order to screen order, or from screen order to phonetic order.

### Syntax

```
               IV       BA       BA        IV
NLSWITCHBUF (langnum,instring,outstring,stringlength,
                LV       LA
             lefttoright,error);
```

### Parameters

*langnum*

*integer by value (required)*
The language ID number.

*instring*

*byte array (required)*
The string, in phonetic order, to be converted to screen order.

*outstring*

*byte array (required)*
Here the string will be returned after being converted. *Outstring* and *instring* may reference the same address.

*stringlength*

*integer by value (required)*
Length, in characters, of the string to be converted.

*lefttoright*

*logical by value (required)*
A logical value that specifies whether the implied primary mode of the data (if it were to be displayed on a terminal) is left to right (TRUE) or right to left (FALSE). This determines what the opposite language is and hence strings of which characters get switched.

*error*

*logical array (required)*
In the first word of this two-word array the error number will be returned. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS not installed. |
| 2 * | Specified language is not installed. |
| 3 | Invalid string length. |
| 4 | Not returned. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

# NLSWITCHBUF

## Additional Discussion

This intrinsic is designed to handle data from languages written from right to left (for example, Arabic). Screen order is defined to be right to left if the primary mode of the terminal or printer is from right to left, as it is when used principally for entering or displaying data from a right to left language. Otherwise, screen order is defined to be left to right.

NLSWITCHBUF can be used by a program to convert a buffer that is in phonetic order (the order in which the characters would be typed at the terminal or spoken by a person) to screen order (the order in which the characters are displayed on a terminal screen or piece of paper). It can also convert data from screen order to phonetic order.

In general, phonetic order and screen order will not be the same if USASCII text is mixed with text from a right to left language. The relationship between phonetic order and screen order is further complicated by the use of Hindi digits in Arabic: Hindi digits play a third role intermediate between ASCII characters and characters of the right to left language.

Note that this intrinsic is designed for a special purpose. Its primary value lies in its application to languages that are written from right to left and which may, occasionally, intermix left to right text–for example, the occasional use of English in Arabic text.

Nonetheless, NLSWITCHBUF can serve the needs of a general purpose program, one not specifically designed for handling right to left data. Such a program can call NLSWITCHBUF to convert data from phonetic order to screen order and back to phonetic order. An example is an editor that needs to track cursor movement on a terminal against a buffer of text in memory. If the data is not that of a right to left language, then this intrinsic will simply return the same text,unchanged, because for all other languages phonetic order and screen order are the same.

## NLTRANSLATE (Intrinsic Number 404)

The NLTRANSLATE intrinsic translates a string of characters from EBCDIC-to-ASCII or ASCII-to-EBCDIC using the appropriate native language table. This intrinsic performs the same function as CTRANSLATE using native language tables.

**NOTE**

This intrinsic does not support 16-bit characters.

### Syntax

```
              IV    BA       BA          IV
NLTRANSLATE (code,instring,outstring,stringlength,
               IV      LA   LA        O-V
             langnum,error,table);
```

The *instring* parameter is translated into *outstring* for length of *stringlength* using a translation table determined according to the first rule that applies from the following list:

1. If *table* is present, a translation will be made using *table*.
2. If *langnum* equals NATIVE-3000, a standard ASCII-to-EBCDIC or EBCDIC-to-ASCII translation is made.
3. The ASCII-to-EBCDIC or EBCDIC-to-ASCII translation table for the language specified will be used.

# NLTRANSLATE

## Parameters

*code*

*integer by value (required)*
The direction of translation:

| | |
|---|---|
| 1 | EBCDIC-to-ASCII |
| 2 | ASCII-to-EBCDIC |

*instring*

*byte array (required)*
The string of characters to be translated.

*outstring*

*byte array (required)*
A byte array to which the translated string is returned. The parameters *instring* and *outstring* may specify the same array.

*stringlength*

*integer by value (required)*
A positive integer specifying the number of bytes of *instring* to be translated.

*langnum*

*integer by value (required)*
The language ID number, specifying which translation tables are to be used.

*error*

*logical array (required)*
The first word of this two-word array contains the error number. The second word is reserved and always contains zero. If the call is successful, both words contain zero.

| Error # | Meaning |
|---|---|
| 1 * | NLS is not installed. |
| 2 * | Specified language is not configured. |
| 3 | Invalid *code* specified. |
| 4 | Invalid *length* parameter. |
| 5 * | NLS internal error. |
| 6 * | NLS internal error. |
| 7 * | Translation table is not supported for this language. |

* These errors do not apply to calls with *langnum* equal to 0 (NATIVE-3000).

*table*

*logical array (optional)*
A 256-byte array which holds a translation table. Each byte contains the translation of the byte whose value is its index. This parameter corresponds to NLINFO items 13 and 14. If present, *langnum* parameter will be ignored and this routine will be much more efficient.

## Special Considerations

Split-stack calls are not permitted.

# System Utilities A

## NLUTIL Program

The NLUTIL program allows the user to verify the language/character set configuration on the system. It displays the configured languages and their character sets, and prompts the user to see if a full listing is required as shown in the dialog below:

```
:RUN NLUTIL.PUB.SYS

Lang     Lang              Char          Char
  ID     Name                ID          Name

   3     DANISH               1          ROMAN8
   5     ENGLISH              1          ROMAN8
  12     SPANISH              1          ROMAN8

Do you require a full listing of the current configuration? (Y/N) x
```

An "N" response will terminate the program. A "Y" response will produce a complete formatted listing of the currently configured languages written to the file NLLIST on device class LP.

## NLS File Structure

The file NLSDEF.PUB.SYS lists all character sets supported by Hewlett-Packard and its related character set names to character set ID numbers. It does the same for languages, and it indicates, for every language, what character set is required to support that language.

The file CHRDEF*xx* (where *xx* is the character set ID number) contains the data pertaining to the character set with ID number *xx*, and all languages supported by that character set. There are numerous CHRDEF*xx* files.

The NLSDEF and the CHRDEF*xx* files are used by the program LANGINST.PUB.SYS to build or modify the file LANGDEF.PUB.SYS. This file is used at system startup to build a number of system data segments holding the information required by NLS. The number of data segments built at startup is one, plus one for every language configured.

## Language Installation Utility (LANGINST)

The file LANGDEF.PUB.SYS contains all language-dependent information for every language to be configured on a system at the next COOLSTART/WARMSTART. It is an MPE file that is built or modified by running the program LANGINST. It gathers data from NLSDEF.PUB.SYS and CHRDEF*xx*.PUB.SYS *files into* LANGDEF.PUB.SYS.

Only a user logged on as MANAGER.SYS,PUB can run LANGINST to:

- Add a language to the configuration file.
- Remove a language from the configuration file.
- Display and modify local formats of a configured language.
- Display the languages supported by Hewlett-Packard.
- Display the languages currently configured.
- Modify the system default language.

**NOTE**

The next system COOLSTART/WARMSTART will implement the changes made to LANGDEF.

### Adding a Language

LANGINST prompts the user MANAGER.SYS for the language to add to LANGDEF. The user may supply either the language ID number or name. If Return is entered, the operation is aborted. If the language is already installed the user is advised, and the addition is cancelled with an error message:

```
SWEDISH is already configured.
```

Similarly, if the appropriate CHRDEF*xx* file is not available, the add is cancelled with an error message:

```
The CHRDEFxx file is missing.
```
The Addition has been cancelled.

Refer to Table A-1 for a complete list of LANGINST error messages. It is not possible to add NATIVE-3000. This language is hard-coded and is always configured. Any attempt to configure it will result in the error message:

```
NATIVE-3000 is always configured.
```

**NOTE**

The next system COOLSTART/WARMSTART will install the language(s) added.

## Deleting a Language

LANGINST allows the user to delete any configured language with the exception of NATIVE-3000, which cannot be deleted. In addition, a check is made to ensure that the language designated as the system default is not deleted.

---

**NOTE**

The next system COOLSTART/WARMSTART will delete the language(s) designated.

---

## Modifying Local Formats

The System Manager is allowed to modify the following local formats for any language configured in LANGDEF:

- Date format (Dateline format)
- Custom date format (Short)
- Time format
- Currency sign/name
- Decimal and thousands indicator
- Month names
- Abbreviated month names
- Weekday names
- Abbreviated weekday names
- Yes/no indicators
- Direction of text
- ASCII/EBCDIC translation tables
- National date table

If the language supports a special National Table containing date information (KATAKANA), the last option is displayed to allow the user to modify this date information.

Whenever any changes have been made, the new copy of the file is saved under the name LANGDEF. In addition, the old, unchanged version of the file is saved under the name LANGD*xxx*. The number *xxx* increases by one every time a new copy of LANGDEF'n saved. This allows the user to return to the configuration that existed before LANGDEF was changed. To return to the previous configuration, :PURGE or :RENAME the current LANGDEF'n Then :RENAME the LANGDxxx with the highest number LANGDEF. The next system COOLSTART/WARMSTART will delete the changes.

## LANGINST User Dialog

The following are user dialogues for choosing a function, adding a language, deleting a language, and modifying local language formats.

### Choosing a Function

The System Manager selects an item from the main menu:

```
0.  EXIT
1.  ADD LANGUAGE TO LANGDEF
2.  DELETE LANGUAGE FROM LANGDEF
3.  MODIFY NATIVE FORMATS
4.  LIST HP SUPPORTED LANGUAGES
5.  MODIFY THE SYSTEM DEFAULT LANGUAGE
6.  LIST LANGUAGES CURRENTLY CONFIGURED
7.  DISPLAY TRANSLATION TABLES
```

To list languages which can be configured on the system, select Option 4. The following will be displayed:

```
HP SUPPORTED LANGUAGES:

 0 NATIVE-3000          using         USASCII
 1 AMERICAN             using         ROMAN8
 2 CANADIAN-FRENCH      using         ROMAN8
 3 DANISH               using         ROMAN8
 4 DUTCH                using         ROMAN8
 5 ENGLISH              using         ROMAN8
 6 FINNISH              using         ROMAN8
 7 FRENCH               using         ROMAN8
 8 GERMAN               using         ROMAN8
 9 ITALIAN              using         ROMAN8
10 NORWEGIAN            using         ROMAN8
11      .                 .              .
12      .                 .              .
13      .                 .              .

press any key to continue ...
```

## Adding a Language

To add a language, select Option 1:

1. Use the language name or language ID number (*langnum*).
2. The addition is aborted by entering a language that is already configured, a language not supported by NLS, or NATIVE-3000 or by pressing Return.

```
Enter language to be added: SPANISH
SPANISH is already configured.
```

If a language is requested that is supported but has not been previously configured, LANGINST configures it and displays the message:

```
SPANISH has been successfully added.
SPANISH will not be configured until you perform a system WARM/COOLSTART
```

3. When the addition is successfully completed, or else aborted, the main menu is displayed.

## Deleting a Language

To delete a language, select Option 2:

1. Use the language name or language ID number (*langnum*).
2. The deletion is aborted by entering a Return, a language that is not configured, or the system default language.
3. When the deletion is successfully completed, or else aborted, the main menu is displayed.

## Modifying Local Language Formats

To modify local language formats, select Option 3:

1. Use the language name or language ID number (*langnum*).
2. The process is aborted by entering a language that is not configured or NATIVE-3000, or by pressing Return.
3. If the process is aborted, the main menu is displayed.
4. If a configured language is entered, a menu is displayed:

```
 1.  Long calendar format
 2.  Date format          (Calendar format)
 3.  Custom date format   (Short)
 4.  Time format          (Clock format)
 5.  Currency sign
 6.  Currency name
 7.  Decimal and thousands separator
 8.  Alternate numeric format
 9.  YES and NO equivalents
10.  Month names.
11.  Month name abbreviations
12.  Weekday names
13.  Weekday name abbreviations
14.  Direction of text
15.  ASCII/EBCDIC translation tables
16.  Handle truncation in date format

Enter selection number        :5
Business Currency sign        :F
Enter the new value           :[Return]
Fully qualified Currency sign :FF
Enter the new value           :[Return]
The currency sign currently follows the number, e.g., 100DM.

The following currency codes are available:

<CR> to retain the existing value.
0 - The currency symbol precedes the number, e.g., $100.00.
1 - The currency symbol succeeds the number, e.g., 100.00DM.
2 - The currency symbol replaces the decimal point, e.g., 100$00.

Enter the required currency codes (0, 1, or 2) :[Return]
There are to be no blanks before or after the currency symbol.

The following blank-control codes are available:

<CR> to retain the existing value.
0 - No blanks before or after the currency symbol.
1 - A blank is to precede the currency symbol.
2 - A blank is to succeed the currency symbol.
3 - A blank is to precede and succeed the currency symbol.

Enter the required code (0, 1, 2, or 3):[Return]
```

After the selection is made, the current value is displayed. The user is prompted for a new value. If a new value is entered, it is validated and, if valid, it replaces the old value. If no new value is entered (only Return) or if an invalid value is entered, the old value is retained.

## Modification of ASCII/EBCDIC Translation Tables

A new option has been added to the utility program LANGINST to modify the ASCII/EBCDIC translation tables for any language other than NATIVE-3000 The modifications will appear in the file LANGDEF and will become effect the next time a COOLSTART/WARMSTART is performed on the system.

For example, assume you need to change the ASCII/EBCDIC translation for two characters in AMERICAN:

| CURRENT | | DESIRED | |
|---|---|---|---|
| ASCII | EBCDIC | ASCII | EBCDIC |
| 04 | 37 | 04 | 44 |
| C8 | 44 | C8 | 37 |

In order to make the changes, the System Manager should run the utility program LANGINST.PUB.SYS and select Option 3 (MODIFY NATIVE FORMAT). After entering the language ID, select Option 15 (ASCII/EBCDIC Translation Tables). Respond to the dialog as follows:

```
Input ROMAN8 character to be changed (HEX please) : 04
The current EBCDIC value is : 37
Enter the new EBCDIC value : 44
The ROMAN8 to EBCDIC table was updated
The EBCDIC to ROMAN8 table will be updated too
ASCII/EBCDIC table inconsistent for 44 <== 04,C8   (*)
The tables are inconsistent for ROMAN8 character C8   (**)
The current EBCDIC value is : 44
Enter the new EBCDIC value : 37
The ROMAN8 to EBCDIC table was updated
The EBCDIC to ROMAN8 table will be updated too
Input ROMAN8 character to be changed (HEX please):[Return]
Do you want to save the changes (Y/N) : Y
```

* There are two ASCII characters mapping to the same EBCDIC character.

** Change the mapping of C8 to its new EBCDIC value.

Both the ROMAN8/EBCDIC and EBCDIC/ROMAN8 translation tables are updated and written out to the LANGDEF file. If you would like to display the translation tables, return to the main menu and enter Option 7. Then enter the *langnum* and the desired table you wish to display.

In the case you have more than two characters to modify, just follow the same steps for every two characters as mentioned above until you finish all pair exchanges.

## Error Messages

Table A-1 contains LANGINST error messages.

Table A-1. LANGINST Error Messages

| MESSAGE | MEANING | ACTION |
|---|---|---|
| A NONNUMERIC GRAPHIC CHARACTER IS EXPECTED... | An alphabetic or special character (not numeric) is expected. | Enter a valid character. |
| ATTEMPTING TO ADD TOO MANY CHARACTER SETS. | Adding this language would exceed the maximum configurable character sets. | Don't configure languages from so many character sets. |
| BUILDING AN EMPTY LANGDEF ... | There was no existing LANGDEF file, so a new, empty one is being built. | None. If you have already configured languages, find LANGDEF.PUB.SYS on a backup and restore it; or else, reconfigure the languages with this program. |
| DELETION TERMINATED ... ATTEMPTING TO DELETE NATIVE-3000. | The language NATIVE-3000 may not be deleted from the list of configured languages. | None. |
| ERRONEOUS STARTING YEAR NUMBER. EXPECTED A NUMBER BETWEEN 0 AND 99. | The year number entered in not valid. | Enter the year number again. It must be a number between 0 and 99. |
| INPUT TOO LONG ... PLEASE REENTER: | The program does not expect so much input in this context. | Reenter the data correctly. |
| INTERNAL ERROR ... PLEASE REPORT. | Internal error. | Contact your Hewlett-Packard representative. |
| INVALID DATE FORMAT. EXPECTED MM/DD/YY. | The entered date is not valid. | Enter the date again in the form MM/DD/YY. |
| *langname*IS ALREADY CONFIGURED. | The language selected has already been configured. | None. |
| *langname*IS AN ILLEGAL LANGUAGE NAME (OR NUMBER). | The language name or number entered is not valid. | Enter the language again, correctly. |
| *langname*IS AN INVALID SYSTEM DEFAULT LANGUAGE. | The language selected is not configured on the system. | Add the language to the list of currently configured languages with this program. |

Table A-1. LANGINST Error Messages (cont.)

| MESSAGE | MEANING | ACTION |
|---|---|---|
| *langname*IS NOT A CONFIGURED LANGUAGE. | The language selected is not configured on your system. | Add the language to the list of currently configured languages with this program. |
| *langname*IS NOT CONFIGURED. | The language entered is not configured on your system. | Add the language to the list of currently configured languages with this program. |
| *langname*IS NOT IN THE CHRDEF FILE. | One of the CHRDEF*xx* files is not consistent with the NLSDEF file. | Restore all CHRDEF*xx* files and NLSDEF from your master backup. |
| NATIVE-3000 IS ALWAYS CONFIGURED. | NATIVE-3000 may not be added to the list of configured languages, because it is always configured. | None. |
| NATIVE-3000 MAY NOT BE MODIFIED. | The language definition of NATIVE-3000 may not be modified. | None. |
| THE CHRDEF*xx*FILE IS MISSING. THE ADDITION HAS BEEN CANCELLED. | The character definition file for the selected language is missing. | Restore the missing file from your master backup. |
| THE DECIMAL SEPARATOR AND THOUSANDS SEPARATOR SHOULD BE DIFFERENT. | The decimals and thousands separators have been defined to be the same. | Change the decimal and/or thousands indicator. |
| THE EXPECTED NAME SHOULD CONTAIN ALPHABETIC CHARACTERS ONLY. | Only alphabetic characters are allowed in this context. | Please re-enter the value, restricting the input to alphabetic characters. |
| THE FILECODE FOR CHRDEF*xx*.PUB.SYS IS INCORRECT. | The character definition file for the selected language has a bad file code. | Restore the missing CHRDEF*xx* file from the master backup. |
| THE FILECODE FOR LANGDEF.PUB.SYS IS INCORRECT. | The current language definition file has a bad file code. | Restore LANGDEF.PUB.SYS from a backup copy. Or purge it, and recreate it by reconfiguring the desired languages with this program. |
| THE FILECODE FOR NLSDEF.PUB.SYS IS INCORRECT. | The master NLS definition file has a bad file code. | Restore NLSDEF.PUB.SYS from the master backup. |

Table A-1. LANGINST Error Messages (cont.)

| MESSAGE | MEANING | ACTION |
|---|---|---|
| THE LANGUAGE YOU ARE ATTEMPTING TO DELETE IS THE SYSTEM DEFAULT LANGUAGE. | The system default language may not be deleted from the list of configured languages. | If you wish to delete this language, you must first change the system default language to another language. |
| THE USER SHOULD BE MANAGER.SYS, RUNNING IN THE PUB GROUP. | The user is not MANAGER.SYS or is not logged on in the PUB group. | Log on as MANAGER.SYS in the PUB group and run the program again. |
| THERE IS NO MORE ROOM FOR ADDITIONAL DATE PERIODS. PLEASE REPORT. | There is no room for additional entries in the national date table. | Contact your Hewlett-Packard representative. |
| TOO MANY LANGUAGES HAVE BEEN CONFIGURED. | Adding another language would exceed the maximum configurable languages. | Don't configure so many languages on one system. |
| UNABLE TO RENAME LANGDEF TO LANGD*nnn*. THE EXISTING LANGDEF WILL BE PURGED. | The old LANGDEF file could not be renamed because all files LANGD000 through LANGD999 already existed. | Purge some or all of the files LANGD000 to LANGD999 so the most recent changes to LANGDEF can be saved in the future. |
| UNKNOWN OPTION ....<br>PLEASE REENTER. | The option selected is not a valid one. | Enter the number corresponding to one of the currently valid options. |

# SUPPORTED LANGUAGES AND CHARACTER SETS B

## Character Set Definitions

Every language supported in NLS is uniquely identified by number and name. Every language has:

- A character set number.
- A language identification number.
- A language name.

The pages that follow in this appendix are devoted to unique character sets. Every set consists of NATIVE-3000, language identification number (*langnum*) 00, and may include one or more languages affiliated with the character set.

All character sets are supersets of USASCII and are occasionally referred to generically as "ASCII" character sets, as in the term "ASCII-to-EBCDIC translation".

For every character set, a character attribute table is defined. This table of 256 entries holds an attribute type for every character. The type identification is:

0: Numeric character
1: Alphabetic lowercase character
2: Alphabetic uppercase character
3: Undefined graphic character
4: Special character
5: Control code (for example, linefeed, escape)
6: First byte of a two-byte Asian character

The following items are defined for every supported language:

- The upshift and downshift table
- The collating sequence table
- The ASCII-to-EBCDIC and EBCDIC-to-ASCII translate tables
- The long date format (the DATELINE format)
- The short date format (the custom date format)
- The time format
- The currency symbol (one character)
- The currency name (up to sixteen characters)
- The currency descriptor (up to four characters)
- The position and spacing of the currency sign
- The decimal and thousands separators for numbers
- The equivalents of YES and NO (both up to six characters)
- The full weekday names (up to twelve characters)
- The abbreviated weekday names (up to three characters)
- The full month names (up to twelve characters)
- The abbreviated month names (up to four characters)
- Text direction (left to right or right to left)
- Alternate set of digits (where applicable)
- The National Date table (where applicable)

Refer to the discussion on the NLINFO intrinsic, in Chapter 4, for a complete description of these items.

## Language Definitions and Character Sets

The following pages contain the character sets and definitions supported by NLS.

### NATIVE-3000

| USASCII<br>**Language Number** | **(Set #0)**<br>**Language Name** |
|---|---|
| 00 | NATIVE-3000 |

The USASCII character set is a subset of the ROMAN8 character set shown in Figure B-1. It is contained in columns 0 through 7.

### ROMAN8

| **Language Number** | **(Set #1)**<br>**Language Name** |
|---|---|
| 00 | NATIVE-3000 |
| 01 | AMERICAN |
| 02 | CANADIAN-FRENCH |
| 03 | DANISH |
| 04 | DUTCH |
| 05 | ENGLISH |
| 06 | FINNISH |
| 07 | FRENCH |
| 08 | GERMAN |
| 09 | ITALIAN |
| 10 | NORWEGIAN |
| 11 | PORTUGUESE |
| 12 | SPANISH |
| 13 | SWEDISH |
| 14 | ICELANDIC |
| 15 - 40 | Reserved |

|  |  |  |  | b8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  | b7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|  |  |  |  | b6 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|  |  |  |  | b5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| b4 | b3 | b2 | b1 |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ‘ | p | ▨ | ▨ | ▨ | ¯ | â | Å | Á | Þ |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q | ▨ | ▨ | À |  | ê | î | Ã | þ |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r | ▨ | ▨ | Â |  | ô | Ø | ã |  |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s | ▨ | ▨ | È | ° | û | Æ | Ð |  |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t | ▨ | ▨ | Ê | Ç | á | å | đ |  |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u | ▨ | ▨ | Ë | ç | é | í | Í |  |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v | ▨ | ▨ | Î | Ñ | ó | ø | Ì | — |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ’ | 7 | G | W | g | w | ▨ | ▨ | Ï | ñ | ú | æ | Ó | ¼ |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x | ▨ | ▨ | ´ | ¡ | à | Ä | Ò | ½ |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y | ▨ | ▨ | ` | ¿ | è | ì | Õ | ª |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z | ▨ | ▨ | ^ | ¤ | ò | Ö | õ | º |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { | ▨ | ▨ | ¨ | £ | ù | Ü | Š | « |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | \ | l | \| | ▨ | ▨ | ˜ | ¥ | ä | É | š | ■ |
| 1 | 1 | 0 | 1 | 13 | CR | GS | - | = | M | ] | m | } | ▨ | ▨ | Ù | § | ë | ï | Ú | » |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ^ | n | ~ | ▨ | ▨ | Û | ƒ | ö | ß | Ÿ | ± |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL | ▨ | ▨ | ₤ | ¢ | ü | Ô | ÿ | ▨ |

Figure B-1. ROMAN8 Character Set

# KANA8

| Language Number | (Set #2)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 41 | KATAKANA (Phonetic Japanese) |

| | | | | $b_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $b_7$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | | $b_6$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | | $b_5$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $b_4$ | $b_3$ | $b_2$ | $b_1$ | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ‘ | p | ▨ | ▨ | ▨ | ー | タ | ミ | | |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q | ▨ | ▨ | 。 | ア | チ | ム | | |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r | ▨ | ▨ | 「 | イ | ツ | メ | | |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s | ▨ | ▨ | 」 | ウ | テ | モ | | |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t | ▨ | ▨ | 、 | エ | ト | ヤ | | |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u | ▨ | ▨ | ・ | オ | ナ | ユ | | |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v | ▨ | ▨ | ヲ | カ | ニ | ヨ | | |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w | ▨ | ▨ | ァ | キ | ヌ | ラ | | |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x | ▨ | ▨ | ィ | ク | ネ | リ | | |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y | ▨ | ▨ | ゥ | ケ | ノ | ル | | |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z | ▨ | ▨ | ェ | コ | ハ | レ | | |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { | ▨ | ▨ | ォ | サ | ヒ | ロ | | |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | ¥ | l | \| | ▨ | ▨ | ャ | シ | フ | ワ | | |
| 1 | 1 | 0 | 1 | 13 | CR | GS | - | = | M | ] | m | } | ▨ | ▨ | ュ | ス | ヘ | ン | | |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ^ | n | ~ | ▨ | ▨ | ョ | セ | ホ | ゛ | | |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL | ▨ | ▨ | ッ | ソ | マ | ゜ | | ▨ |

Figure B-2. KANA8 Character Set

# ARABIC8

| Language Number | (Set #3)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 49 | ARABICL |
| 50 | ARABICR |
| 51 | ARABIC |
| 52 | ARABICW |
| 53 | ARABICWL |
| 54 | ARABICWR |

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | b8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | b7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | | b6 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | | b5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| b4 | b3 | b2 | b1 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p | ▨ | ▨ | ▨ | ٠ | @ | ذ | — | ِ |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q | ▨ | ▨ | ! | ١ | ء | ر | ف | ّ |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r | ▨ | ▨ | " | ٢ | آ | ز | ق | ْ |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s | ▨ | ▨ | | ٣ | أ | س | ك | |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t | ▨ | ▨ | | ٤ | ؤ | ش | ل | |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u | ▨ | ▨ | ٪ | ٥ | إ | ص | م | |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v | ▨ | ▨ | | ٦ | ئـ | ض | ن | َّ |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w | ▨ | ▨ | | ٧ | ا | ط | ه | ُّ |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x | ▨ | ▨ | ) | ٨ | ب | ظ | و | ِّ |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y | ▨ | ▨ | ( | ٩ | ة | ع | ى | |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z | ▨ | ▨ | | : | ت | غ | ي | |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { | ▨ | ▨ | + | ؛ | ث | | ، | |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | \ | l | \| | ▨ | ▨ | ٫ | | ج | | | |
| 1 | 1 | 0 | 1 | 13 | CR | GS | — | = | M | ] | m | } | ▨ | ▨ | | = | ح | | | |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ∧ | n | ~ | ▨ | ▨ | | | خ | | | |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL | ▨ | ▨ | / | ؟ | د | — | | ▨ |

Figure B-3. ARABIC8 Character Set

## GREEK8

| Language Number | (Set #4)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 61 | GREEK |

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | b8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | b7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | | b6 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | | b5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| b4 | b3 | b2 | b1 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p | ▨ | ▨ | ▨ | | | Ο | ύ | ο |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q | ▨ | ▨ | | | Α | Π | α | π̇ |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r | ▨ | ▨ | | | Β | Ρ | β | ρ |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s | ▨ | ▨ | | | Γ | Σ | γ | σ |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t | ▨ | ▨ | | | Δ | Τ | δ | τ |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u | ▨ | ▨ | | | Ε | Υ | ε | υ |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v | ▨ | ▨ | | | Ζ | Φ | ζ | φ |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w | ▨ | ▨ | | | Η | | η | ς |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x | ▨ | ▨ | | | Θ | Χ | θ | χ |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y | ▨ | ▨ | | | Ι | Ψ | ι | ψ |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z | ▨ | ▨ | | | | Ω | | ώ |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { | ▨ | ▨ | | | Κ | ά | κ | ΐ |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | \ | l | \| | ▨ | ▨ | | L̈ | Λ | ή | λ | Ĺ |
| 1 | 1 | 0 | 1 | 13 | CR | GS | − | = | M | ] | m | } | ▨ | ▨ | | | Μ | ό | μ | ώ |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ^ | n | ~ | ▨ | ▨ | | Ü | Ν | | ν | ’ |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL | ▨ | ▨ | | | Ξ | | ξ | ▨ |

Figure B-4. GREEK8 Character Set

## TURKISH8

| Language Number | (Set #6)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 81 | TURKISH |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  | b8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  |  |  |  | b7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|  |  |  |  |  | b6 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|  |  |  |  |  | b5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| b4 | b3 | b2 | b1 |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 0 | 0 | 0 |  | NUL | DLE | SP | 0 | @ | P | ` | p | ▨ | ▨ | ▨ |  |  | Å | ğ | Þ |
| 0 | 0 | 0 | 1 | 1 |  | SOH | DC1 | ! | 1 | A | Q | a | q | ▨ | ▨ | Ç | Ý | ê | î | Ã | þ |
| 0 | 0 | 1 | 0 | 2 |  | STX | DC2 | " | 2 | B | R | b | r | ▨ | ▨ | Ğ | ý | ô | Ø | ã | · |
| 0 | 0 | 1 | 1 | 3 |  | ETX | DC3 | # | 3 | C | S | c | s | ▨ | ▨ | È | · |  | Æ | Ð | µ |
| 0 | 1 | 0 | 0 | 4 |  | EOT | DC4 | $ | 4 | D | T | d | t | ▨ | ▨ | Ê |  | á | å | ŏ | ¶ |
| 0 | 1 | 0 | 1 | 5 |  | ENQ | NAK | % | 5 | E | U | e | u | ▨ | ▨ | Ë |  | é | í | Í | ¾ |
| 0 | 1 | 1 | 0 | 6 |  | ACK | SYN | & | 6 | F | V | f | v | ▨ | ▨ | Î | Ñ | ó | ø | Ì | – |
| 0 | 1 | 1 | 1 | 7 |  | BEL | ETB | ' | 7 | G | W | g | w | ▨ | ▨ | Ï | ñ | ú | æ | Ó | ¼ |
| 1 | 0 | 0 | 0 | 8 |  | BS | CAN | ( | 8 | H | X | h | x | ▨ | ▨ | ´ | ¡ | à | Ä | Ò | ½ |
| 1 | 0 | 0 | 1 | 9 |  | HT | EM | ) | 9 | I | Y | i | y | ▨ | ▨ | ` | ¿ | è | ì | Õ | ª |
| 1 | 0 | 1 | 0 | 10 |  | LF | SUB | * | : | J | Z | j | z | ▨ | ▨ | ^ | TL | ò |  | õ | º |
| 1 | 0 | 1 | 1 | 11 |  | VT | ESC | + | ; | K | [ | k | { | ▨ | ▨ | ¨ | £ | ù | İ | Š | ı |
| 1 | 1 | 0 | 0 | 12 |  | FF | FS | , | < | L | \ | l | \| | ▨ | ▨ | ~ | ¥ | ä | Ö | š | ö |
| 1 | 1 | 0 | 1 | 13 |  | CR | GS | - | = | M | ] | m | } | ▨ | ▨ | Ù | § | ë | Ş | Ú | ş |
| 1 | 1 | 1 | 0 | 14 |  | SO | RS | . | > | N | ^ | n | ~ | ▨ | ▨ | Û | ƒ |  | Ü | Ÿ | ü |
| 1 | 1 | 1 | 1 | 15 |  | SI | US | / | ? | O | _ | o | DEL | ▨ | ▨ | ₤ | c |  | ç | ÿ | ▨ |

Figure B-5. TURKISH8 Character Set

## PRC15

| Language Number | (Set #51)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 201 | SIMPLIFIED CHINESE (CHINESE-S) |

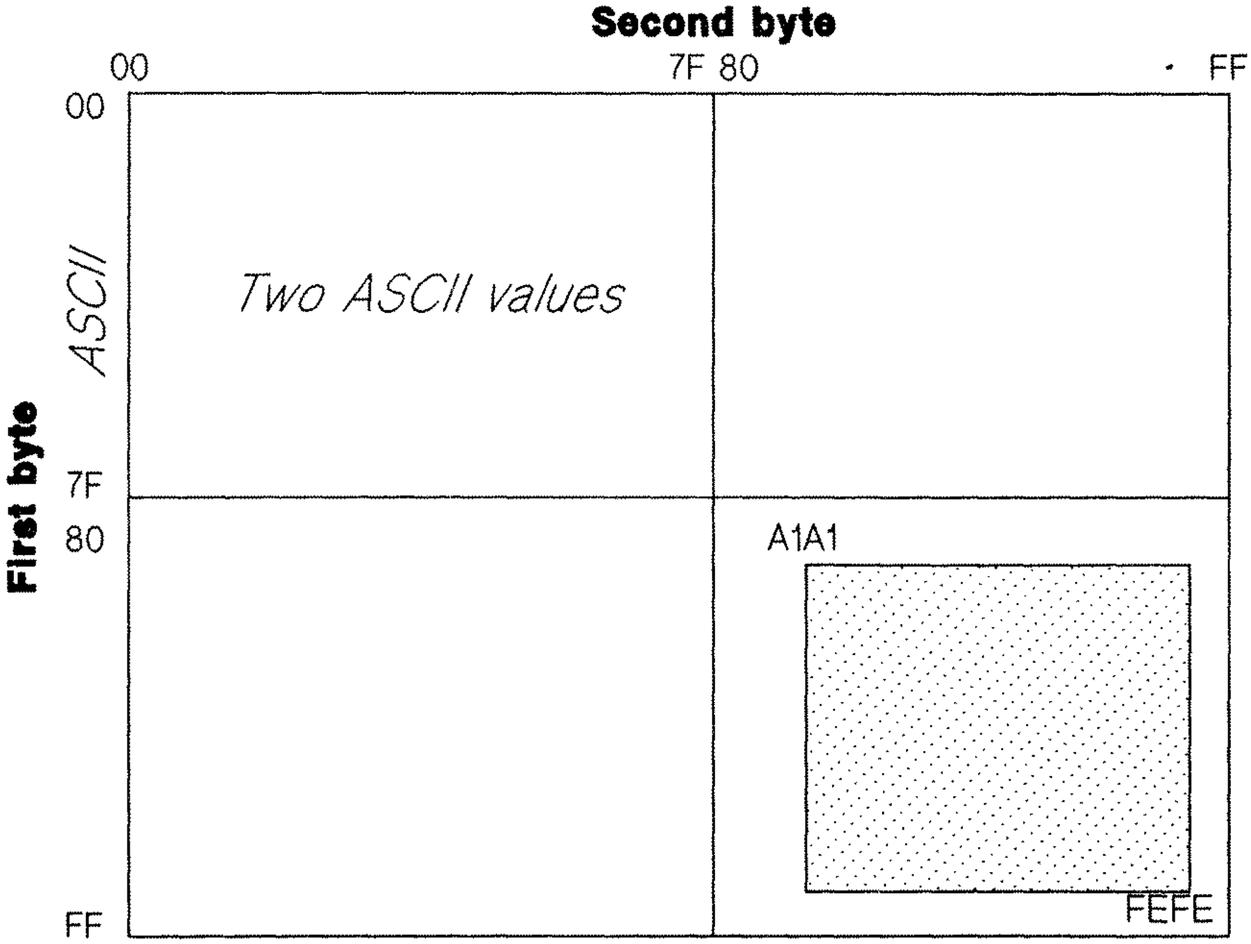


Figure B-6. PRC15 Character Set

## ROC15

(Set #56)

| Language Number | Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 211 | TRADITIONAL CHINESE (CHINESE-T) |

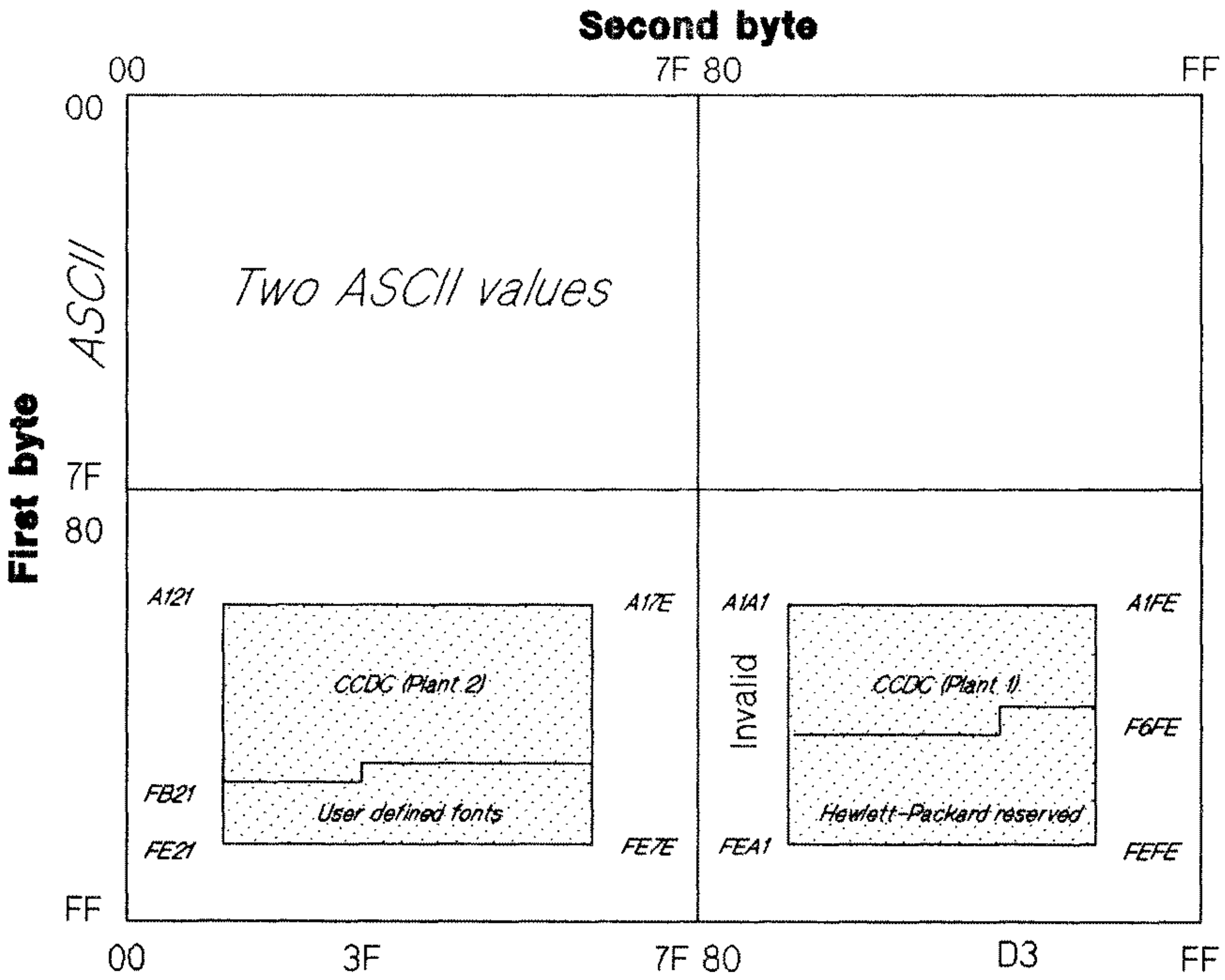


Figure B-7. ROC15 Character Set

## JAPAN15

| Language Number | (Set #61)<br>Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 221 | JAPANESE |

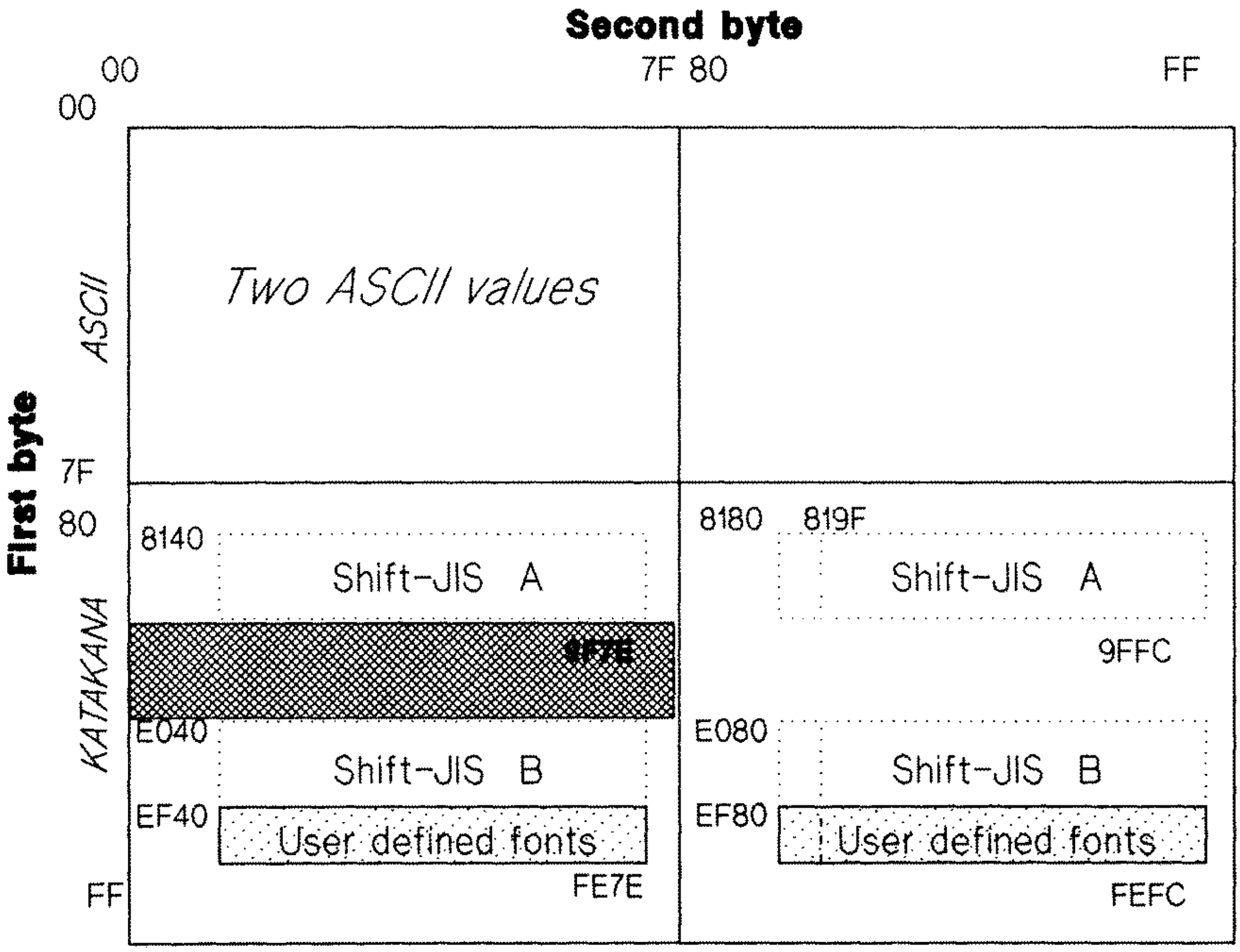


**Figure B-8. JAPAN15 Character Set**

## KOREA15

(Set #66)

| Language Number | Language Name |
|---|---|
| 00 | NATIVE-3000 |
| 231 | KOREAN |

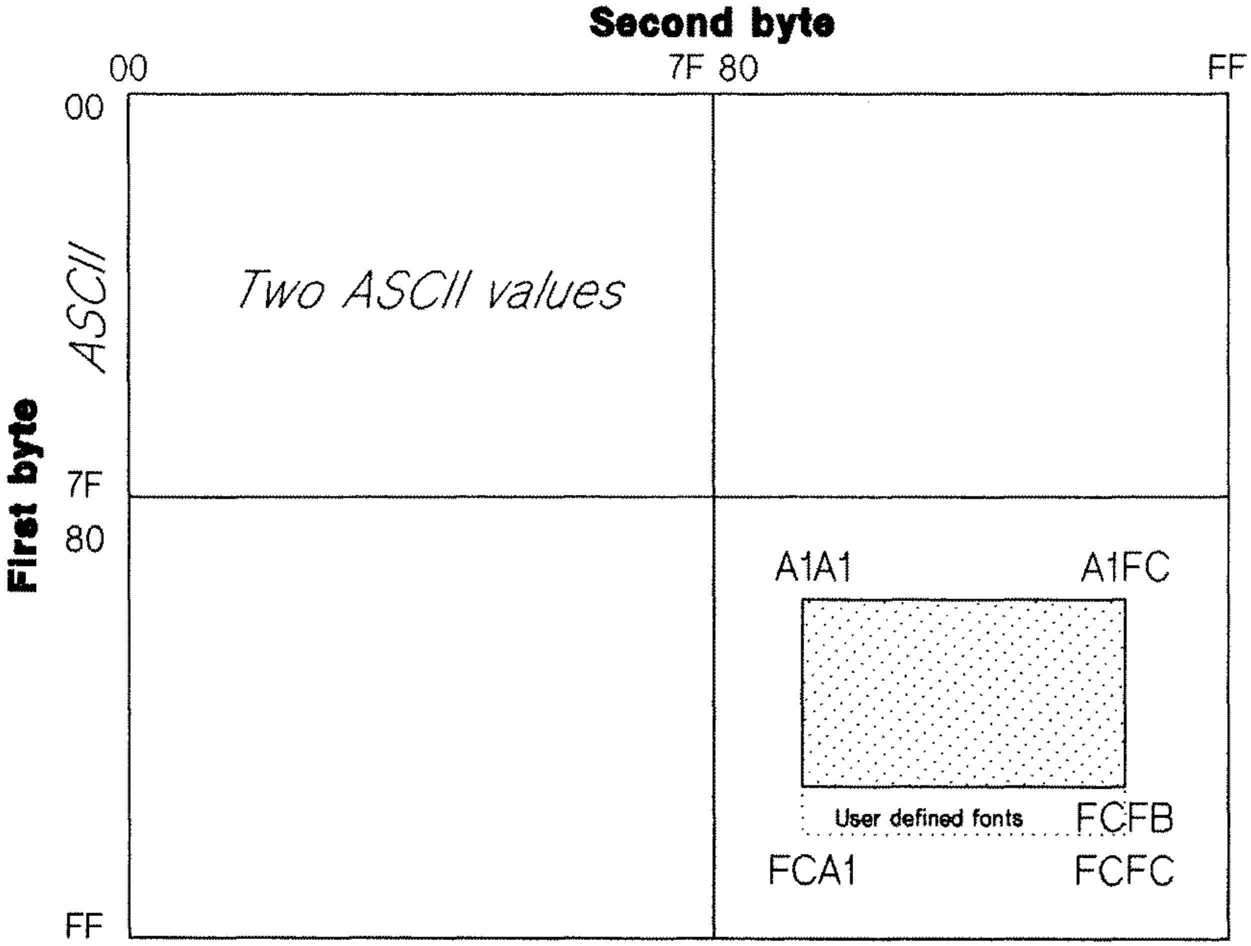


**Figure B-9. KOREA15 Character Set**

# COLLATING IN EUROPEAN LANGUAGES C

Collating is defined as arranging character strings into some (usually alphabetic) order. To do this a mechanism must be available that, given two character strings, decides which one comes first. In Native Language Support (NLS) this mechanism is the NLCOLLATE intrinsic.

Look at the full ROMAN8 character set and consider that all these characters can appear in every European language. Even if a character does not exist in a language, it can still show up in names and/or addresses. It is quite useful to address a letter to Spain correctly, even if it originates in Germany. Therefore, the full ROMAN8 character set is considered to be used in all languages, and a collating sequence has been defined for all characters in the ROMAN8 character set for the languages it supports. Figure C-1 lists the collating sequence for:

AMERICAN
CANADIAN-FRENCH
DANISH
DUTCH
ENGLISH
FINNISH
FRENCH
GERMAN
ITALIAN
NORWEGIAN
PORTUGUESE
SPANISH
SWEDISH

All characters in a group, indicated by brackets (or, in a few footnotes, by underlining) collate the same. These characters usually differ only in uppercase versus lowercase priority, or accent priority. In sorting, they are initially considered the same. If the remaining characters in the two strings do not determine which string comes first, then the priorities of characters will be used to determine the order. Refer to Table C-1 for examples of collating sequence priority.

Table C-1. Examples of Collating Sequence Priority

| Sorted Strings | Explanation |
|---|---|
| aéb, aéc | The third character in each string is different. The "b" precedes the "c". |
| aeb, aéb | The characters in the two strings are identical, so accent priority determines the order. The "e" precedes the "é". |
| abc, Abd | The last characters in the strings are different. The "c" precedes the "d". |
| aBc, abc | The characters in the two strings are the same, so the uppercase priority determines the order. "B" precedes "b". |

## NOTE

This Appendix deals with collating or lexical ordering and does not include matching. For matching purposes, there is generally a difference between "A" and "a".

Figures C-1 and C-2 display the collating sequence in three ways: the graphic representation of the character, the decimal equivalent of the character's binary value, and a description of the character. Language-dependent variations to the collating sequence appear in Figure C-2.

## Collating Sequence

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
|  | 32 | Space |
|  | 160 | Do Not Use |
| 0 | 48 | Zero |
| 1 | 49 | One |
| 2 | 50 | Two |
| 3 | 51 | Three |
| 4 | 52 | Four |
| 5 | 53 | Five |
| 6 | 54 | Six |
| 7 | 55 | Seven |
| 8 | 56 | Eight |
| 9 | 57 | Nine |
| A | 65 | Uppercase A |
| a | 97 | Lowercase a |
| Á | 224 | Uppercase A Acute |
| á | 196 | Lowercase a Acute |
| À | 161 | Uppercase A Grave |
| à | 200 | Lowercase a Grave |
| Â | 162 | Uppercase A Circumflex |
| â | 192 | Lowercase a Circumflex |
| Ä | 216 | Uppercase A Umlaut/Diaeresis |
| ä | 204 | Lowercase a Umlaut/Diaeresis |
| Å | 208 | Uppercase A Degree |
| å | 212 | Lowercase a Degree |
| Ã | 225 | Uppercase A Tilde |
| ã | 226 | Lowercase a Tilde |
| B | 66 | Uppercase B |
| b | 98 | Lowercase b |

Note that Æ ligature (211) and æ (215) are expanded for collating purposes to AE or ae and collate as: ad AE Ae Æ aE ae æ AF.

**Figure C-1. Collating Sequence (1 of 7)**

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| C | 67 | Uppercase C |
| c | 99 | Lowercase c |
| Ç | 180 | Uppercase C Cedilla |
| ç | 181 | Lowercase c Cedilla |
| D | 68 | Uppercase D |
| d | 100 | Lowercase d |
| Đ | 227 | Uppercase D Stroke |
| đ | 228 | Lowercase d Stroke |
| E | 69 | Uppercase E |
| e | 101 | Lowercase e |
| É | 220 | Uppercase E Acute |
| é | 197 | Lowercase e Acute |
| È | 163 | Uppercase E Grave |
| è | 201 | Lowercase e Grave |
| Ê | 164 | Uppercase E Circumflex |
| ê | 193 | Lowercase e Circumflex |
| Ë | 165 | Uppercase E Umlaut/Diaeresis |
| ë | 205 | Lowercase e Umlaut/Diaeresis |
| F | 70 | Uppercase F |
| f | 102 | Lowercase f |
| G | 71 | Uppercase G |
| g | 103 | Lowercase g |
| H | 72 | Uppercase H |
| h | 104 | Lowercase h |
| I | 73 | Uppercase I |
| i | 105 | Lowercase i |
| Í | 229 | Uppercase I Acute |
| í | 213 | Lowercase i Acute |
| Ì | 230 | Uppercase I Grave |
| ì | 217 | Lowercase i Grave |
| Î | 166 | Uppercase I Circumflex |
| î | 209 | Lowercase i Circumflex |
| Ï | 167 | Uppercase I Umlaut/Diaeresis |
| ï | 221 | Lowercase i Umlaut/Diaeresis |
| J | 74 | Uppercase J |
| j | 106 | Lowercase j |
| K | 75 | Uppercase K |
| k | 107 | Lowercase k |

Figure C-1. Collating Sequence (2 of 7)

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| L | 76 | Uppercase L |
| l | 108 | Lowercase l |
| M | 77 | Uppercase M |
| m | 109 | Lowercase m |
| N | 78 | Uppercase N |
| n | 110 | Lowercase n |
| Ñ | 182 | Uppercase N Tilde |
| ñ | 183 | Lowercase n Tilde |
| O | 79 | Uppercase O |
| o | 111 | Lowercase o |
| Ó | 231 | Uppercase O Acute |
| ó | 198 | Lowercase o Acute |
| Ò | 232 | Uppercase O Grave |
| ò | 202 | Lowercase o Grave |
| Ô | 223 | Uppercase O Circumflex |
| ô | 194 | Lowercase o Circumflex |
| Ö | 218 | Uppercase O Umlaut/Diaeresis |
| ö | 206 | Lowercase o Umlaut/Diaeresis |
| Õ | 233 | Uppercase O Tilde |
| õ | 234 | Lowercase o Tilde |
| Ø | 210 | Uppercase O Crossbar |
| ø | 214 | Lowercase o Crossbar |
| P | 80 | Uppercase P |
| p | 112 | Lowercase p |
| Q | 81 | Uppercase Q |
| q | 113 | Lowercase q |
| R | 82 | Uppercase R |
| r | 114 | Lowercase r |
| S | 83 | Uppercase S |
| s | 115 | Lowercase s |
| Š | 235 | Uppercase S Caron |
| š | 236 | Lowercase s Caron |
| T | 84 | Uppercase T |
| t | 116 | Lowercase t |

Note that the ß (222, sharp s) is expanded to ss and collates according to the German standard as: sr ß ss st.

Figure C-1. Collating Sequence (3 of 7)

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| U | 85 | Uppercase U |
| u | 117 | Lowercase u |
| Ú | 237 | Uppercase U Acute |
| ú | 199 | Lowercase u Acute |
| Ù | 173 | Uppercase U Grave |
| ù | 203 | Lowercase u Grave |
| Û | 174 | Uppercase U Circumflex |
| û | 195 | Lowercase u Circumflex |
| Ü | 219 | Uppercase U Umlaut/Diaeresis |
| ü | 207 | Lowercase u Umlaut/Diaeresis |
| V | 86 | Uppercase V |
| v | 118 | Lowercase v |
| W | 87 | Uppercase W |
| w | 119 | Lowercase w |
| X | 88 | Uppercase X |
| x | 120 | Lowercase x |
| Y | 89 | Uppercase Y |
| y | 121 | Lowercase y |
| Ÿ | 238 | Uppercase Y Umlaut/Diaeresis |
| ÿ | 239 | Lowercase y Umlaut/Diaeresis |
| Z | 90 | Uppercase Z |
| z | 122 | Lowercase z |
| Þ | 240 | Uppercase Thorn |
| þ | 241 | Lowercase Thorn |
|  | 177 | Currently Undefined |
|  | 178 | Currently Undefined |
|  | 242 | Currently Undefined |
|  | 243 | Currently Undefined |
|  | 244 | Currently Undefined |
|  | 245 | Currently Undefined |

Figure C-1. Collating Sequence (4 of 7)

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| ( | 40 | Left Parenthesis |
| ) | 41 | Right Parenthesis |
| [ | 91 | Left Bracket |
| ] | 93 | Right Bracket |
| { | 123 | Left Brace |
| } | 125 | Right Brace |
| « | 251 | Left Guillemets |
| » | 253 | Right Guillemets |
| < | 60 | Less Than Sign |
| > | 62 | Greater Than Sign |
| = | 61 | Equal Sign |
| + | 43 | Plus |
| - | 45 | Minus |
| ± | 254 | Plus/Minus |
| ¼ | 247 | One Quarter |
| ½ | 248 | One Half |
| ° | 179 | Degree (Ring) |
| % | 37 | Percent Sign |
| * | 42 | Asterisk |
| . | 46 | Period (Point) |
| , | 44 | Comma |
| ; | 59 | Semicolon |
| : | 58 | Colon |

Figure C-1. Collating Sequence (5 of 7)

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| ¿ | 185 | Inverse Question Mark |
| ? | 63 | Question Mark |
| ¡ | 184 | Inverse Exclamation Point |
| ! | 33 | Exclamation Point |
| / | 47 | Slant |
| \ | 92 | Reverse Slant |
| \| | 124 | Vertical Bar |
| @ | 64 | Commercial At |
| & | 38 | Ampersand |
| # | 35 | Number Sign (Hash) |
| § | 189 | Section |
| $ | 36 | U. S. Dollar Sign |
| ¢ | 191 | U. S. Cent Sign |
| £ | 187 | British Pound Sign |
| ₤ | 175 | Italian Lira Sign |
| ¥ | 188 | Japanese Yen Sign |
| ƒ | 190 | Dutch Guilder Sign |
| ¤ | 186 | General Currency Sign |
| " | 34 | Double Quote |
| ‘ | 96 | Opening Single Quote |
| ’ | 39 | Closing Single Quote |
| ^ | 94 | Caret |
| ~ | 126 | Tilde |

Figure C-1. Collating Sequence (6 of 7)

| CHARACTER | DECIMAL<br>EQUIVALENT | DESCRIPTION |
|---|---|---|
| ´ | 168 | Accent Acute |
| ` | 169 | Accent Grave |
| ^ | 170 | Accent Circumflex |
| ¨ | 171 | Umlaut/Diaeresis |
| ~ | 172 | Tilde Accent |
| _ | 95 | Underscore |
| — | 246 | Long Dash |
| ‾ | 176 | Overline |
| ª | 249 | Feminine Ordinal Indicator |
| º | 250 | Masculine Ordinal Indicator |
| ■ | 252 | Solid |
|  | 0<br>.<br>.<br>.<br>31 | Control Codes |
|  | 128<br>.<br>.<br>.<br>159 | Currently Undefined<br>Control Codes |
|  | 127 | DEL |
|  | 255 | Do Not Use |

Figure C-1. Collating Sequence (7 of 7)

## Language-Dependent Variations

Listed below are language-dependent variations for Spanish, Danish/Norwegian, Swedish and Finnish.

**SPANISH.** CH is considered a separate character, which collates between C and D. The same applies to LL, which collates after L and before M:

| | |
|---|---|
| C@ | l@ |
| CH | LL |
| Ch | Ll |
| cH | lL |
| ch | ll |
| D@ | M@ |

The @ symbol can equal anything. Therefore, CH comes after C followed by anything, and before D followed by anything.

In Spanish N and Ñ are not considered the same in collating (this also applies to n and ñ). They are different characters which follow one another in the collating sequence:

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| N | 78 | Uppercase N |
| n | 110 | Lowercase n |
| Ñ | 182 | Uppercase N Tilde |
| ñ | 183 | Lowercase n Tilde |

**DANISH/NORWEGIAN.** The Æ, Ø, and Å collate at the end of the alphabet:

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| Z | 90 | Uppercase Z |
| z | 122 | Lowercase z |
| Æ | 211 | Uppercase AE Ligature |
| æ | 215 | Lowercase ae Ligature |
| Ø | 210 | Uppercase O Crossbar |
| ø | 214 | Lowercase o Crossbar |
| Å | 208 | Uppercase A Degree |
| å | 212 | Lowercase a Degree |
| Þ | 240 | Uppercase Thorn |
| þ | 241 | Lowercase Thorn |

Figure C-2. Language-Dependent Variations (1 of 3)

**SWEDISH**. The Å, Ä and Ö are collated at the end of alphabet:

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| Z | 90 | Uppercase Z |
| z | 122 | Lowercase z |
| Å | 208 | Uppercase A Degree |
| å | 212 | Lowercase a Degree |
| Ä | 216 | Uppercase A Umlaut/Diaeresis |
| ä | 204 | Lowercase a Umlaut/Diaeresis |
| Ö | 218 | Uppercase O Umlaut/Diaeresis |
| ö | 206 | Lowercase o Umlaut/Diaeresis |
| Þ | 240 | Uppercase Thorn |
| þ | 241 | Lowercase Thorn |

**FINNISH**. The Å, Ä, and Ö are treated the same as in Swedish. The Ø is considered to be the same as Ö. V and W, and Y and Ü are regarded as the same in Finnish.

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| U | 85 | Uppercase U |
| u | 117 | Lowercase u |
| Ú | 237 | Uppercase U Acute |
| ú | 199 | Lowercase u Acute |
| Ù | 173 | Uppercase U Grave |
| ù | 203 | Lowercase u Grave |
| Û | 174 | Uppercase U Circumflex |
| û | 195 | Lowercase u Circumflex |
| V | 86 | Uppercase V |
| v | 118 | Lowercase v |
| W | 87 | Uppercase W |
| w | 119 | Lowercase w |
| X | 88 | Uppercase X |
| x | 120 | Lowercase x |
| Y | 89 | Uppercase Y |
| y | 121 | Lowercase y |
| Ÿ | 238 | Uppercase Y Umlaut/Diaeresis |
| ÿ | 239 | Lowercase y Umlaut/Diaeresis |
| Ü | 219 | Uppercase U Umlaut/Diaeresis |
| ü | 207 | Lowercase u Umlaut/Diaeresis |

**Figure C-2. Language-Dependent Variations (2 of 3)**

| CHARACTER | DECIMAL EQUIVALENT | DESCRIPTION |
|---|---|---|
| Z | 90 | Uppercase Z |
| z | 122 | Lowercase z |
| Å | 208 | Uppercase A Degree |
| å | 212 | Lowercase a Degree |
| Ä | 216 | Uppercase A Umlaut/Diaeresis |
| ä | 204 | Lowercase a Umlaut/Diaeresis |
| Ö | 218 | Uppercase O Umlaut/Diaeresis |
| ö | 206 | Lowercase o Umlaut/Diaeresis |
| Ø | 210 | Uppercase O Crossbar |
| ø | 214 | Lowercase o Crossbar |
| Þ | 240 | Uppercase Thorn |
| þ | 241 | Lowercase Thorn |

Figure C-2. Language-Dependent Variations (3 of 3)

# EBCDIC MAPPINGS

# D

NLS provides mappings, through NLTRANSLATE and NLINFO, from HP 3000 supported character sets (ROMAN8, KANA8) to the various national versions of the EBCDIC code. This applies to all native languages supported on the HP 3000 and is done differently for each language.

## Background Data

EBCDIC is an 8-bit code which originally used only 128 of the 256 possible code values. These 128 characters have almost the same graphic representations as the traditional 7-bit, 128-character, USASCII code. Three characters are different. USASCII has the left and right square brackets ([ and ]) and the caret (^), while EBCDIC includes the American cent (¢), the logical OR (|), and the logical NOT (¬).

The EBCDIC code was modified to accommodate the extra characters required by European languages. For example, when the German EBCDIC was defined some less important characters were traded for German national characters, and the vertical bar (|) became lowercase ö. Similar things happened to create EBCDIC codes for Norwegian/Danish, Swedish/Finnish, Spanish, Belgian, Italian, Portuguese, French, and English in the UK.

The 128 unused positions in the various national language EBCDIC codes were later used to accommodate all national characters which appeared in any of the EBCDIC codes. Each resulting Country Extended Code Page became a superset of each existing national EBCDIC. In the German table, for instance, the empty space was used to accommodate characters from other languages, but the traditional German characters ä, ö, ü, and ß retained their original position in the German national EBCDIC. There are many Country Extended Code Pages now, all showing exactly the same characters, but showing them in different locations. Consider, for example, the character which has decimal code 161 (octal 241, hexadecimal A1). In original EBCDIC, this is the tilde (~) in Spanish, the sharp s (ß) in German, the diaeresis accent " in French, the lowercase ü in Swedish/Finnish and Norwegian/Danish, the lowercase ì in Italian, and the lowercase ç in Portuguese.

This situation makes it necessary to map the Hewlett-Packard ROMAN8 character set to the many different EBCDIC Country Extended Code Pages.

## ROMAN8 to EBCDIC Mapping

In mapping from ROMAN8 to and from any EBCDIC, characters look the same, or as close as possible, before and after conversion. The majority of the symbols appearing in ROMAN8 also exist in the EBCDIC Country Extended Code Pages. In ROMAN8 there are nine characters which have no similar EBCDIC character, and six undefined characters. Since there are no undefined characters in the EBCDIC Country Extended Code Pages, 15 characters in EBCDIC have no look-alike in ROMAN8. For these characters a one-to-one mapping has been defined as shown in Table D-1.

**Table D-1. ROMAN8 to EBCDIC Mapping**

| dec. | oct. | hex. | | ROMAN8 | | EBCDIC |
|---|---|---|---|---|---|---|
| 169 | 251 | A9 | ` | Grave Accent | \| | Logical OR |
| 170 | 252 | AA | ^ | Circumflex Accent | ¬ | Logical NOT |
| 172 | 254 | AC | ~ | Tilde Accent | $^{2}$ | Superscript 2 |
| 175 | 257 | AF | ₤ | Italian Lira Sign | $^{3}$ | Superscript 3 |
| 177 | 261 | B1 | | Presently Undefined | µ | MU Character |
| 178 | 262 | B2 | | Presently Undefined | ‗ | Double Underline |
| 235 | 353 | EB | Š | Uppercase S Caron | Ý | Uppercase Y Acute |
| 236 | 354 | EC | š | Lowercase s Caron | ý | Lowercase y Acute |
| 238 | 356 | EE | Ÿ | Uppercase Y Umlaut | ı | Lowercase i Without Dot |
| 242 | 362 | F2 | | Presently Undefined | ¸ | Cedilla |
| 243 | 363 | F3 | | Presently Undefined | ¶ | Paragraph Sign |
| 244 | 364 | F4 | | Presently Undefined | ® | "Registered" Sign |
| 245 | 365 | F5 | | Presently Undefined | ¾ | Three Quarters |
| 246 | 366 | F6 | — | Long Dash | SHY | Syllable Hyphen |
| 252 | 374 | FC | ■ | Solid | • | Middle Dot |

For the Hewlett-Packard KANA8 character set, which supports KATAKANA, the mapping to and from EBCDIC is defined by Japanese Industrial Standards (JIS) and IBM.

In all languages, the character mappings defined and implemented on the HP 3000 are such that any character mapped from any Hewlett-Packard 8-bit character set to EBCDIC and then back again, or vice versa, will result in the original character value. A complete listing of the Hewlett-Packard 8-bit character set to EBCDIC mappings and vice versa can be obtained by running NLUTIL.PUB.SYS.

The mappings can be made available to a program by the NLINFO intrinsic item 13 or 14. The mappings are used by the NLTRANSLATE intrinsic, which performs the Hewlett-Packard 8-bit to EBCDIC translation or the reverse. The CTRANSLATE intrinsic maps USASCII to EBCDIC (and vice versa) and maps JISCII to EBCDIC (and vice versa). For the languages NATIVE-3000 and KATAKANA, there is no difference between the mappings produced by NLTRANSLATE and CTRANSLATE.

# PERIPHERAL CONFIGURATION E

Native Language Support (NLS) relies on the use of 8-bit character sets to encode alphabetic, numeric, and special characters required for the proper representation of native languages. Two character sets are available, ROMAN8 and KANA8. This Appendix explains how to configure various printers and terminals supported on the HP 3000 for 8-bit operation, so that ROMAN8 or KANA8 characters may be entered and displayed.

Most Hewlett-Packard terminals and printers are designed for 8-bit operation. Some have limitations which are listed as "Notes" at the end of this Appendix. A listing of relevant notes is included with the instructions for each peripheral, and the peripherals to which such notes apply are listed in Table E-2.

## NLS Peripheral Support Summary

Tables E-1, E-2, and E-3 contain information on which peripherals are fully supported, those that have limited support, and those that are not supported.

Table E-1. Peripherals Fully Supported in 8-Bit Operation - All Language Options

| Model/Type | Conforms To Processing Standard | Supports Full ROMAN8 | Supports Old ROMAN8 |
|---|---|---|---|
| HP 150 PC/As Terminal | YES | YES | YES |
| HP 2392A Terminal | YES | NO | YES |
| HP 2563A Printer | YES | YES | YES |
| HP 2621B Terminal | YES | NO | YES |
| HP 2622J Terminal | YES | YES* | N/A* |
| HP 2623J Terminal | YES | YES* | N/A* |
| HP 2625A Terminal | YES | YES | YES |
| HP 2627A Terminal | YES | NO | YES |
| HP 2628A Terminal | YES | YES | YES |
| HP 2700 Terminal | YES | NO | YES |
| HP 2932A Printer | YES | YES | YES |
| HP 2933A Printer | YES | YES | YES |
| HP 2934A Printer | YES | YES | YES |

* Supports KANA8 rather than ROMAN8.

Table E-2. Peripherals With Limited Support in 8-Bit Operation

| Model/Type | Conforms To Processing Standard | Supports Full ROMAN8 | Supports Old ROMAN8 |
|---|---|---|---|
| HP 2382A Terminal | NO | NO | YES |
| HP 2608A Printer | NO | NO | YES |
| HP 2608S Printer | NO | NO | YES |
| HP 2622A Terminal | NO | NO | YES |
| HP 2623A Terminal | NO | NO | YES |
| HP 2626A Terminal | NO | NO | YES |
| HP 2626W Terminal | NO | NO | YES |
| HP 2631B Printer | NO | NO | YES |
| HP 2635B Prntr/Term | NO | NO | YES |
| HP 2645J Terminal | NO | YES* | N/A* |
| HP 2680A Printer | NO | NO | YES |
| HP 2688A Printer | NO | YES | YES |

* Supports KANA8 rather than ROMAN8.

Table E-3. Peripherals Not Supported in 8-Bit Operation

| Model/Type | Conforms To Processing Standard | Supports Full ROMAN8 | Supports Old ROMAN8 |
|---|---|---|---|
| HP 2624B Terminal | NO | NO | NO |
| HP 2687A Printer | YES | NO | NO** |

** This printer functions correctly in 8-bit operation (it has no 7-bit operation). However, much of the ROMAN8 character set is not implemented, and KANA8 is unavailable. Some of Roman Extension is not implemented; but 8-bit characters with some of the Roman Extension values print in a degraded fashion (for example, accented vowels print as the corresponding vowel without accent, and the international currency symbol prints as "0").

## Specifics of 7-Bit Support

No peripherals are supported in 7-bit Native Language operation.

All peripherals are supported in 7-bit USASCII operation, though the non-USASCII characters are then unavailable. This includes the devices not listed at all in the preceding tables, because they are devices which have only 7-bit operation.

If 8-bit data is sent to a device configured for 7-bit USASCII operation, those characters with the eighth bit on will be displayed as unrelated (but predictable) USASCII characters or else as blanks, depending on the device. For example, an "à" displays as "H" on a 2645A terminal.

This Appendix contains specific information on each device supported in 8-bit mode to help configure these peripherals to utilize NLS capabilities.

## NLS Peripheral Support Details

There are two ways to access ROMAN8 characters not on the keyboard.

From many of the terminal keyboard layouts (for example, French and Spanish), you can access a few ROMAN8 characters (certain accented vowels) from the standard keyboard by using mutes. Enter a non-spacing diacritical character (such as an accent mark or circumflex), then the unaccented vowel. The result on the screen is a single, merged character; usually, a single, merged character is transmitted to the system. (See Notes 7 and 10 for some of the peripherals.)

Accessing ROMAN8 or KANA8 characters that do not appear on your keyboard can be accomplished by using "CTRL N"/"CTRL O", "CTRL ."/"CTRL ,", or Extend char, depending on the terminal. If your terminal uses CTRL N (or "shifting out"), please consult Notes 1-4 at the end of this Appendix.

## HP 150 P.C. as a Terminal

### Requirements

None. ROMAN8 character set is standard.

### Character Set Supported

ROMAN8

### Configuring For 8-Bit Operation

| | |
|---|---|
| Global Configuration | Language = Language of the keyboard |
| Port1 or Port2 | Parity = None<br>DataBits = 8<br>Check Parity = No |
| Terminal Configuration | ASCII 8-Bits = Yes |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC) |

### Typing ROMAN8 Characters Not On The Keyboard

Access the ROMAN8 characters not on the national keyboard by pressing Extend char, holding it down while pressing one of the other keys. Most of the accented vowels, as well as the Spanish "Ñ" or "ñ", are accessed from most of the national keyboards by means of mutes. The mute is a diacritical mark such as an accent, circumflex, or diaeresis. Enter a non-spacing diacritical character (if it is not on the keyboard layout, press Extend char), then the unaccented vowel (or "Ñ" or "ñ"). The screen displays a single, merged character, and a single, merged character is transmitted to the system. The non-spacing diacritical character is not displayed on the screen until the second character is typed.

### Notes

None.

## HP 2382A Terminal

### Requirements

Option 001, 002, 003, 004, 005, 006 or 007 (National keyboard and ROM).

### Character Set Supported

USASCII plus Roman Extension

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None<br>Chk Parity = No |
| Terminal Configuration | ASCII 8-Bits = Yes<br>Language = Language of the keyboard layout. |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

To configure the terminal for 8-bit operation as the default, set switches A5=up, A6=down, A7=up, B1=down.

### Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish and LANGUAGE=FRANCAIS azM, FRANCAIS qwM, or ESPANOL M, some Roman Extension characters (certain accented vowels) are accessible from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character. With a national keyboard, the USASCII characters, which are replaced on the keyboard, cannot be entered, but they can be displayed when received from the system.

Access the Roman Extension characters not on the keyboard by shifting out the keyboard. Enter CTRL N to do so. Enter CTRL O to return to the usual keyboard layout.

### Notes

1,2,4,5,6,7,9.

## HP 2392A Terminal

### Requirements

None. A subset of the ROMAN8 character set is standard.

### Character Set Supported

A subset of ROMAN8 (the last two columns of the ROMAN8 table are missing).

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity/DataBits = None/8 |
| Terminal Configuration | Keyboard = National layout of keyboard.<br>Language = Language in which terminal messages and labels are to appear |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing ROMAN8 Characters Not On Keyboard

Some ROMAN8 characters (certain accented vowels) are accessible from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character, and a single, merged character is transmitted to the system (in both character and block mode).

ROMAN8 characters not on the keyboard are accessible by pressing Extend char, holding it down while pressing another key. Most accented vowels are accessed via mute character combinations. The mute character itself is accessed via Extend char, and the vowel from the standard keyboard. The placement of extended characters is in Appendix B of the *HP 2392A Display Station Reference Manual* (02392-90001).

### Notes

None.

## HP 2563A Printer

### Requirements

\image 2 None. ROMAN8 character set is standard. (KANA8 is available with Option #002.)

### Character Set Supported

ROMAN8, KANA8

### Configuring For 8-Bit Operation

| | |
|---|---|
| Printer | Set primary character set = 20 (ROMAN8) or = 21 (KANA8) via the switches on the front panel. If the printer has a serial interface, set DataBits = 8, Parity = None. These configurations can also be done programmatically with escape sequences. |
| MPE I/O Configuration | For serial interface, configure the printer on the HP 3000 as Termtype = 20 (8-bits of data). On a Multipoint line, use Termtype = 18 or 22. For HPIB interface, use Type = 32, Subtype = 9. This permits programmatic reconfiguration via escape sequences. |

### Notes

None.

## HP 2608A/HP 2608S Printers

### Requirements

Option 001 and 002 for KANA8.
Option 002 for Roman Extension.

### Character Set Supported

KANA8
USASCII plus Roman Extension

### Configuring For 8-Bit Operation

Set switches on front panel: USASCII + RomExt
Primary Language = 0000
Secondary Language = 1111

KANA8
Primary Language = 1110
Secondary Language = 0011

On the HP 2608S only, a program can also set these values via escape sequences.

MPE I/O Configuration Termtype = 20 or 22.

### Notes

9,11.

# HP 2621B Terminal

## Requirements

Option 001,002,003,004,005,006 and/or 010 (National keyboard and/or extended character set ROMs).

Option 101,102,103,104,105,106 and/or 110 (Extended national keyboard and/or ROMs).

## Character Set Supported

USASCII plus Roman Extension

## Configuring For 8-Bit Operation

| | |
|---|---|
| Set switches P0,P1,P2: | Set to 0,1,0 (down,up,down) |
| Set switches L0,L1,L2: | Set to language of keyboard layout (see *HP 2621B Manual* (02620-90062), for settings for keyboard layout), and switch 5 of the left-hand group = 0 to activate the keyboard of that language. |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

## Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish, a few Roman Extension characters (certain accented vowels) are accessible from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character, and a single, merged character is transmitted to the system.

Roman Extension characters not available on the keyboard (except those available via mutes) cannot be entered, but they can be displayed when received from the system.

The USASCII characters which are replaced on the native keyboard are available after pressing f1 in the "modes" level (an asterisk will appear next to the "USASCII" label for this function key). This causes the keyboard to become the standard USASCII layout. Press f1 again (the asterisk will disappear) to return to the native keyboard.

## Notes

10.

## HP 2622A/HP 2623A Terminals

### Requirements

Option 001, 002, 003, 004, 005, 006 or 202 (National keyboard and/or extended character set ROMs).

### Character Set Supported

USASCII plus Roman Extension

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None<br>Chk Parity = No |
| Terminal Configuration | ASCII 8-Bits = Yes<br>Language = Language of the keyboard layout. |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish and LANGUAGE=FRANCAIS azM, FRANCIAS qwM, or ESPANOL M, a few Roman Extension characters (certain accented vowels) can be accessed from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character. Access the USASCII characters replaced on a national keyboard by pressing Shift and one of the numeric pad keys.

Access the Roman Extension characters not on the keyboard by shifting out the keyboard. Enter CTRL N to do so. Enter CTRL O to return to the usual keyboard layout.

### Notes

1,2,4,5,6,7,9.

## HP 2622J/HP 2623J Terminals

### Requirements

None. KATAKANA is standard.

### Character Set Supported

KANA8.

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None<br>Chk Parity = No |
| Terminal Configuration | ASCII 8-Bits = Yes |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing KANA8 Characters Not On The Keyboard

Access the KANA8 characters not in JISCII by pressing the "KATAKANA" key to enter KATAKANA mode. Press the Caps key to return to the JISCII keyboard.

### Notes

None.

# HP 2625A/HP 2628A Terminals

## Requirements

None. ROMAN8 character set is standard.

## Character Set Supported

ROMAN8

## Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None<br>Chk Parity = No<br>DataBits = 8 (in Multipoint: Code = ASCII8). |
| Terminal Configuration | ASCII 8-Bits = Yes |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC) |

## Typing ROMAN8 Characters Not On The Keyboard

If the keyboard layout is French or Spanish, a few ROMAN8 characters (certain accented vowels) can be accessed from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character, and a single, merged character is transmitted to the system (in both character and block mode).

Access the ROMAN8 characters not on the keyboard by pressing CTRL C to enter "Extended Characters Mode." When not using the USASCII keyboard, this may not actually be the key labeled period (.), but the period key for the USASCII keyboard. A keyboard layout showing the placement of extended characters is located in the *User's Manual for the HP 2625A Dual-System Display Terminal and HP 2628A Word-Processing Terminal* (02625-90001). Enter "CTRL," to return to the usual keyboard layout.

## Notes

None.

## HP 2626A/HP 2626W Terminals

### Requirements

Option 001, 002, 003, 004, 005, 006 or 201 (National keyboard and/or extended character set ROMs).

### Character Set Supported

USASCII plus Roman Extension

### Configuring For 8-Bit Operation

| | |
|---|---|
| Global Configuration | Language = Language of keyboard layout. |
| Datacomm Configuration | Parity = None<br>Chk Parity = No<br>DataBits = 8 (In Multipoint: Code = ASCII8). |
| Terminal Configuration | ASCII 8-Bits = Yes<br>ESC) A = RomanExt*<br>Alternate Set = A. |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

*On some versions of the 2626W the RomanExt and BOLD alternate sets are exchanged. Press IDENTIFY ROMS; if CHARACTER ROMS show 1818-1916 and 1818-1917, Rev.A, set ESC ) A = BOLD to access ROMAN8.

### Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish and LANGUAGE=FRANCAIS azM, FRANCAIS qwM, or ESPANOL M, a few Roman Extension characters (certain accented vowels) can be accessed from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character. Access the USASCII characters replaced on a national keyboard by pressing [Shift] and one of the numeric pad keys.

Access the Roman Extension characters not on the keyboard by shifting out the keyboard. Enter [CTRL]N to do so. Enter [CTRL]O to return to the usual keyboard layout.

### Notes

1,2,3,5,6,7,8,9.

## HP 2627A Terminal

### Requirements

None. Roman Extension is standard.

### Character Set Supported

USASCII plus Roman Extension

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None<br>Chk Parity = No |
| Terminal Configuration | Language = Language of keyboard layout.<br>ASCII 8-Bits = Yes |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish and LANGUAGE=FRANCAIS azM, FRANCAIS qwM, or ESPANOL M, a few Roman Extension characters (certain accented vowels) can be accessed from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character, and a single, merged character is transmitted to the system (in both character and block mode).

Access the USASCII or Roman Extension characters not on the keyboard by putting the keyboard in Foreign Characters mode. Enter "CTRL." to do so. Find the keyboard location of any desired character in the *HP 2627A Display Station Reference Manual* (02627-90002). Enter "CTRL," to return to the usual keyboard layout.

### Notes

4.

## HP 2631B Printer

### Requirements

Roman Extension and KATAKANA are now standard. Formerly option #008 (KATAKANA) or #009 (Roman Extension) was required.

### Character Set Supported

KANA8
USASCII plus Roman Extension

### Configuring For 8-Bit Operation

Set the rocker switches on the Serial I/O Interface PCA (S2, inside the printer) as follows:

| | |
|---|---|
| Switches 6,7 | Set to 00 (both open).<br>(Received eighth bit passed). |

Set the rocker switches on the Printer Logic PCA (inside the printer) as follows:

| | |
|---|---|
| In 1st Group of 7 | Set Switch 7 = 0 (Open) (8-bit Datacomm). |
| In 2nd Group of 10 | Set Switches 1-5 = 11111(USASCII); 10110 (JISCII).<br>Set Switches 6-10 = 10001(Roman Extension); 10101(KATAKANA). |
| Front Panel Switches | Parity = 00 (None). |
| MPE I/O Configuration | Subtype = 14 (not supported if connection is ATC).<br>Terminal Type = 20 or 22. |

### Notes

9,11,14.

## HP 2635B Printer/Terminal

### Requirements

Roman extension is now standard. Formerly one of options #001, 002, 003, 004, 005 or 006 (national keyboards) was required.

### Character Set Supported

USASCII plus Roman Extension

### Configuring For 8-Bit Operation

Set the rocker switches on the Serial I/O Interface PCA (S2, inside the printer) as follows:

| | |
|---|---|
| Switches 6,7 | Set 00 (both open).<br>(Received eighth bit passed). |

Set the rocker switches on the Printer Logic PCA (inside the terminal) as follows:

| | |
|---|---|
| In 1st Group of 7 | Set Switch 7 = 0 (Open) (8-bit Datacomm). |
| In 2nd Group of 10 | Set Switches 1-5 = 11111 (USASCII).<br>Set Switches 6-10 = 10001 (Roman Extension). |

Set the rocker switches on the keyboard PCA (inside the terminal) as follows:

| | |
|---|---|
| Set Switches 4-8 | Set to language of terminal keyboard. Refer to the *HP 2630B Family Reference Manual* (02631-90918) for a list of keyboard layouts and the corresponding switch settings. |
| Front Panel Switch | Parity = None. |
| MPE I/O Configuration | Terminal Type = 15. |

### Notes

1,2,5,7,9,11.

## HP 2645J Terminal

### Requirements

None. KATAKANA is standard.

### Character Set Supported

KANA8

### Configuring For 8-Bit Operation

| | |
|---|---|
| Datacomm Configuration | Parity = None |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing KANA8 Characters Not On Keyboard

Access the KANA8 characters not in JISCII by pressing the "KATAKANA" key to enter KATAKANA mode. Press the KATAKANA key again to return the keyboard to its JISCII layout. Alternatively, press the right Shift key (once by itself) to enter KATAKANA mode, and the left Shift key to exit from it.

### Notes

9,12.

## HP 2680A Printer

### Requirements

Environment files ending in "x" for USASCII plus Roman Extension.
Environment files ending in "K" for KANA8.

### Character Set Supported

USASCII plus Roman Extension
KANA8

### Configuring For 8-Bit Operation

Use the environment files ending in "x" (for USASCII plus Roman Extension) or those ending in "K" (for KANA8).

### Notes

9,11.

## HP 2688A Printer

### Requirements

Environment files COURxA, GOTHxA, LP88, PICAxA, PRESxA, ROMPxA, SCRPRA.

### Character Set Supported

ROMAN8

### Configuring For 8-Bit Operation

Use one of the environment files listed above for support of ROMAN8.

### Notes

9,11.

## HP 2700 Terminal

### Requirements

None. Roman Extension is standard.

### Character Set Supported

USASCII plus Roman Extension.

### Configuring For 8-Bit Operation

| | |
|---|---|
| Pot1 or Port2<br>Configuration | Parity/DataBits = None/8.<br>Chk Parity = No |
| Terminal Configuration | Language = Language of keyboard layout.<br>ASCII 8-Bits = ON. |
| MPE I/O Configuration | Terminal Type = 10 (12 if connection is ATC). |

### Typing USASCII/Roman Extension Characters Not On Keyboard

If the keyboard layout is French or Spanish and `LANGUAGE=FRANCAIS azM, FRANCAIS gwM,` or `ESPANOL M,` a few Roman Extension characters (certain accented vowels) can be accessed from the standard keyboard by using mutes. Enter a non-spacing diacritical character, then the unaccented vowel. The screen displays a single, merged character, and a single, merged character is transmitted to the system (in both character and block mode).

Access the USASCII or Roman Extension characters not on the keyboard by putting the keyboard in Foreign Characters mode. Enter "[CTRL]." to do so. Find the keyboard location of any desired character using the algorithm in the *HP 2700 Family Alphanumeric Reference Manual* (02703-90003). Enter "[CTRL]," to return to the usual keyboard layout.

### Notes

3,13.

## HP 2932A/HP 2933A/HP 2934A Printers

### Requirements

None. ROMAN8 and KANA8 character sets are standard.

### Character Set Supported

ROMAN8, KANA8

### Configuring For 8-Bit Operation

Printer

From the front panel, in the Printer Print Settings, set Primary Character Set = 1 (ROMAN8) or = 2 (KANA8).

For serial interface, in the Interface Data Settings, set DataBits = 8, Parity = None.

For Multipoint, set Parity = None, Code = ASCII8.

These can also be done programmatically with escape sequences.

MPE I/O Configuration

For serial interface, configure the printer on your HP 3000 as Termtype = 20 (8 bits of data) (not supported via ATC connection or ADCC with HIOTERM0.) On a Multipoint line, use Terminal Type = 18 or 22.

### Notes

None.

## Notes

The following Notes apply to the peripherals covered in this Appendix. Refer to the description of each peripheral for a list of which Notes apply to it.

1. When "CTRL N" (shift out) and "CTRL O" (shift in), are used to shift the keyboard out for Roman Extension, they are transmitted to the system when the terminal is in character mode. This results in superfluous data in the byte stream sent to the system.
   (HP 2382, 2622, 2623, 2626, 2635)

2. When shift out and shift in are sent to the terminal, they have no effect on the active character set (as expected by some software), but they do affect subsequent keyboard operation, as if they had been typed in.
   (HP 2382, 2622, 2623, 2626, 2635)

3. When the keyboard is shifted out, (in Foreign Characters mode for the HP 2700 family), the space bar sends %240 instead of %40, and the DEL key sends %377 instead of %177.
   (HP 2626, 2700)

4. When the keyboard is shifted out (in Foreign Characters mode for the HP 2627), the space bar sends %240 instead of %40, and the DEL key sends nothing. This has been fixed in the most recent versions of the 2622 and 2623 terminals. These will show as ROMs 1818-3199/3203 with Date Code 2313 or later (2622), and 1818-3223/3228 with Date Code 2335 or later (2623).
   (HP 2382, 2622, 2623, 2627)

5. If "ESC)B" or "ESC)C" is entered or transmitted to the terminal, the alternate character set will be redefined (for example, to line draw or math). This will cause all would be Roman Extension characters, whether displayed on the terminal or entered via one of the methods listed above, to appear as the corresponding line draw or math symbols (or blanks, if that alternate set is not present in the terminal). To remedy this, enter "CTRL O ESC)A" (on the HP 2626A, reset Alternate Set to A in the TERMINAL CONFIGURATION menu). Note that data entered or displayed while the terminal has another alternate character set defined is correct internally even though it may not display correctly on the terminal.
   (HP 2382, 2622, 2623, 2626, 2635)

6. When the terminal is in block mode and one or more Roman Extension characters are entered (for example, ü), then Enter is pressed, what is transmitted to the system, and written to the buffer of the program reading from the terminal, is "ESC))ü". This is the terminal's way of compensating for Note 5. It means that when the data is sent back again from the computer, "ü" will always display this way, and not as the corresponding line draw or math symbol. It also means that there may be more information in the program buffer than the user or the programmer is expecting, or there is less room in that buffer for other information. Note that if the terminal is controlled by VPLUS/3000, it strips out the escape sequence before passing the data on to the calling program's buffer (and from there to the data file or data base).
   (HP 2382, 2622, 2623, 2626)

7. For the languages FRANCAIS azM, FRANCAIS qwM, and ESPANOL M when mutes are used and the terminal is in character mode, two characters are sent to to the system although a single, merged character appears on the screen. This means that an incorrect two-byte representation of the accented character will be received by the program or file. The next time they are displayed the terminal will put them back together, provided the terminal is still configured for FRANCAIS azM, FRANCAIS qwM, or ESPANOL M. In block mode a single character (the correct ROMAN8 code for the merged character) is sent to the system. (HP 2382, 2622, 2623, 2626, 2635)

8. When softkey labels which contain extended characters (in the range %200-%377) are received from the system, the extended characters are lost and the inverse video is turned off on the label. (HP 2626)

9. This device does not actually support 8-bit character sets, but simulates them by handling two 7-bit character sets, a primary and an alternate. Legitimate data from real alternate character sets (line draw or math) cannot be used in a supported (standard) way together with general ROMAN8 (KANA8) data because these devices treat Roman Extension (KATAKANA) as an alternate character set, in 8-bit mode. All alternate character sets are addressed by codes with the eighth bit set to one; Roman Extension (KATAKANA) must share this position with the other alternate sets through the use of escape sequences ("ESC)*x*"), and, on the terminals, shift-in/shift-out are unsuitable for invoking alternate sets. The practical result of this is that NLS will not support the use of alternate character sets together with ROMAN8 (KANA8) data on these devices. Configure the device for 8-bit mode as documented, then limit the data to (old) ROMAN8 (KANA8). (HP 2382, 2608, 2622A, 2623A, 2626, 2631, 2635, 2645J, 2680, 2688)

10. For the French and Spanish keyboards, when mutes are used and a mute diacritical is entered followed by a space, the ROMAN8 codes for the diacritical and the space are both transmitted to the system, not just the ROMAN8 character for the diacritical. (HP 2621B)

11. When a shift-out character is sent to the printer, it causes subsequent data (until a shift-in is sent) to be selected from the alternate character set, whether or not the eighth bit is on. (HP 2608, 2631, 2635, 2680, 2688)

12. When the system sends an 8-bit character the terminal shifts into KATAKANA mode until a 7-bit character is received. For example, switching terminal speed with the MPE :SPEED command sometimes results in the receipt of an 8-bit character from the system. The user will need to exit KATAKANA mode before entering "MPE" to signal that the speed has been changed. (HP 2645J)

13. When the terminal is in Block Format mode (for example, under control of VPLUS/3000), an attempt to read the character %254 (tilde accent in ROMAN8) from an input field causes the read to hang. (HP 2700)

14. Versions of the 2631B with Printer Logic PCA #02631-60225 are not supported, because switch 7 (8 bit datacomm) is ignored. It is possible to configure 8 bit datacomm on this PCA programmatically via an escape sequence; but the program must do so before every data transfer. (HP 2631B)

# CONVERTING 7-BIT TO 8-BIT DATA F

Many Hewlett-Packard peripherals can be configured for 7-bit operation with one of the European language national substitution character sets. These peripherals must be converted to 8-bit operation to access Native Language Support (NLS) capability. NLS requires the use of 8-bit character sets which include USASCII and native language characters.

NLS for western European languages is based on the ROMAN8 character set in which the additional characters required are assigned to unique values between 128 and 255. It requires eight bits to hold the value of a ROMAN8 character. All the special European characters are accessible in ROMAN8 without losing any of the USASCII characters.

The 7-bit national substitution sets do not offer a full complement of characters. New characters replace existing ones. For example, in FRANCAIS the graphic symbol "#" is not available. In Spanish and French, even the substitutions made are not sufficient to obtain all the necessary new characters. The use of mute characters is required. Mute characters provide a single graphic on the terminal screen or paper for two bytes of storage and two keystrokes. For example, an "é" in Spanish or French would be produced with an accent mark plus an "e", whereas ROMAN8 contains the "é" as a single character. In any one language, the graphic symbols for other European countries are not available at all. For example, a French user does not have access to the necessary characters to properly address a letter to someone in Germany. The ROMAN8 8-bit character set eliminates these problems.

## National Substitution Sets

Many Hewlett-Packard peripherals support the 7-bit national substitution sets for the following languages. (They are listed here as they appear on the terminal configuration menus of the terminals which support them):

SVENSK/SUOMI
FRANCAIS
ESPANOL M
DANSK/NORSK
DEUTSCH
ESPANOL
FRANCAIS M
UK
ITALIANO (On a few devices only.)

These are 7-bit national substitution character sets or languages in which one or more of 12 USASCII graphic symbols are replaced by other graphic symbols required for the national language being used. The same 7-bit internal code is displayed as a different symbol than that assigned to it by USASCII. For example, in USASCII the decimal value 35 is assigned to the graphic symbol "#"; but in the FRANCAIS national substitution set, the same decimal value 35 is assigned to the graphic symbol "£".

Users who have been using these (HP 262X) terminals in 7-bit operation for many years may have a substantial investment in data which is encoded in one of these 7-bit national substitution character sets. Hewlett-Packard is making several conversion utilities available to convert this data to ROMAN8.

## Conversion Utilities

Because NLS involves using full 8-bit character sets for all data, customers wanting to use the facility will need to configure their peripherals for 8-bit operation. (This is not possible for the HP 264X terminals.) The national substitution characters, if input on a terminal configured for 7-bit operation, will not display correctly on a terminal or printer configured for 8-bit operation.

Several utilities are available to convert existing data that has been input with an HP 262X terminal configured for 7-bit operation. Refer to Table F-1 for a listing of these utilities. The premise of these utilities is that users will run them once for each file which needs converting, and will configure all their peripherals for 8-bit operation. Thereafter, peripherals will only be used in 8-bit operation.

Table F-1. Conversion Utilities by File Type

| File Type | Utility to be Used for Conversion |
|---|---|
| EDITOR files | N7MF8CNV (text option) |
| Other MPE files which are all text | N7MF8CNV (text option) |
| MPE files in which text data is organized in fields which need to start in fixed columns | N7MF8CNV (text option; data option if language is FRANCAIS M or ESPANOL M) |
| MPE files which include some non-text data (for example, integer or real) | N7MF8CNV (data option) |
| IMAGE/3000 data bases | I7DB8CNV |
| VPLUS/3000 forms files | V7FF8CNV |
| HPWORD files | HPWORD internal files have always been based on a subset of ROMAN8. No conversion is necessary. |
| TDP files | Run N7MF8CNV and then change back whatever command backslash is converted to in the chosen language in case you need the command backslash for embedded TDP commands. |

## Conversion Algorithm

The conversion utilities convert records or fields from files which are assumed to have been created at an HP 262X terminal configured for 7-bit operation, and for a language other than USASCII. The conversion is from the HP 262X implementation of a European 7-bit substitution character set to the 8-bit ROMAN8 character set. This involves converting the values with which certain characters are stored in the file. Before conversion, the file should look correct on an HP 262X terminal configured for 7-bit operation with the appropriate substitution set. After conversion the file will look correct on any terminal configured for 8-bit operation.

Records and/or fields from files of all types are converted using the same algorithm which is expressed in Figure F-1. The conversion affects only the 12 characters shown in the table. All other characters remain unchanged.

To use this table, find the desired national substitution set on the left. The uppermost row shows the 7-bit decimal values for which substitutions may have been made. There are two rows of information opposite each national substitution set. The upper row shows the graphic assigned in 7-bit operation and the lower row the decimal value assigned the graphic in ROMAN8 after using the conversion algorithm.

When certain FRANCAIS M and ESPANOL M characters are followed immediately by certain other characters, the two-character combination is converted to a single ROMAN8 character, and the field or record being converted is padded at the end with a blank:

**Table F-2. Special Two-Character Combination Conversion**

| | |
|---|---|
| FRANCAIS M | ^(94) followed by a, e, i, o, or u is converted to @(192), ê(193), î(209), ô(194), or û(195). |
| | +(126) followed by a, e, i, o, or u is converted to L(204), ë(205), ï(221), ö(206), ü(207). |
| | +(126) followed by A, O, or U is converted to X(216), Ö(218), or Ü(219). |
| ESPANOL M | ((39) followed by a, e, i, o, or u is converted to D(196), é(197), í(213), ó(198), or ú(199). |

If these characters are followed by any other character, they are converted to their ROMAN8 equivalent as shown in Figure F-1.

| National<br>Subst.Set | 35 | 39 | 64 | 91 | 92 | 93 | 94 | 96 | 123 | 124 | 125 | 126 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| USASCII | # | ’ | @ | [ | \ | ] | ^ | ‘ | { | \| | } | ~ |
| SVE/SUOMI | # | ’ | É | Ä | Ö | Å | Ü | é | ä | ö | å | ü |
| | 35 | 39 | 220 | 216 | 218 | 208 | 219 | 197 | 204 | 206 | 212 | 207 |
| DANSK/NORSK | # | ’ | @ | Æ | Ø | Å | ^ | ‘ | æ | ø | å | ~ |
| | 35 | 39 | 64 | 211 | 210 | 208 | 94 | 96 | 215 | 214 | 212 | 126 |
| FRANCAIS | £ | ’ | à | ° | ç | § | ^ | ‘ | é | ù | è | ¨ |
| | 187 | 39 | 200 | 179 | 181 | 189 | 170 | 96 | 197 | 203 | 201 | 171 |
| FRANCAIS M | £ | ’ | à | ° | ç | § | ^ | ‘ | é | ù | è | ¨ |
| | 187 | 39 | 200 | 179 | 181 | 189 | 170 | 96 | 197 | 203 | 201 | 171 |
| DEUTSCH | £ | ’ | § | Ä | Ö | Ü | ^ | ‘ | ä | ö | ü | ß |
| | 187 | 39 | 189 | 216 | 218 | 219 | 94 | 96 | 204 | 206 | 207 | 222 |
| U K | £ | ’ | @ | [ | \ | ] | ^ | ‘ | { | \| | } | ~ |
| | 187 | 39 | 64 | 91 | 92 | 93 | 94 | 96 | 123 | 124 | 125 | 126 |
| ESPANOL | # | ’ | @ | ¡ | Ñ | ¿ | ° | ‘ | { | ñ | } | ~ |
| | 35 | 39 | 64 | 184 | 182 | 185 | 179 | 96 | 123 | 183 | 125 | 126 |
| ESPANOL M | # | ´ | @ | ¡ | Ñ | ¿ | ° | ‘ | { | ñ | } | ~ |
| | 35 | 168 | 64 | 184 | 182 | 185 | 179 | 96 | 123 | 183 | 125 | 126 |
| ITALIANO | £ | ’ | @ | ° | ç | é | ^ | ù | à | ò | è | ì |
| | 187 | 39 | 64 | 179 | 181 | 197 | 94 | 203 | 200 | 202 | 201 | 217 |

**Figure F-1. Character Conversion Data**

## Conversion Procedure

To convert 7-bit substitution data to 8-bit ROMAN8 data:

1. Determine which files need to be converted. A file must be converted if the data was input from an HP 262X terminal configured for 7-bit operation or for a national substitution set other than US-ASCII.
2. Determine the national substitution set ("language" on the terminal configuration menu) from which the conversion should be done for each file. This is the language the HP 262X terminal was configured for at the time the file data was input.
3. Determine which utility should be used to convert each file, refer to Table F-1.
4. Back up all files to be converted (store to tape or perform a SYSDUMP).
5. Run each utility, supplying it with the language and filenames as determined above. Instructions for running each utility are found at the end of this Appendix.
6. Configure all terminals and printers for 8-bit operation. (At least one terminal must already be configured for 8-bit operation when the V7FF8CNV utility is run.) Refer to Appendix E, "Peripheral Configuration."

The sample dialog, on the following page, is from a session executing N7MF8CNV for both text and data files.

```
:RUN N7MF8CNV.PUB.SYS
HP European 7-Bit character sets are:

   1. SVENSK/SUOMI
   2. DANSK/NORSK
   3. FRANCAIS M
   4. FRANCAIS
   5. DEUTSCH
   6. UK
   7. ESPANOL M
   8. ESPANOL
   9. ITALIANO

From which character set should conversion be done: 5
File types which can be converted are:

   1. MPE text files (each record converted as one field).
   2. MPE data files (define fields; only defined fields are converted).
   3. Test Conversion.

Type of file to be converted: 1

Name of text file to be converted: ABC

   112 records converted in ABC

Name of text file to be converted:[Return]

File types which can be converted are:

   1. MPE text files (each record converted as one field).
   2. MPE data files (define fields; only defined fields are converted).
   3. Test Conversion.

Type of file to be converted: 2

Name of data file to be converted: XYZ

Please supply one at a time the field to be converted (first
   Start, Length: 1,12
   Start, Length: 15,30
   Start, Length: 61,6
   Start, Length:[Return]

Data file XYZ: fields to be converted are:

 1,    12
15,    30
61,     6
Correct?[Return]
  287 records converted in XYZ

Name of data file to be converted:[Return]

File types which can be converted are:

  1. MPE text files (each record converted as one field).
  2. MPE data files (define fields; only defined fields are converted).
  3. Test Conversion.

Type of file to be converted:[Return]

HP European 7-Bit character sets are:

  1. SVENSK/SUOMI
  2. DANSK/NORSK
  3. FRANCAIS M
  4. FRANCAIS
  5. DEUTSCH
  6. UK
  7. ESPANOL M
  8. ESPANOL
  9. ITALIANO

From which character set should conversion be done:[Return]

END OF PROGRAM
:
```

## N7MF8CNV Utility

N7MF8CNV converts data in EDIT/3000 and other MPE text and data files from a Hewlett-Packard 7-bit national substitution character set to ROMAN8. The user is prompted for language and file type (text or data). For a data file, the user will be prompted on each file for the starting position and length of each field (portion of a record) to be converted. For a text file, each record is converted as one field.

The user is prompted for the name of each file to be converted. Files are read one record at a time; each record is converted (or certain fields of it are converted for data files), and the result is written to a new temporary file. When all records have been read, converted, and written to the new file, the old (unconverted) copy is deleted, and the new one is saved in its place. An exception to this is KSAM/3000 files, which are converted in place, rather than written to a new temporary file. A count of the number of records read and converted is displayed on $STDLIST.

This utility will not convert files containing bytes with the eighth bit set. This situation probably indicates a misunderstanding or error. The likely causes are:

- File is not a text or data file.
- File is a data file for which the fields have been inaccurately located.
- File was created on a terminal configured for 8-bit operation.
- File has already been converted.

The maximum record length supported is 8192 bytes. The maximum number of fields supported in the records of a data file is 256.

If the file being converted contains user labels, these are copied to the new file without conversion. If a fatal error is encountered during the conversion (for example, 8-bit data or file system error found) the conversion stops, the old copy of the file is saved, and the new copy is purged. The data is unchanged. An exception to this is KSAM/3000 files. Since these are converted in place, some records may already have been modified. KSAM/3000 files (including key file) should be restored from the backup tape to ensure a consistent copy.

A [Ctrl]Y entered during conversion displays the number of records successfully converted and conversion continues. On variable length data files, if a field or portion of a field is beyond the length of the record just read, a warning is displayed and that field is not converted on that record. Other fields on the same record are converted, and processing continues with subsequent records. After each file has been converted, the user is prompted for another filename.

In addition to the text and data options, there is a test conversion option which shows how the conversion algorithm operates. The test conversion option must be run from a terminal configured for 7-bit operation with the chosen national substitution set. The user is instructed to enter a string, and the result of the conversion is displayed. The user does not have to switch back and forth between 7-bit and 8-bit operation to see the result. Each character converted is displayed as a decimal value in parentheses rather than graphically. Other characters are displayed unchanged.

At any point in the program, pressing [Return] exits the current program level at which the user is located. A [Return] in response to a request for the starting position and length of a field in a data file indicates that the definition of fields is complete, and the program proceeds with the conversion of the data file. A [Return] entered in response to a request for a text file name indicates the conversion of text files is complete; the program goes back to the question: "Type of file to be converted?".

## I7DB8CNV Utility

I7DB8CNV converts the character data in an IMAGE/3000 data base from an Hewlett-Packard 7-bit national substitution set to ROMAN8. The program is a special version of the `DBLOAD.PUB.SYS` program, and the conversion is done as part of a database load. The procedure for running I7DB8CNV is:

1. Enter :`RUN DBUNLOAD.PUB.SYS` to unload your database to tape.
2. Enter :`RUN DBUTIL.PUB.SYS,ERASE` to erase the data in your database.
3. Enter :`RUN I7DB8CNV.PUB.SYS` to convert the data and load it back into your database.

I7DB8CNV will request the following:

1. The 7-bit national substitution set from which the conversion is to be made.
2. The database name.
3. The utility prompts the user, `Convert all data fields of type X or U?`. `YES` or (Return) means "yes". If `NO` is entered, the user will be prompted in each data set for each field of type U or X.

   The single field in an automatic data set is not proposed for conversion. Whether or not its values are converted depends on the response to the item(s) through which it is linked to detail data set(s). At the end of each data set, the user is asked to confirm that the correct fields to be converted from that data set have been selected. Again, a (Return) is treated as a "yes" answer. Enter `N` or `n` to change the data fields in the data set to be converted.

I7DB8CNV then loads the database from tape. As each record is read, those fields which were selected have their data converted according to the algorithm for the 7-bit national substitution set which was selected at the beginning of the program.

I7DB8CNV will not allow 8-bit data (bytes with the high-order bit set) in the data fields it is trying to convert. The utility will not abort, but the field in question will not be converted, and a warning will be issued:

```
** 8-bit data encountered in item [itemname in DS data set]
```

If the program should abort for any reason during the conversion, the user must log on again to clear the temporary files used during the conversion process before running the program again.

The dialog on the following page is a sample run of the I7DB8CNV program.

```
:RUN I7DB8CNV.PUB.SYS
HP European 7-bit character sets are:
    1. SVENSK/SUOMI
    2. DANSK/NORSK
    3. FRANCAIS
    4. FRANCAIS M
    5. DEUTSCH
    6. U K
    7. ESPANOL
    8. ESPANOL M
    9. ITALIANO
From which character set should conversion be done: 2
WHICH DATA BASE:QWERTZ
Convert all fields of type U,X in all data sets (Y/N)? N
Data Set SET1 fields to be converted:
ITEM1         (Y/N)?[Return]
ITEM2         (Y/N)?[Return]
ITEM3         (Y/N)? N
ITEM4         (Y/N)?[Return]
Is Data Set SET1 correctly defined (Y/N)?[Return]
Data Set SET2 - Automatic Master
Data Set SET3 fields to be converted:
ITEM1         (Y/N)?[Return]
ITEM5         (Y/N)? N
ITEM6         (Y/N)? N
Is Data Set SET3 correctly defined (Y/N)?[Return]
DATA SET 1:   19 ENTRIES
DATA SET 2:    0 ENTRIES
DATA SET 3:   25 ENTRIES
END OF VOLUME 1, 0 READ ERRORS RECOVERED
DATA BASE LOADED
END OF PROGRAM
:
```

## V7FF8CNV Utility

V7FF8CNV converts text and literals in VPLUS/3000 forms files from a Hewlett-Packard 7-bit national substitution character set to ROMAN8. V7FF8CNV is a special version of FORMSPEC.PUB.SYS and is run the same way. Before running this utility back up the forms file (store to tape or perform a SYSDUMP), then:

1. Configure your terminal for 8-bit operation. (Refer to Appendix E, "Peripheral Configuration," for information on specific terminal configuration.)
2. Run V7FF8CNV.PUB.SYS, stepping through each form, field definition, save field, function key label. As each screen is presented on the terminal, 7-bit substitution characters have already been converted to their ROMAN8 equivalent.
3. If the data is correct, press Enter and proceed to the next screen. If not, correct the data, then press Enter to continue.
4. After all screens are converted, recompile the forms file as usual.

Conversion applies to substitution characters found in all source records in VPLUS/3000 forms files with the following exception: substitution characters for "[" and "]" are not converted in screen source records, since these indicate start and stop of data fields. The following would be converted:

- Text in screens
- Function key labels
- Initial values in save field definitions
- Initial values in field definitions
- Literals in processing specifications

## V7FF8CNV and Alternate Character Sets

Hewlett-Packard block-mode terminals which have the capability to handle all or part of ROMAN8 can be divided into two groups, based on how they handle alternate character sets when configured for 8-bit operation.

### GROUP ONE - HP 2392A, 2625A, 2627A, 2628A, 2700, and 150

Use shift-out and shift-in characters to switch back and forth between an 8-bit base character set and an 8-bit alternate character set. This is the standard for new Hewlett-Packard terminals and printers.

### GROUP TWO - HP 2622A, 2623A, 2626A, and 2382A

(Do not use an HP 2624A or HP 2624B as they are unable to handle 8-bit characters properly.) Group Two terminals use the eighth bit to switch back and forth between a 7-bit base character set and a 7-bit alternate character set. Therefore, it is not possible to get true 8-bit operation (ROMAN8) and use an alternate character set (for example, line draw) at the same time because the base character set is not really 8-bit, but 7-bit with the additional characters defined in the alternate character set. Using both 8-bit ROMAN8 characters and line draw in the same file is not recommended, since the user must continually redefine the alternate character set, switching back and forth between Roman Extension and the line drawing character set. Shift-out and shift-in are ignored by the terminal, which goes to the alternate character set when the high order bit is on.

Files using alternate character sets on one group of terminals will not display correctly on the terminals of the other group, even when terminals from both groups are configured for 8-bit operation.

Therefore, the use of characters from an alternate set affects the conversion procedure. If the forms file does contain characters from an alternate character set, choose one of the following alternatives:

1. Eliminate the use of alternate character sets (either with FORMSPEC or while running V7FF8CNV).
2. Define alternate character sets to appear correctly on Group One terminals. This happens automatically when V7FF8CNV is run from a Group One terminal. Characters from these alternate sets will appear as USASCII characters on a Group Two terminal.

## V7FF8CNV Operation

V7FF8CNV must be run on a terminal supported by VPLUS/3000 which supports display of all characters, enhancements and alternate characters sets used in the forms file. If alternate character sets are used, the HP 2392, 2625, 2627, 2628, 2700, or 150 are recommended.

The V7FF8CNV procedure is:

1. Configure your terminal type properly for 8-bit operation by using the settings recommended in Appendix E, "Peripheral Configuration."
2. Run `V7FF8CNV.PUB.SYS`. Respond to prompts for the terminal group and the national substitution set.
3. Press [Next] once to begin going through the forms file.
4. Press [Enter] after each screen until the end of the forms file is reached. Two exceptions to Step 4 are:
   - Enter Y in "Function Key Labels" on each FORM MENU and the GLOBALS MENU to see and convert function key labels.
   - On the field definition screen, if the processing specs have converted data which you want to save, press the FIELD TOGGLE key, then [Enter] to save that conversion.

**NOTE**

If you try to redisplay a screen which has already been converted and this conversion has been saved by pressing [Enter], a message `Form contains 8 bit data` will be displayed. Do not press [Enter] again, but continue on through the forms file.

5. Compile your forms file as usual.

**NOTE**

These conversion utilities are designed to be used once to update existing data to 8-bit compatibility.

# APPLICATION GUIDELINES G

Currently, the HP 3000 supports six conventional programming languages (SPL, FORTRAN, COBOLII, Pascal, RPG, and BASIC). Some general guidelines, and some specific to each of the supported programming languages, are included in this Appendix to help the programmer select a language to use for writing a local language or localizable application.

## All Programming Languages

- Create and use message catalogs. Do not hard-code any text messages, including prompts. For example, never require a hard-coded "Y" or"N" in response to a question. The equivalents of "yes" and "no" for every language supported by NLS are available through a call to NLINFO item 8.
- Use the NLS date and time formatting intrinsics. Do not use the MPE intrinsics DATELINE, FMTCLOCK, FMTDATE, and FMTCALENDAR. They all result in American-style output.
- Check a character's attribute, available through NLINFO item 12, to determine printability. Alternatively, use the NLREPCHAR intrinsic to check whether the character gets replaced or not. Do not use range checking on the binary value of a character to decide whether it is printable or not.
- Use the NLCOLLATE intrinsic to compare character strings. Do not compare character strings (IF abc > pqr ..., where abc and pqr are both character strings). Since these comparisons are based on binary values of characters as they appear in the USASCII sequence, they usually produce incorrect results. Obviously, this is not applicable in case an exact match is tested (IF abc = pqr ...).
- Use NLSCANMOVE for upshifting and downshifting. Do not upshift or downshift based on the character's binary value. For a...z in USASCII, upshifting can be done by subtracting 32 from the binary value. This does not work for all characters in all character sets.
- To determine whether a character is uppercase or lowercase, use the character attributes table available through NLINFO item 12. Do not use a character's binary value in range checks to decide whether it is an uppercase or lowercase alphabetic character.
- Much Hewlett-Packard and user-written software assumes that numeric characters (0 through 9) are represented by code values 48 through 57 (decimal). In general, this is valid because standard Hewlett-Packard 8-bit character sets are supersets of USASCII. However, some character sets may have different or additional characters which should be treated as numeric. Therefore, if at all possible, avoid doing range checks on code values to recognize or process numeric characters. For recognition of numeric characters, interrogate the character attributes table, available through a call to NLINFO item 12.
- Use the NLTRANSLATE intrinsic, not CTRANSLATE, to translate to or from EBCDIC.

- Do your own formatting using the decimal separator, the thousands separator, and the currency symbol available through NLINFO items 9 and 10. Use the standard statements to output into a character string type variable. Replace the decimal and thousands separators by those required in the language being used. Do not use standard output statements (PRINT, WRITE) for real numbers, since this formats them according to the definition of the programming language. This usually results in American formats with a period used as the decimal separator.

- Input data into a character string, and preprocess the string to replace any decimal or thousands separators used in the American formats. Then supply the string to the standard READ statement. Standard input statements for real numbers (READ, ACCEPT) should not be used, as they accept the period as the decimal separator. Many non-American users will input something else (a comma, for example).

- Always store standard formats for date and time (like those returned by FMTCALENDAR and FMTCLOCK), if dates or times have to be stored in files or databases. Never store a date or a time in a local format. Intrinsics are available to convert from the standard format to a local format, but the reverse is not always possible.

- Use VPLUS/3000 local edits. VPLUS/3000 edit processing specifications and terminal edit processing statements are separate and are not checked for compatibility. There will be no check that the designer has specified a terminal local edit which is consistent with the language-dependent symbol for the decimal point (DEC TYPE EUR, DEC TYPE US) in the configuration phase.

## COBOLII (HP 32233A)

- Use the character attributes table of the character set being used to determine whether a character is ALPHABETIC or NUMERIC. This table is available through a call to NLINFO item 12. Do not use the COBOLII ALPHABETIC and NUMERIC class tests to determine this (for example, if data-item IS ALPHABETIC).

- Do not use input-output translation by COBOLII from an EBCDIC character set by means of the ALPHABET-NAME clause and the CODE SET clause. Use the NLTRANSLATE intrinsic.

- Use the NLS date and time formatting intrinsics for display purposes. Do not use TIME-OF-DAY and CURRENT-DATE. These items are formatted in the conventional American way, and are unsuitable for use in many other countries.

- Use the COLLATING SEQUENCE IS *language-name* or the COLLATING SEQUENCE IS *language-ID* phrase in the enhanced SORT and MERGE statements to specify the language name or number whose collating sequence is to be used. Do not use the COLLATING SEQUENCE IS *alphabet-name* phrase for sorting and/or merging in COBOLII.

- In condition-name data descriptions (88-level items), avoid the THRU option in the VALUE clause (for example, 88 SELECTED-ITEMS VALUE "A" THRU "F").

## FORTRAN (HP 32102B)

- Format specifiers N and M will output in an American numerical format (with commas between thousands and a decimal point) or an American monetary format (like N, with a "$" added). Additional post-processing will be required.
- Outputting logicals will result in a "T" (for true) or an "F" (for false). Similarly, "T" and "F" are expected for logical input. A non-English speaking user may want to use another character.
- The intrinsic functions RNUM, DNUM and STR all assume an American format in the input and produce an American formatted output.
- The EXTIN' and INEXT' entry points of the compiler library assume American formats. Do not use them.

## SPL (HP 32100A)

- To determine whether or not the byte is alphabetic, numeric, or special, consult the character attribute table of the character set used. This table is available through NLINFO item 12. Do not use the IF xyz = (or <>) ALPHA (or NUMERIC or SPECIAL) construct to determine this.
- Do not use the MOVE ... WHILE construct or the MVBW machine instruction. It stops moving bytes based on the USASCII binary value of bytes, by which it determines whether the byte is alphabetic or numeric. Use the NLSCANMOVE intrinsic.

## RPG (HP 32104A)

The features of NLS are accessed primarily through intrinsic calls. Using MPE and subsystem intrinsics from RPG requires expertise. For this reason, the use of RPG as a vehicle to write localizable applications or to access native language structures is not recommended. Some RPG functions, such as date and numeric formatting, provide some control for national custom differences, but the choices are very limited and can only be made by recompiling.

## BASIC (HP 32101B)

The features of NLS are accessed primarily through intrinsic calls. Since most intrinsics are not callable from BASIC, the use of BASIC as a language to write localizable programs is not supported.

## Pascal (HP 32106A)

A type of CHAR indicates an 8-bit entity, and thus allows processing of 8-bit characters without problems.

# EXAMPLE PROGRAMS H

The example programs in this Appendix demonstrate calls to NLS-related intrinsics from several programming languages. They are not intended to be used as application programs.

## A. SORT in a COBOLII Program

This program shows how to sort an input file (formal designator INPTFILE) to an output file (formal designator OUTPFILE) using a COBOLII SORT verb.

Lines 3.5 and 4.1 show how to specify the language to determine the collating sequence.

```
1      $CONTROL USLINIT
1.1     IDENTIFICATION DIVISION.
1.2     PROGRAM-ID.    EXAMPLE.
1.3    * ---------------------------------------------
1.4     ENVIRONMENT DIVISION.
1.5     INPUT-OUTPUT SECTION.
1.6     FILE-CONTROL.
1.7     SELECT INPTFILE ASSIGN TO "INPTFILE".
1.8     SELECT OUTPFILE ASSIGN TO "OUTPFILE".
1.9     SELECT SORTFILE ASSIGN TO "SORTFILE".
2      * ---------------------------------------------
2.1     DATA DIVISION.
2.2     FILE SECTION.
2.3     SD   SORTFILE.
2.4     01   SORTFILE-RECORD.
2.5          05  SORTFILE-KEY    PIC X(4).
2.6          05  FILLER          PIC X(68).
2.7
2.8     FD   INPTFILE.
2.9     01   INPTFILE-RECORD     PIC X(72).
3
3.1     FD   OUTPFILE.
3.2     01   OUTPFILE-RECORD     PIC X(72).
3.3
3.4     WORKING-STORAGE SECTION.
3.5     01   LANGUAGE            PIC S9(4) COMP VALUE 12.
3.6    * ---------------------------------------------
3.7     PROCEDURE DIVISION.
3.8     MAIN SECTION.
3.9         SORT SORTFILE
4                ASCENDING SORTFILE-KEY
4.1              SEQUENCE IS LANGUAGE
4.2              USING  INPTFILE
4.3              GIVING OUTPFILE.
4.4         STOP RUN.
```

Line 3.5 could be written also as:

```
3.5     01    LANGUAGE           PIC X(16) VALUE "SPANISH ".
```

In the example execution the input and output files are associated with the terminal ($STDIN and $STDLIST):

```
:FILE INPTFILE=$STDIN
:FILE OUTPFILE=$STDLIST
:RUN PROGRAM;MAXDATA=12000

character
credit
DEBIT
:EOD

credit
character
DEBIT

END OF PROGRAM
:
```

## B. SORT in a Pascal Program

This program shows how to sort an input file (formal designator INPF) to an output file (formal designator OUTF) using the SORTINIT intrinsic call.

```
$USLINIT$
$STANDARD_LEVEL 'HP3000'$

PROGRAM example (inpf,outf);

TYPE
   smallint  = -32768 .. 32767;

   sort_rec  = RECORD
                 position:  smallint;
                 length:    smallint;
                 seq_type:  smallint;
               END;

   char_seq  = RECORD
                 array_code:smallint;
                 language:  smallint;
               END;

   file_arr  = RECORD
                 num_file:  smallint;
                 num_zero:  smallint;
               END;

   file_rec  = PACKED ARRAY [1..72] of CHAR;

   file_num  = FILE of file_rec;

VAR
   numkeys: smallint;
   reclen:  smallint;
   keys:    sort_rec;
   cseq:    char_seq;
   inp:     file_arr;
   out:     file_arr;
   inpf:    file_num;
   outf:    file_num;

PROCEDURE sortinit;   INTRINSIC;
PROCEDURE sortend;    INTRINSIC;

PROCEDURE main;
BEGIN
   numkeys := 1;
   reclen  :=72;

```

```
     WITH keys DO
     BEGIN
       position := 1;
       length   := 4;
       seq_type := 9;
     END;

     WITH cseq DO
     BEGIN
       array_code:=1;
       language:= 12;
     END;

     WITH inp  DO
     BEGIN
       RESET (inpf);
       num_file := FNUM (inpf);
       num_zero := 0;
     END;

     WITH out  DO
     BEGIN
       REWRITE (outf);
       num_file := FNUM (outf);
       num_zero := 0;
     END;

     sortinit (inp,out,,reclen,,numkeys,keys,,,,,,,,cseq);
     sortend;

  END;

  BEGIN
     main;
  END.
```

In the example execution, the input and output files are associated with the terminal ($STDIN and $STDLIST):

```
:FILE INPF=$STDIN
:FILE OUTF=$STDLIST
:RUN PROGRAM;MAXDATA=12000

character
credit
DEBIT
:EOD

credit
character
DEBIT

END OF PROGRAM
:
```

## C. SORT in a FORTRAN Program

This program shows how to sort an input file (formal designator FTN21) to an output file (formal designator FTN22) using the SORTINIT intrinsic call.

```
$CONTROL USLINIT,FILE=21-22
      PROGRAM EXMP
      INTEGER FNUM
      INTEGER N(4)
      INTEGER KEYS (3)
      INTEGER CSEQ (2)
      SYSTEM INTRINSIC SORTINIT, SORTEND
C
C      KEY (3) = 9  character type key
C      CSEQ(2) = 12 Spanish collating sequence
C
      KEYS (1) = 1
      KEYS (2) = 4
      KEYS (3) = 9
      CSEQ (1) = 1
      CSEQ (2) = 12
C
C     Sort file FTN21 into FTN22
C
      N (1) = FNUM (21)
      N (3) = FNUM (22)
      N (2) = 0
      N (4) = 0
      CALL SORTINIT (N(1),N(3),,,,1,KEYS,,,,,,,,CSEQ)
      CALL SORTEND
      STOP
      END
```

In the example execution, the input and output files are associated with the terminal ($STDIN and $STDLIST):

```
:FILE FTN21=$STDIN
:FILE FTN22=$STDLIST
:RUN PROGRAM;MAXDATA=12000

character
credit
DEBIT
:EOD

credit
character
DEBIT

END OF PROGRAM
:
```

## D. DATE/TIME Formatting Intrinsics in a FORTRAN Program

The user is asked to enter a language. All date and time formatting and conversion is done by using the language entered by the user. The time and date used in the examples is the current system time obtained by calling the HP 3000 system intrinsics CALENDAR and CLOCK.

```
$CONTROL USLINIT
      PROGRAM EXAMPLE
      LOGICAL LANGUAGE(8)
      CHARACTER *16 BLANGUAGE
C
      LOGICAL LERROR(2)
      INTEGER IERROR(2)
C
      CHARACTER *13 BCUSTOMDATE
      CHARACTER *28 BDATE
      CHARACTER *18 BCALENDAR
      CHARACTER  *8 BCLOCK
C
      LOGICAL LWEEKDAYS(42)
      CHARACTER *12 BWEEKDAYS(7)
C
      LOGICAL LMONTHS(72)
      CHARACTER *12 BMONTHS(12)
C
      EQUIVALENCE (LANGUAGE, BLANGUAGE)
      EQUIVALENCE (LWEEKDAYS,BWEEKDAYS)
      EQUIVALENCE (LMONTHS,  BMONTHS)
      EQUIVALENCE (LERROR,   IERROR)
      LOGICAL DATE
      INTEGER *4 TIME
      INTEGER LANGNUM, LGTH, WEEKDAY, MONTH
      SYSTEM INTRINSIC CLOCK, CALENDAR, ALMANAC, NLINFO,
     #       NLFMTCLOCK, QUIT, NLCONVCLOCK, NLFMTDATE,
     #       NLFMTCALENDAR, NLFMTCUSTDATE, NLCONVCUSTDATE
C
1001  FORMAT (1X,A12)
1002  FORMAT (1X,A13)
1003  FORMAT (1X,A18)
1004  FORMAT (1X,A8)
1005  FORMAT (1X,A28)
2001  FORMAT (A16)
2002  FORMAT (A1)
C
1     WRITE (6,*)
     #"ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):"
      READ (5, 2001) BLANGUAGE
C
C     NLINFO item 22 returns the corresponding
C     lang number in integer format for this language.
C
      CALL NLINFO (22, LANGUAGE, LANGNUM, LERROR)
      IF (IERROR(1) .EQ. 0) GO TO 400
C
C
100   IF (IERROR(1) .NE. 1) GO TO 200
C
      WRITE (6, *) "NLS IS NOT INSTALLED"
      CALL QUIT (1001)
C
200   IF (IERROR(1) .NE. 2) GO TO 300
C
      WRITE (6, *) "THIS LANGUAGE IS NOT CONFIGURED"
      CALL QUIT (1002)
C
```

```
300   CALL QUIT (1000 + IERROR(1))
C
C     This obtains the machine internal clock and calendar
C     formats, which are provided by the HP 3000 intrinsics.
C
400   TIME = CLOCK
      DATE = CALENDAR
C
C     Call ALMANAC and convert the machine internal
C     date format into numeric values, which will be used
C     as indices into the name tables.
C
      CALL ALMANAC(DATE, LERROR, , MONTH, ,WEEKDAY)
      IF (IERROR(1) .NE. 0) CALL QUIT (2000 + IERROR(1))
C
C     Call the tables for month and weekday names and
C     display todays day name and the current month's name.
C
      CALL NLINFO(5, LMONTHS, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (3000 + IERROR(1))
C
      WRITE (6, 1001) BMONTHS (MONTH)
C
      CALL NLINFO(7, LWEEKDAYS, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (4000 + IERROR(1))
C
      WRITE (6, 1001) BWEEKDAYS (WEEKDAY)
C
C     Format the machine internal date format
C     into the custom date format (short version).
C     The result will be displayed.
C
      CALL NLFMTCUSTDATE (DATE, BCUSTOMDATE, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (5000 + IERROR(1))
C
      WRITE (6,*) "CUSTOM DATE:"
      WRITE (6,1002) BCUSTOMDATE
C
C     Use the output of NLFMTCUSTDATE as input for
C     NLCONVCUSTDATE and convert back to the internal format.
C
      DATE = NLCONVCUSTDATE(BCUSTOMDATE, 13, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (6000 + IERROR(1))
C
C     Format the machine internal date format into the
C     date format (long format) according to the language.
C     The result will be displayed.
C
      CALL NLFMTCALENDAR(DATE, BCALENDAR, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (7000 + IERROR(1))
C
      WRITE (6,*) "DATE FORMAT:"
      WRITE (6,1003) BCALENDAR
C
C     Format the machine internal time format into the
C     language-dependent clock format.
C     The result will be displayed.
C
      CALL NLFMTCLOCK(TIME, BCLOCK, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (8000 + IERROR(1))
```

```
C
      WRITE (6,*) "TIME FORMAT:"
      WRITE (6,1004) BCLOCK
C
C     Use the output of NLFMTCLOCK as input for
C     NLCONVCLOCK and convert back to the internal format.
C
      TIME = NLCONVCLOCK(BCLOCK, 8, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (9000 + IERROR(1))
C
C     Format the machine internal time and date format
C     into the language dependent format.
C     The result will be displayed.
C
      CALL NLFMTDATE(DATE, TIME, BDATE, LANGNUM, LERROR)
      IF (IERROR(1) .NE. 0) CALL QUIT (10000 + IERROR(1))
C
      WRITE (6,*) "DATE AND TIME FORMAT:"
      WRITE (6, 1005) BDATE
C
C
      STOP
      END
```

Executing the program gives the following result:

```
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
NATIVE-3000
JANUARY
TUESDAY
CUSTOM DATE:
01/31/84
DATE FORMAT:
TUE, JAN 31, 1984
TIME FORMAT:
5:15 PM
DATE AND TIME FORMAT:
TUE, JAN 31, 1984,  5:15 PM

END OF PROGRAM
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
 8
Januar
Dienstag
CUSTOM DATE:
31.01.84
DATE FORMAT:
Di., 31. Jan. 1984
TIME FORMAT:
17:15
DATE AND TIME FORMAT:
Di., 31. Jan. 1984, 17:15

END OF PROGRAM
:
```

## E. DATE/TIME Formatting Intrinsics in an SPL Program

The user is asked to enter a language. All date and time formatting and conversion is done by using the language entered by the user. The time and date used in the examples is the current system time obtained by calling the HP 3000 system intrinsics CALENDAR and CLOCK.

```
$CONTROL USLINIT
BEGIN
   LOGICAL ARRAY
      L'ERROR       (0:1),
      L'LANGUAGE    (0:7),
      L'PRINT       (0:39),
      L'CUSTOM'DATE (0:6),
      L'DATE        (0:13),
      L'CALENDAR    (0:8),
      L'MONTHS      (0:71),
      L'WEEKDAYS    (0:41),
      L'CLOCK       (0:3);

   BYTE ARRAY
      B'PRINT(*)       = L'PRINT,
      B'CUSTOM'DATE(*) = L'CUSTOM'DATE,
      B'CALENDAR(*)    = L'CALENDAR,
      B'DATE(*)        = L'DATE,
      B'MONTHS(*)      = L'MONTHS,
      B'WEEKDAYS(*)    = L'WEEKDAYS,
      B'CLOCK(*)       = L'CLOCK;

   BYTE POINTER
      BP'PRINT;

   DOUBLE
      TIME;

   LOGICAL
      DATE,
      HOUR'MINUTE = TIME,
      SECONDS     = TIME + 1;

   INTEGER
      YEAR,
      MONTH,
      DAY,
      WEEKDAY,
      LGTH,
      LANGNUM;

   DEFINE
      WEEKDAY'NAME = B'WEEKDAYS((WEEKDAY - 1) * 12)#,

      MONTH'NAME   = B'MONTHS((MONTH - 1) * 12)#,

      ERR'CHECK    = IF L'ERROR(0) <> 0 THEN
                        QUIT #,

      CCNE         = IF <> THEN
                        QUIT #,

      DISPLAY      = MOVE B'PRINT := #,

      ON'STDLIST   = ,2;
                    @BP'PRINT := TOS;
                     LGTH := LOGICAL(@BP'PRINT) -
                             LOGICAL(@B'PRINT);
                     PRINT(L'PRINT, -LGTH, 0) #;
```

```

    INTRINSIC
       READ,
       QUIT,
       PRINT,
       CLOCK,
       CALENDAR,
       ALMANAC,
       NLINFO,
       NLFMTCLOCK,
       NLCONVCLOCK,
       NLFMTDATE,
       NLFMTCALENDAR,
       NLFMTCUSTDATE,
       NLCONVCUSTDATE;


<< Start of main code.
   The user is asked to enter a language name or number.>>

   DISPLAY
   "ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):"
   ON'STDLIST;

   READ(L'LANGUAGE,-16);

<< NLINFO item 22 returns the corresponding
   lang number in integer format for this language.     >>

   NLINFO(22,L'LANGUAGE,LANGNUM,L'ERROR);
   IF L'ERROR(0) <> 0 THEN
      BEGIN
         IF L'ERROR(0) = 1 THEN
            BEGIN
               DISPLAY
               "NL/3000 IS NOT INSTALLED"
               ON'STDLIST;
               QUIT(1001);
            END
         ELSE
            IF L'ERROR(0) = 2 THEN
               BEGIN
                  DISPLAY
                  "THIS LANGUAGE IS NOT CONFIGURED"
                  ON'STDLIST;
                  QUIT(1002);
               END
            ELSE
               QUIT (1000 + L'ERROR(0));
      END;

<< This obtains the machine internal clock and
   calendar formats which is maintained by MPE.          >>

   TIME := CLOCK;

   DATE := CALENDAR;

<< Call ALMANAC and convert the machine internal date
   format into numeric values, which will be used as indices
   into the name tables.                                  >>

   ALMANAC(DATE, L'ERROR, , MONTH, , WEEKDAY);
   ERR'CHECK (2000 + L'ERROR(0));

<< Call the tables for month and weekday names and
   display todays day name and the current month's name.  >>

   NLINFO(5, L'MONTHS, LANGNUM, L'ERROR);
   ERR'CHECK (3000 + L'ERROR(0));

   DISPLAY MONTH'NAME,(12) ON'STDLIST;

   NLINFO(7, L'WEEKDAYS, LANGNUM, L'ERROR);
   ERR'CHECK (4000 + L'ERROR(0));

   DISPLAY WEEKDAY'NAME,(12) ON'STDLIST;

<< Format the machine internal date format
   into the custom date format (short version).
   The result will be displayed.                          >>

   NLFMTCUSTDATE(DATE,L'CUSTOM'DATE,LANGNUM,L'ERROR);
   ERR'CHECK (5000 + L'ERROR(0));

   DISPLAY "CUSTOM DATE:"     ON'STDLIST;
   DISPLAY B'CUSTOM'DATE,(13) ON'STDLIST;

<< Use the output of NLFMTCUSTDATE as input for
   NLCONVCUSTDATE and convert back to the internal format.>>

   DATE := NLCONVCUSTDATE(B'CUSTOM'DATE,13,LANGNUM,L'ERROR);
   ERR'CHECK (6000 + L'ERROR(0));

<< Format the machine internal date format into the       >>
<< date format (long format) according to the language.   >>
<< The result will be displayed.                          >>

   NLFMTCALENDAR(DATE,L'CALENDAR,LANGNUM,L'ERROR);
   ERR'CHECK (7000 + L'ERROR(0));

   DISPLAY "DATE FORMAT:"  ON'STDLIST;
   DISPLAY B'CALENDAR,(18) ON'STDLIST;

<< Format the machine internal clock format
   into the language-dependent clock format.
   The result will be displayed.                        >>

   NLFMTCLOCK(TIME,L'CLOCK,LANGNUM,L'ERROR);
   ERR'CHECK (8000 + L'ERROR(0));

   DISPLAY "TIME FORMAT:" ON'STDLIST;
   DISPLAY B'CLOCK,(8)    ON'STDLIST;

<< Use the output of NLFMTCLOCK as input for
   NLCONVCLOCK and convert back to the internal format. >>

   TIME := NLCONVCLOCK(B'CLOCK,8,LANGNUM,L'ERROR);
   ERR'CHECK (9000 + L'ERROR(0));

<< Format the machine internal time and date
   format into the language-dependent format.
   The result will be displayed.                        >>

   NLFMTDATE(DATE,TIME,L'DATE,LANGNUM,L'ERROR);
   ERR'CHECK (10000 + L'ERROR(0));

   DISPLAY "DATE AND TIME FORMAT:" ON'STDLIST;
   DISPLAY B'DATE,(28)    ON'STDLIST;

END.
```

Executing the program results in the following:

```
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
GERMAN
Januar
Dienstag
CUSTOM DATE:
31.01.84
DATE FORMAT:
Di., 31. Jan. 1984
TIME FORMAT:
17:12
DATE AND TIME FORMAT:
Di., 31. Jan. 1984, 17:12

END OF PROGRAM

:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
 0
JANUARY
TUESDAY
CUSTOM DATE:
01/31/84
DATE FORMAT:
TUE, JAN 31, 1984
TIME FORMAT:
5:13 PM
DATE AND TIME FORMAT:
TUE, JAN 31, 1984,  5:13 PM

END OF PROGRAM
:
```

## F. NLSCANMOVE Intrinsic in a COBOLII Program

In this program there are six different calls to NLSCANMOVE. In every call all parameters are passed to NLSCANMOVE. Since the upshift/downshift table and the character attributes table are optional parameters, they may be omitted. For performance reasons (if NLSCANMOVE is called frequently), they should be passed to the intrinsic after being read in by the appropriate calls to NLINFO.

```
$CONTROL USLINIT
 IDENTIFICATION DIVISION.
     PROGRAM-ID. EXAMPLE.
     AUTHOR. LORO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    77      QUITPARM            PIC S9(4) COMP VALUE 0.
    77      LANGNUM             PIC S9(4) COMP VALUE 0.
    77      FLAGS               PIC S9(4) COMP VALUE 0.
    77      LEN                 PIC S9(4) COMP VALUE 70.
    77      NUMCHAR             PIC S9(4) COMP VALUE 0.

    01      TABLES.
      05    CHARSET-TABLE       PIC X(256) VALUE SPACES.
      05    UPSHIFT-TABLE       PIC X(256) VALUE SPACES.
      05    DOWNSHIFT-TABLE     PIC X(256) VALUE SPACES.

    01      STRINGS.
      05    INSTRING.
        10  INSTR1              PIC X(40) VALUE SPACES.
        10  INSTR2              PIC X(30) VALUE SPACES.
      05    OUTSTRING           PIC X(70) VALUE SPACES.
      05    LANGUAGE            PIC X(16) VALUE SPACES.

    01      ERRORS.
      05    ERR1                PIC S9(4) COMP.
      88    NO-NLS                        VALUE 1.
      88    NOT-CONFIG                    VALUE 2.
      05    ERR2                PIC S9(4) COMP VALUE 0.

 PROCEDURE DIVISION.
 START-PGM.
*  Initializing the arrays.

     MOVE "abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7"
     TO INSTR1.
     MOVE "a  123&i12fSXgVhklKLabCDASPO6i"
     TO INSTR2.

*  The user is asked to enter a language name or

     DISPLAY
     "ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):".
     ACCEPT LANGUAGE.

 CONVERT-NAME-NUM.
*  NLINFO item 22 returns the corresponding
*  lang number in integer format for this language.

```

```
           CALL INTRINSIC "NLINFO" USING 22,
                                        LANGUAGE,
                                        LANGNUM,
                                        ERRORS.
           IF ERR1 NOT EQUAL 0
              IF NO-NLS
                 DISPLAY "NL/3000 IS NOT INSTALLED"
                 CALL INTRINSIC "QUIT" USING 1001
              ELSE
                 IF NOT-CONFIG
                    DISPLAY "THIS LANGUAGE IS NOT CONFIGURED"
                    CALL INTRINSIC "QUIT" USING 1002
                 ELSE
                    COMPUTE QUITPARM = 1000 + ERR1
                    CALL INTRINSIC "QUIT" USING QUITPARM.

       GET-TABLES.
      *  Obtain the character attributes table
      *  using NLINFO item 12.

           CALL INTRINSIC "NLINFO" USING 12,
                                        CHARSET-TABLE,
                                        LANGNUM,
                                        ERRORS.
           IF ERR1 NOT EQUAL 0
              COMPUTE QUITPARM = 2000 + ERR1
              CALL INTRINSIC "QUIT" USING QUITPARM.

      *  Obtain the upshift table using NLINFO item 15.

           CALL INTRINSIC "NLINFO" USING 15,
                                        UPSHIFT-TABLE,
                                        LANGNUM,
                                        ERRORS.
           IF ERR1 NOT EQUAL 0
              COMPUTE QUITPARM = 3000 + ERR1
              CALL INTRINSIC "QUIT" USING QUITPARM.

      *  Obtain the downshift table using NLINFO item 16.

           CALL INTRINSIC "NLINFO" USING 16
                                        DOWNSHIFT-TABLE,
                                        LANGNUM,
                                        ERRORS.
           IF ERR1 NOT EQUAL 0
              COMPUTE QUITPARM = 4000 + ERR1
              CALL INTRINSIC "QUIT" USING QUITPARM.

           DISPLAY "THE FOLLOWING STRING IS USED IN ALL EXAMPLES:"
           DISPLAY INSTRING.

       EXAMPLE-1-1.
      *  The string passed in the array instring should be moved
      *  and upshifted simultaneously to the array outstring.
      *  Set the until flag (bit 11 = 1) and the
      *  upshift flag (bit 10 = 1).  All other flags remain
      *
      *     0 1 2 3 4 5 6 7 8 9
      *     0 0 0 0 0 0 0 0 0 0
      *
```

```
*  Note: The 'until flag' is set.  Therefore, the operation continues
*        until one of the ending criteria will be true.
*        If no ending condition is set, the operation
*        continues for the number of characters contained in
*        length.
     MOVE 48    TO FLAGS.

     CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                       OUTSTRING,
                                       FLAGS,
                                       LEN,
                                       LANGNUM,
                                       ERRORS,
                                       CHARSET-TABLE,
                                       UPSHIFT-TABLE
                              GIVING NUMCHAR.
      IF ERR1 NOT EQUAL 0
         COMPUTE QUITPARM = 5000 + ERR1
         CALL INTRINSIC "QUIT" USING QUITPARM.

      DISPLAY "UPSHIFTED:  (EXAMPLE 1-1)".
      DISPLAY OUTSTRING.

 EXAMPLE-1-2.
*
*  The string passed in the array instring should be moved
*  and upshifted to the array outstring (same as EXAMPLE 1-1).
*  Set the while flag (bit 11 = 0) and the
*  (bit 10 = 1).  In addition all ending conditions will
*  set (bits 12 - 15 all 1).
*
*    0 1 2 3 4 5 6 7 8 9
*    0 0 0 0 0 0 0 0 0 0
*
*  Note: The 'while flag' is set.  Therefore, the operation
*        continues while one of the end criteria is true.
*        Since all criteria are set, one of them will be
*        always true, and the operation continues for the
*        number of characters contained in length.

     MOVE SPACES  TO OUTSTRING.
     MOVE 0       TO FLAGS.
     MOVE 47      TO FLAGS.

     CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                       OUTSTRING,
                                       FLAGS,
                                       LEN,
                                       LANGNUM,
                                       ERRORS,
                                       CHARSET-TABLE,
                                       UPSHIFT-TABLE
                              GIVING NUMCHAR.

      IF ERR1 NOT EQUAL 0
         CALL INTRINSIC "QUIT" USING 6.

      DISPLAY "UPSHIFTED:  (EXAMPLE 1-2)".
      DISPLAY OUTSTRING.

```

```
EXAMPLE-2-1.
* The string passed in the array instring should be
* scanned for the first occurrence of a special character.
* All characters before the first special character are
* moved to outstring.
* Set the until flag (bit 11 = 1) and the
* character flag (bit 12 = 1). All other flags remain
*
*   0 1 2 3 4 5 6 7 8 9
*   0 0 0 0 0 0 0 0 0 0
*
* Note: The 'until flag' is set and the ending condition
*       set to 'special character'.  Therefore, the operation
*       continues until the first special character is found
*       or until the number of characters contained in
*       length is processed.

    MOVE SPACES  TO OUTSTRING.

    MOVE 24      TO FLAGS.

    CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                      OUTSTRING,
                                      FLAGS,
                                      LEN,
                                      LANGNUM,
                                      ERRORS,
                                      CHARSET-TABLE,
                                      UPSHIFT-TABLE
                             GIVING NUMCHAR.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITPARM = 7000 + ERR1
        CALL INTRINSIC "QUIT" USING QUITPARM.

     DISPLAY "SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)".
     DISPLAY OUTSTRING.

EXAMPLE-2-2.
* The string passed in the array instring should
* be scanned for the first occurrence of a special
* character.  All characters before the first special
* character are moved to outstring (same as EXAMPLE 2-1).
* Set the while flag (bit 11 = 0) and all
* flags except for special characters (bits 13 - 15 =
*
*   0 1 2 3 4 5 6 7 8 9
*   0 0 0 0 0 0 0 0 0 0
*
* Note: The 'while flag' is set and all ending criteria
*       except for special characters are set.  Therefore, the
*       operation continues while an uppercase, a lowercase, or
*       a numeric character is found.  When a special
*       character is found, or the number of characters
*       contained in length is processed, the operation will
*       terminate.

    MOVE SPACES  TO OUTSTRING.

    MOVE 7       TO FLAGS.

```

```
     CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                       OUTSTRING,
                                       FLAGS,
                                       LEN,
                                       LANGNUM,
                                       ERRORS,
                                       CHARSET-TABLE,
                                       UPSHIFT-TABLE
                              GIVING NUMCHAR.

     IF ERR1 NOT EQUAL 0
        COMPUTE QUITPARM = 8000 + ERR1
        CALL INTRINSIC "QUIT" USING QUITPARM.

     DISPLAY "SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)".
     DISPLAY OUTSTRING.

 EXAMPLE-3-1.
*  The string passed in the array instring should be
*  scanned for the first occurrence of a special or numeric
*  character.  All characters before one of these characters
*  are moved to outstring and downshifted simultaneously.
*  Set the until flag (bit 11 = 1) and the
*  flags for special and numeric characters (bits 12-13 = 1).
*  To perform downshifting set bit 9 to 1.
*
*    0 1 2 3 4 5 6 7 8 9
*    0 0 0 0 0 0 0 0 0 1
*
*  Note: The 'until flag' is set and the ending condition
*        set to 'special character' and to 'numeric character'.
*        Therefore, the operation continues until the first
*        special or numeric character is found, or
*        until the number of characters contained in length
*        is processed.
*

     MOVE SPACES  TO OUTSTRING.

     MOVE 92      TO FLAGS.

     CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                       OUTSTRING,
                                       FLAGS,
                                       LEN,
                                       LANGNUM,
                                       ERRORS,
                                       CHARSET-TABLE,
                                       DOWNSHIFT-TABLE
                              GIVING NUMCHAR.

     IF ERR1 NOT EQUAL TO 0
        COMPUTE QUITPARM = 9000 + ERR1
        CALL INTRINSIC "QUIT" USING QUITPARM.

     DISPLAY
     "SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)".
     DISPLAY OUTSTRING.

 EXAMPLE-3-2.
```

```
*  The string passed in the array instring should be
*  scanned for the first occurrence of a special or numeric
*  character.  All characters before one of these characters
*  are moved to outstring and downshifted simultaneously
*  (same as EXAMPLE-3-2).
*  Set the while flag (bit 11 = 0) and the
*  flags for upper and lower case characters (bits 14-15 =
*  To perform downshifting set bit 9 to 1.
*
*    0 1 2 3 4 5 6 7 8 9
*    0 0 0 0 0 0 0 0 0 1
*
*  Note: The 'while flag' is set and the ending criteria
*        upppercase and lowercase characters are set.
*        Therefore, the operation continues while an uppercase or
*        a lowercase character is found.  When a special
*        or a numeric character is found, or the number of
*        characters contained in length is processed, the
*        operation will terminate.

     MOVE SPACES  TO OUTSTRING.

     MOVE 67      TO FLAGS.

     CALL INTRINSIC "NLSCANMOVE" USING INSTRING,
                                        OUTSTRING,
                                        FLAGS,
                                        LEN,
                                        LANGNUM,
                                        ERRORS,
                                        CHARSET-TABLE,
                                        DOWNSHIFT-TABLE
                              GIVING NUMCHAR.

     IF ERR1 NOT EQUAL 0
        COMPUTE QUITPARM = 10000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITPARM.

     DISPLAY
     "SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)".
     DISPLAY OUTSTRING.

     STOP RUN.
```

Executing the program results in the following:

```
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
GERMAN
THE FOLLOWING STRING IS USED IN ALL EXAMPLES:
abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7a  123&i12fSXgVhklKLabCDASPO6i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBRCGJGF1F$E!SP[6DXE\1A23%&7A  123&I12FSXGRHKLKLABCDASP[6I
UPSHIFTED:  (EXAMPLE 1-2)
ABCDFG6IJKASXBRCGJGF1F$E!SP[6DXE\1A23%&7A  123&I12FSXGRHKLKLABCDASP[6I
SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)
abCDfg6ijkaSXbVcGjGf1f
SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)
abCDfg6ijkaSXbVcGjGf1f
SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)
abcdfg
SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)
abcdfg

END OF PROGRAM

:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
0
THE FOLLOWING STRING IS USED IN ALL EXAMPLES:
abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7a  123&i12fSXgVhklKLabCDASPO6i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBVCGJGF1F$E!SPO6DLE\1A23%&7A  123&I12FSXGVHKLKLABCDASPO6I
UPSHIFTED:  (EXAMPLE 1-2)
ABCDFG6IJKASXBVCGJGF1F$E!SPO6DLE\1A23%&7A  123&I12FSXGVHKLKLABCDASPO6I
SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)
abCDfg6ijka
SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)
abCDfg6ijka
SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)
abcdfg
SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)
abcdfg

END OF PROGRAM
:
```

## G. NLSCANMOVE Intrinsic in an SPL Program

In this program there are six different calls to NLSCANMOVE. In every call, parameters are passed to NLSCANMOVE. Since the upshift/downshift table and the character attributes table are optional parameters, they may be omitted. For performance reasons (if NLSCANMOVE is called frequently), they should be passed to the intrinsic after being read in by the appropriate calls to NLINFO.

```
$CONTROL USLINIT
BEGIN
   LOGICAL ARRAY
      L'UPSHIFT    (0:127),
      L'DOWNSHIFT  (0:127),
      L'CHARSET    (0:127),
      L'ERROR      (0:1),
      L'INSTRING   (0:34),
      L'OUTSTRING  (0:34),
      L'PRINT      (0:34),
      L'LANGUAGE   (0:7);

   BYTE ARRAY
      B'INSTRING(*)    = L'INSTRING,
      B'OUTSTRING(*)   = L'OUTSTRING,
      B'PRINT(*)       = L'PRINT;

   BYTE POINTER
      BP'PRINT;

   INTEGER
      LANGNUM,
      NUM'CHAR,
      LGTH,
      LENGTH;

   LOGICAL
      FLAGS;

   DEFINE
      LOWER'CASE      = FLAGS.(15:1)#,
      UPPER'CASE      = FLAGS.(14:1)#,
      NUMERIC'CHAR    = FLAGS.(13:1)#,
      SPECIAL'CHAR    = FLAGS.(12:1)#,

      WHILE'UNTIL     = FLAGS.(11:1)#,

      UPSHIFT'FLAG    = FLAGS.(10:1)#,
      DOWNSHIFT'FLAG = FLAGS.(9:1)#,

      ERROR'CHECK   = IF L'ERROR(0) <> 0 THEN
                         QUIT #,

      CCNE          = IF <> THEN
                         QUIT #,

      DISPLAY       = MOVE B'PRINT := #,

      ON'STDLIST    = ,2;
                      @BP'PRINT := TOS;
                      LGTH := LOGICAL(@BP'PRINT) -
                              LOGICAL(@B'PRINT);
                      PRINT(L'PRINT, -LGTH, 0) #;
   INTRINSIC
      READ,
      QUIT,
      PRINT,
      NLINFO,
      NLSCANMOVE;


<< Start of main code.
   Initializing the arrays.                                  >>

   MOVE B'INSTRING
            := "abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7",2;
   MOVE  *  := "a  123&i12fSXgVhklKLabCDASPO6i";

   MOVE L'OUTSTRING       := "  ";
   MOVE L'OUTSTRING(1)    := L'OUTSTRING,(39);

   MOVE L'LANGUAGE        := "  ";
   MOVE L'LANGUAGE(1)     := L'LANGUAGE,(7);

<< The user is asked to enter a language name or

   DISPLAY
      "ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):"
   ON'STDLIST;

   READ(L'LANGUAGE,-16);

<< NLINFO item 22 returns the corresponding language
   number in integer format for this language.               >>

   NLINFO(22,L'LANGUAGE,LANGNUM,L'ERROR);
   IF L'ERROR(0) <> 0 THEN
      BEGIN
         IF L'ERROR(0) = 1 THEN
            BEGIN
               DISPLAY
               "NL/3000 IS NOT INSTALLED"
               ON'STDLIST;
               QUIT (1001);
            END
         ELSE
            IF L'ERROR(0) = 2 THEN
               BEGIN
                  DISPLAY
                  "THIS LANGUAGE IS NOT CONFIGURED"
                  ON'STDLIST;
                  QUIT (1002);
               END
            ELSE
               QUIT (1000 + L'ERROR(0));
      END;


<< Obtain the character attributes table using
   NLINFO item 12.                                            >>

   NLINFO(12,L'CHARSET,LANGNUM,L'ERROR);
   ERROR'CHECK (2000 + L'ERROR(0));

<< Obtain the upshift table using NLINFO item 15.              >>

   NLINFO(15,L'UPSHIFT,LANGNUM,L'ERROR);
   ERROR'CHECK (3000 + L'ERROR(0));

<< Obtain the downshift table using NLINFO item 16.            >>

   NLINFO(16,L'DOWNSHIFT,LANGNUM,L'ERROR);
   ERROR'CHECK (4000 + L'ERROR(0));

<< Print the character string used in all examples(instring).  >>

   DISPLAY
      "THE FOLLOWING STRING IS USED IN ALL EXAMPLES:"
   ON'STDLIST;
   DISPLAY B'INSTRING,(70) ON'STDLIST;

EXAMPLE'1'1:
<< The string passed in the array instring is moved and
   UPSHIFTED to the array outstring.
   Note: The 'until flag' is set.  Therefore, the operation
         continues until one of the ending criteria is true.
         If no ending condition was set the
         operation continues for the number of characters
         contained in length.                                  >>

   LENGTH         := 70;

   FLAGS          := 0;

   WHILE'UNTIL    := 1;
   UPSHIFT'FLAG   := 1;

   NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
             LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'UPSHIFT);
   ERROR'CHECK (5000 + L'ERROR(0));

   DISPLAY "UPSHIFTED:  (EXAMPLE 1-1)" ON'STDLIST;
   DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

EXAMPLE'1'2:
<< Note: The 'while flag' is set.  Therefore, the operation will
         continue while one of the end criteria is true.  Since
         all conditions are set, one of them will be always
         true and the operation continues for the number of
         characters contained in length.  This example performs
         the same operation as EXAMPLE 1-1.                    >>

   MOVE L'OUTSTRING      := " ";
   MOVE L'OUTSTRING(1)   := L'OUTSTRING,(39);

   FLAGS          := 0;

   LOWER'CASE     := 1;
   UPPER'CASE     := 1;
   SPECIAL'CHAR   := 1;
   NUMERIC'CHAR   := 1;

   WHILE'UNTIL    := 0;
   UPSHIFT'FLAG   := 1;

   NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
              LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'UPSHIFT);
   ERROR'CHECK (6000 + L'ERROR(0));

   DISPLAY "UPSHIFTED:  (EXAMPLE 1-2)" ON'STDLIST;
   DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

EXAMPLE'2'1:
<< The string contained in instring should be scanned for the
   first occurrence of a special character.  All characters
   before the first special are moved to outstring.
   Note: The 'until flag' is set and the ending condition is
         set to 'special character'.  Therefore, the operation
         continues until the first special character is found or
         until the number of characters contained in length
         is processed.                                        >>


   MOVE L'OUTSTRING      := "  ";
   MOVE L'OUTSTRING(1)   := L'OUTSTRING,(39);

   FLAGS          := 0;

   SPECIAL'CHAR   := 1;

   WHILE'UNTIL    := 1;
   UPSHIFT'FLAG   := 0;

   NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
              LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'UPSHIFT);
   ERROR'CHECK (7000 + L'ERROR (0));

   DISPLAY "SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)"
   ON'STDLIST;
   DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

EXAMPLE'2'2:
<< Note: The 'while flag' is set and all ending criteria
         except for special characters are set.  Therefore, the
         operation continues while an uppercase, a lowercase, or
         a numeric character is found.  When a special
         character is found or the number of characters
         contained in length is processed, the operation will
         terminate.
         This is the same operation as in EXAMPLE 2-1.       >>

   MOVE L'OUTSTRING      := "  ";
   MOVE L'OUTSTRING(1)   := L'OUTSTRING,(39);

   FLAGS          := 0;

   LOWER'CASE     := 1;
   UPPER'CASE     := 1;
   SPECIAL'CHAR   := 0;
   NUMERIC'CHAR   := 1;

   WHILE'UNTIL    := 0;
   UPSHIFT'FLAG   := 0;

   NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
              LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'UPSHIFT);
   ERROR'CHECK (8000 + L'ERROR(0));

   DISPLAY "SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)"
   ON'STDLIST;
   DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

EXAMPLE'3'1:
<< The data contained in instring should be scanned for the
   first occurrence of a numeric or a special character.
   All characters preceding the first special or numeric character
   are moved to outstring.
   Note: The 'until flag' is set and the ending conditions are
         set to 'special character' and to 'numeric character'.
         Therefore, the operation runs until the first
         special or numeric character is found, or
         until the number of characters contained in length
         is processed.                                         >>


   MOVE L'OUTSTRING      := " ";
   MOVE L'OUTSTRING(1)   := L'OUTSTRING,(39);

   FLAGS          := 0;

   SPECIAL'CHAR   := 1;
   NUMERIC'CHAR   := 1;

   WHILE'UNTIL    := 1;
   DOWNSHIFT'FLAG := 1;

   NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
            LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'DOWNSHIFT);
   ERROR'CHECK (9000 + L'ERROR(0));

   DISPLAY
   "SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)"
   ON'STDLIST;
   DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

EXAMPLE'3'2:
<< Note: The 'while flag' is set and the ending criteria
         upppercase and lowercase characters are set.
         Therefore, the operation continues while an uppercase or
         a lowercase character is found.  When a special
         or numeric character is found or the number of
         characters contained in length is processed, the
         operation will terminate.
         This is the same operation as in EXAMPLE 3-1.         >>

   MOVE L'OUTSTRING      := " ";
   MOVE L'OUTSTRING(1)   := L'OUTSTRING,(39);

   FLAGS          := 0;

   LOWER'CASE     := 1;
   UPPER'CASE     := 1;
```

```

    WHILE'UNTIL    := 0;
    DOWNSHIFT'FLAG := 1;

    NUM'CHAR := NLSCANMOVE(B'INSTRING, B'OUTSTRING, FLAGS,
             LENGTH, LANGNUM, L'ERROR, L'CHARSET, L'DOWNSHIFT);
    ERROR'CHECK (1000 + L'ERROR(0));

    DISPLAY
    "SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)"
    ON'STDLIST;
    DISPLAY B'OUTSTRING,(NUM'CHAR) ON'STDLIST;

END.
```

Executing the program results in the following:

```
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
GERMAN
THE FOLLOWING STRING IS USED IN ALL EXAMPLES:
abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7a  123&i12fSXgVhklKLabCDASPO6i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBRCGJGF1F$E!SP[6DXE\1A23%&7A  123&I12FSXGRHKLKLABCDASP[6I
UPSHIFTED:  (EXAMPLE 1-2)
ABCDFG6IJKASXBRCGJGF1F$E!SP[6DXE\1A23%&7A  123&I12FSXGRHKLKLABCDASP[6I
SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)
abCDfg6ijkaSXbVcGjGf1f
SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)
abCDfg6ijkaSXbVcGjGf1f
SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)
abcdfg
SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)
abcdfg

END OF PROGRAM
:RUN PROGRAM

ENTER A LANGUAGE NAME OR NUMBER (MAX. 16 CHARACTERS):
NATIVE-3000
THE FOLLOWING STRING IS USED IN ALL EXAMPLES:
abCDfg6ijkaSXbVcGjGf1f$E!SPO6dLe\1a23%&7a  123&i12fSXgVhklKLabCDASPO6i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBVCGJGF1F$E!SPO6DLE\1A23%&7A  123&I12FSXGVHKLKLABCDASPO6I
UPSHIFTED:  (EXAMPLE 1-2)
ABCDFG6IJKASXBVCGJGF1F$E!SPO6DLE\1A23%&7A  123&I12FSXGVHKLKLABCDASPO6I
SCAN/MOVE  UNTIL SPECIAL:  (EXAMPLE 2-1)
abCDfg6ijka
SCAN/MOVE WHILE ALPHA OR NUM:  (EXAMPLE 2-2)
abCDfg6ijka
SCAN/MOVE/DOWNSHIFT UNTIL NUM. OR SPEC.:  (EXAMPLE 3-1)
abcdfg
SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)
abcdfg

END OF PROGRAM
:
```

## H. NLTRANSLATE/NLREPCHAR Intrinsics in a COBOLII Program

The string used in the example is 256 bytes in length and contains all possible byte values from 0 to 255. This string is converted from USASCII to EBCDIC. Then the converted string is taken and translated back to USASCII. This is done according to the ASCII-to-EBCDIC and EBCDIC-to-ASCII translation tables corresponding to the entered language.

Afterwards this twice-translated string is displayed. All characters which are non-printable (control and undefined characters) in the character set supporting the given language are replaced by a period before the string is displayed by calling NLREPCHAR intrinsic.

```
$CONTROL USLINIT
 IDENTIFICATION DIVISION.
     PROGRAM-ID. EXAMPLE.
     AUTHOR. LORO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    77      QUITNUM                 PIC S9(4) COMP VALUE 0.
    77      LANGNUM                 PIC S9(4) COMP VALUE 0.
    77      IND                     PIC S9(4) COMP VALUE 0.

    01      TABLES.
      05    USASCII-EBC-TABLE         PIC X(256) VALUE SPACES.
      05    EBC-USASCII-TABLE         PIC X(256) VALUE SPACES.
      05    CHARSET-TABLE           PIC X(256) VALUE SPACES.

    01      BUFFER-FIELDS.
      05    INT-FIELD               PIC S9(4) COMP VALUE -1.
      05    BYTE-FIELD REDEFINES INT-FIELD.
        10  FILLER                  PIC X.
        10  CHAR                    PIC X.

    01      STRINGS.
      05    LANGUAGE                PIC X(16)  VALUE SPACES.
      05    IN-STRING.
        10  IN-BYTE                 PIC X OCCURS 256.
      05    OUT-STRING.
        10  OUT-STR1                PIC X(80).
        10  OUT-STR2                PIC X(80).
        10  OUT-STR3                PIC X(80).
        10  OUT-STR4                PIC X(16).

    01      REPLACE-WORD            PIC S9(4) COMP VALUE 0.
    01      REPLACE-BYTES REDEFINES REPLACE-WORD.
      05    REPLACEMENT-CHAR        PIC X.
      05    FILLER                  PIC X.

    01      ERRORS.
      05    ERR1                    PIC S9(4) COMP.
      05    ERR2                    PIC S9(4) COMP.
 PROCEDURE DIVISION.
 START-PGM.
* Initialize the instring array with all possible
* byte values starting from binary zero until 255.
     MOVE -1 TO INT-FIELD.
     PERFORM FILL-INSTRING VARYING IND FROM 1 BY 1
             UNTIL IND > 256.
     GO TO GET-LANGUAGE.

 FILL-INSTRING.
```

```
     ADD 1       TO INT-FIELD.
     MOVE CHAR   TO IN-BYTE(IND).

 GET-LANGUAGE.
*The language is hard-coded, set to 8 (GERMAN).

     MOVE 8      TO LANGNUM.

 GET-THE-TABLES.
* Call the USASCII-EBCDIC and EBCDIC-USASCII
* conversion tables and the character attribute table
* by using the appropriate NLINFO items.
* Note: NLTRANSLATE and NLREPCHAR may be called without
*       passing the tables (last parameter).  For performance
*       reasons the tables should be passed, if these
*       intrinsics are called very often.

     CALL INTRINSIC "NLINFO" USING 13,
                                   USASCII-EBC-TABLE,
                                   LANGNUM,
                                   ERRORS.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 1000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.

     CALL INTRINSIC NLINFO ITEM 14,
                                   EBC-USASCII-TABLE,
                                   LANGNUM,
                                   ERRORS.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 2000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.
     CALL INTRINSIC "NLINFO" USING 12,
                                   CHARSET-TABLE,
                                   LANGNUM,
                                   ERRORS.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 3000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.

 CONVERT-ASC-EBC.
* Convert IN-STRING from USASCII into EBCDIC by
* using NLTRANSLATE code 2. The converted string will
* be in OUT-STRING.

     CALL INTRINSIC "NLTRANSLATE" USING 2,
                                       IN-STRING,
                                       OUT-STRING,
                                       256,
                                       LANGNUM,
                                       ERRORS,
                                       USASCII-EBC-TABLE.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 4000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.

 CONVERT-EBC-ASC.
* Convert OUT-STRING back from EBCDIC to USASCII by
* using NLTRANSLATE code 1. The retranslated string will
* be in IN-STRING again.
```

```
     CALL INTRINSIC "NLTRANSLATE" USING 1,
                                        OUT-STRING,
                                        IN-STRING,
                                        256,
                                        LANGNUM,
                                        ERRORS,
                                        EBC-USASCII-TABLE.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 5000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.

 REPLACE-NON-PRINTABLES.
* Replace all non-printable characters
* in IN-STRING and display the string.

     MOVE "." TO REPLACEMENT-CHAR.
     CALL INTRINSIC "NLREPCHAR" USING IN-STRING,
                                      IN-STRING,
                                      256,
                                      REPLACE-WORD,
                                      LANGNUM,
                                      ERRORS.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 6000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.

     DISPLAY "IN-STRING:"
     DISPLAY IN-STRING.
     STOP RUN.
```

## I. NLKEYCOMPARE Intrinsic in a COBOLII Program

The example shows a new KSAM/3000 file built programmatically with a language attribute. This means the keys will be sorted according to the collating sequence of this language. After building the file, the program writes 15 hard-coded data records into it.

Perform a generic FFINDBYKEY with a partial key of *length1* containing "E". This positions the KSAM/3000 file pointer to the first record whose key starts with "E".

After locating this record, read all subsequent records in the file sequentially and call NLKEYCOMPARE to check whether the key found is what was requested. If the result returned by NLKEYCOMPARE is3, the program is done. There are no more records whose key starts with any kind of "E".

```
$CONTROL USLINIT
 IDENTIFICATION DIVISION.
     PROGRAM-ID. EXAMPLE.
     AUTHOR. LORO.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP3000.
 OBJECT-COMPUTER. HP3000.
 SPECIAL-NAMES.
     CONDITION-CODE IS CC.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    77      QUITNUM                  PIC S9(4) COMP VALUE 0.
    77      LANGNUM                  PIC S9(4) COMP VALUE 0.
    77      LEGTH                    PIC S9(4) COMP VALUE 0.
    77      FNUM                     PIC S9(4) COMP VALUE 0.
    77      RESULT                   PIC S9(4) COMP VALUE 0.
    77      FOPTIONS                 PIC S9(4) COMP.
    77      AOPTIONS                 PIC S9(4) COMP.
    77      IND                      PIC S9(4) COMP.

    01      TABLES.
      05    COLL-TABLE               PIC X(800).
      05    KSAM-PARAM.
        10  KEY-FILE                 PIC X(8) VALUE SPACES.
        10  KEY-FILE-SIZ             PIC S9(8) COMP.
        10  FILLER                   PIC X(8) VALUE SPACES.
        10  LANGUAGE-NUM             PIC S9(4) COMP.
        10  FILLER                   PIC X(8) VALUE SPACES.
        10  FLAGWORD                 PIC S9(4) COMP.
        10  NUM-OF-KEYS              PIC S9(4) COMP.
        10  KEY-DESCR                PIC S9(4) COMP.
        10  KEY-LOCATION             PIC S9(4) COMP.
        10  DUPL-BLOCK               PIC S9(4) COMP.
        10  FILLER                   PIC X(20).

    01      STRINGS.
      05    GEN-KEY                  PIC X(4).
      05    FILENAME                 PIC X(8) VALUE SPACES.

    01      ERRORS.
      05    ERR1                     PIC S9(4) COMP.
      05    ERR2                     PIC S9(4) COMP VALUE 0.

    01      DATA-RECS.
      05    DATA-REC1                PIC X(50).
      05    DATA-REC2                PIC X(50).
      05    DATA-REC3                PIC X(50).

    01      DATA-RECS-R REDEFINES DATA-RECS.
```

```
      05    DATA-RECORD                         OCCURS 15.
        10  FILLER                     PIC X(10).

    01      KSAM-RECORD.
      05    FILLER                     PIC X(3).
      05    RECORD-KEY                 PIC X(4).
      05    FILLER                     PIC X(3).

 PROCEDURE DIVISION.
 INIT-KSAM-RECORDS.
* Initialize the Data Record with the data which should be
* written to the KSAM file.

     MOVE "014ABBeZZZ011EZqrzyx001ABCDXXX007EdCDxyx012IzzAzzz"
     TO DATA-REC1.

     MOVE "003EaBCXXX008\\aaYZZ015ABDYZY005eLDFyxy002BBCdxxx"
     TO DATA-REC2.

     MOVE "004eABCYYY006EabcYYY009AAAAyzz010eaxfxyz013FGHIzqs"
     TO DATA-REC3.

*  Hard-code the language used in the example program
*  to 0 (NATIVE - 3000).

     MOVE 0           TO LANGNUM.

*  Build a new KSAM file with the data file name
*  KD000. The key file has the name KK000.

*  Set the values for KSAM parameter array.

     MOVE "KD000   " TO FILENAME.
     MOVE "KK000   " TO KEY-FILE.

     MOVE 1           TO NUM-OF-KEYS.
     MOVE LANGNUM     TO LANGUAGE-NUM.
     MOVE %20         TO FLAGWORD.
     MOVE 0           TO KEY-FILE-SIZ.
     MOVE %10004      TO KEY-DESCR.
     MOVE 4           TO KEY-LOCATION.
     MOVE %100024     TO DUPL-BLOCK.
     MOVE %4000       TO FOPTIONS.
     MOVE 5           TO AOPTIONS.

     CALL INTRINSIC "FOPEN" USING FILENAME,
                                  FOPTIONS,
                                  AOPTIONS,
                                  -10,
                                  \\,
                                  KSAM-PARAM
                           GIVING FNUM.
     IF CC NOT EQUAL 0
        CALL INTRINSIC "PRINTFILEINFO" USING FNUM,
        CALL INTRINSIC "QUIT" USING 1000.

*  Fill the hard-coded data into the KSAM file.

     PERFORM FILL-IN-DATA VARYING IND FROM 1 BY 1
                          UNTIL IND > 15.
     GO TO FIND-DATA.

 FILL-IN-DATA.
     CALL INTRINSIC "FWRITE" USING FNUM,
                                   DATA-RECORD(IND),
                                   -10,
                                   0.
      IF CC NOT EQUAL 0
         CALL INTRINSIC "PRINTFILEINFO" USING FNUM,
         CALL INTRINSIC "QUIT" USING 2000.

 FIND-DATA.
*  Perform a generic FFINDBYKEY with a
*  partial key of length 1 and value "E". The relational
*  operator will be 2 (greater or equal).
*  This FFINDBYKEY will position the KSAM pointer at the
*  first key starting with any kind of "E".

     MOVE "E" TO GEN-KEY.

     CALL INTRINSIC "FFINDBYKEY" USING FNUM,
                                       GEN-KEY,
                                       0,
                                       1,
                                       2.
     IF CC NOT EQUAL 0
        CALL INTRINSIC "PRINTFILEINFO" USING FNUM,
        CALL INTRINSIC "QUIT" USING 3000.

*  Read the subsequent entries and check whether an
*  exact match occurred by using NLKEYCOMPARE.
*  When NLKEYCOMPARE returns 3 as a result, there are no
*  more keys starting with any kind of "E".
*  If an exact match was found the record is printed.

     DISPLAY
     "THE FOLLOWING RECORDS MATCH GEN-KEY (E) EXACTLY:"
     MOVE 0     TO RESULT.
     PERFORM READ-DATA UNTIL RESULT EQUAL 3.
     GO TO TERMINATE-PGM.

 READ-DATA.
     CALL INTRINSIC "FREAD" USING FNUM,
                                  KSAM-RECORD,
                                  -10.
     IF CC NOT EQUAL 0
        CALL INTRINSIC "PRINTFILEINFO" USING FNUM,
        CALL INTRINSIC "QUIT" USING 4000.

     CALL INTRINSIC "NLKEYCOMPARE" USING  GEN-KEY,
                                          1,
                                          RECORD-KEY,
                                          4,
                                          RESULT,
                                          LANGNUM,
                                          ERRORS,
                                          COLL-TABLE.
     IF ERR1 NOT EQUAL 0
        COMPUTE QUITNUM = 5000 + ERR1,
        CALL INTRINSIC "QUIT" USING QUITNUM.
     IF RESULT = 0
        DISPLAY KSAM-RECORD.

 TERMINATE-PGM.
* Close the KSAM file and purge it.

     CALL INTRINSIC "FCLOSE" USING FNUM,
                                   4,
                                   0.

     STOP RUN.
```

Executing the program results in the following:

```
:RUN PROGRAM

THE FOLLOWING RECORDS MATCH GEN-KEY (E) EXACTLY:
011EZqrzyx
003EaBCXXX
007EdCDxyx

END OF PROGRAM
:
```

## J. NLKEYCOMPARE Intrinsic in an SPL Program

The example shows a new KSAM/3000 file built programmatically. This new KSAM/3000 file is built with a language attribute. This means the keys will be sorted according to the collating sequence of this language. After building the file, it is filled with 15 hard-coded data records.

Perform a generic FFINDBYKEY with a partial key of *length1* containing "E". This should position the KSAM/3000 file pointer to the very first record whose key starts with any kind of "E".

After locating this record read all subsequent records in the file sequentially and call NLKEYCOMPARE to check whether the key found is what was requested. If the result returned by NLKEYCOMPARE is3, there are no more records starting with any kind of "E".

```
$CONTROL USLINIT
BEGIN
   LOGICAL ARRAY
      L'ERROR      (0:1),
      L'KSAM'PARAM (0:79),
      L'PRINT      (0:39),
      L'RECORD     (0:4),
      COLL'TABLE   (0:399);

   BYTE ARRAY
      FILENAME     (0:7),
      GEN'KEY      (0:4),
      KEY          (0:4),
      B'KSAM'PARAM(*) = L'KSAM'PARAM,
      B'PRINT(*)      = L'PRINT,
      B'RECORD(*)     = L'RECORD;

   DOUBLE ARRAY
      D'KSAM'PARAM(*) = L'KSAM'PARAM;

   BYTE POINTER
      BP'PRINT;

   INTEGER
      I,
      LGTH,
      FNUM,
      RESULT,
      LANGNUM;

   LOGICAL
      FOPTIONS,
      AOPTIONS;

   LOGICAL ARRAY
      L'DATA(0:74) :=

              <<  |key |  >>
               "014BBeZZZ",
               "011EZqrzyx",
               "001ABCDXXX", << This is the data, which     >>
               "007EdCDxyx", << will be written to the KSAM >>
               "012IzzAzzz", << file.                       >>
               "015ABDYZY", << The key starts in column 4  >>
               "005eLDFyxy", << and is 4 characters long.   >>
               "002BBCdxxx",
               "003EaBCXXX",
               "008\\aaYZZ",
               "004eABCYYY",
               "006EabcYYY",
               "009Ayzz",
               "010eaxfxyz",
               "013FGHIzqs";

<< The following DEFINE statement defines the layout of the
   KSAM parameter array, which is necessary to build a KSAM
   file programmatically.                                   >>

   DEFINE
      KEY'FILE     = L'KSAM'PARAM#,
      KEY'FILE'SIZ = D'KSAM'PARAM(2)#,
      KEY'DEV      = L'KSAM'PARAM(6)#,
      LANGUAGE     = L'KSAM'PARAM(10)#,
      FLAGWORD     = L'KSAM'PARAM(15)#,
      NUM'OF'KEYS  = L'KSAM'PARAM(16)#,
      KEY'TYPE     = L'KSAM'PARAM(17).(0:4)#,
      KEY'LENGTH   = L'KSAM'PARAM(17).(4:12)#,
      KEY'LOCATION = L'KSAM'PARAM(18)#,
      DUP'FLAG     = L'KSAM'PARAM(19).(0:1)#,
      KEY'BLOCK    = L'KSAM'PARAM(19).(1:15)#,
      RANDOM'FLAG  = L'KSAM'PARAM(20).(8:1)#;

   DEFINE

      RECORD       = L'DATA (I * 5)#,

      ERROR'CHECK  = IF L'ERROR(0) <> 0 THEN
                        QUIT #,

      CCNE         = IF <> THEN
                        QUIT #,

      DISPLAY      = MOVE B'PRINT := #,

      ON'STDLIST   = ,2;
                     @BP'PRINT := TOS;
                     LGTH := LOGICAL(@BP'PRINT) -
                             LOGICAL(@B'PRINT);
                     PRINT(L'PRINT, -LGTH, 0) #;

   INTRINSIC
      FOPEN,
      FREAD,
      FWRITE,
      FCLOSE,
      FFINDBYKEY,
      FGETKEYINFO,
      PRINTFILEINFO,
      NLINFO,
      NLKEYCOMPARE,
      FCLOSE,
      PRINT,
      QUIT,
      READ;

<< Initializing the arrays.                               >>

   MOVE L'KSAM'PARAM     := " ";
   MOVE L'KSAM'PARAM(1)  := L'KSAM'PARAM(0),(79);

   MOVE GEN'KEY          := "    ";

   MOVE KEY              := "    ";

<< Hard-code the language used to 8 (GERMAN).             >>

   LANGNUM := 8;

<< Call in the collating sequence table.
   This is done by calling NLINFO ITEM 11.                >>

   NLINFO (11, COLL'TABLE, LANGNUM, L'ERROR);
   IF L'ERROR(0) THEN
      QUIT(1000 + L'ERROR(0));

<< Build a new KSAM file with the data file name
   KD008. The key file has the name KK008.                >>

<< Set the values for KSAM parameter array.               >>

   MOVE FILENAME := "KD008   ";     << KSAM data file     >>
   MOVE KEY'FILE := "KK008   ";     << KSAM key  file     >>

   NUM'OF'KEYS      := 1;           << Num of keys = 0    >>
   LANGUAGE         := LANGNUM;     << Set the language   >>
   FLAGWORD.(11:1)  := 1;           << Indicates that     >>
                                    << language is set    >>
   KEY'FILE'SIZ     := 200D;        << Max. 200 entries   >>
   KEY'TYPE         := 1;           << Byte key           >>
   KEY'LENGTH       := 4;           << 4 byte length      >>
   KEY'LOCATION     := 4;           << Key start at col.4 >>
   DUP'FLAG         := 1;           << Allow dupl. keys   >>
   KEY'BLOCK        := 10;          << Keys per block 10  >>

   FOPTIONS         := %4000;       << KSAM file          >>
   AOPTIONS         := %5;          << Update             >>

   FNUM := FOPEN(FILENAME,FOPTIONS,AOPTIONS,-10,,
                                   B'KSAM'PARAM);
   IF <> THEN
      BEGIN
         PRINTFILEINFO(FNUM);
         QUIT(2000);
      END;

<< Copy the hard-coded data into the KSAM file.           >>
   I := -1;
   WHILE (I := I + 1) < 15 DO
   BEGIN
      FWRITE(FNUM, RECORD, -10, %0);
      IF <> THEN
         BEGIN
            PRINTFILEINFO(FNUM);
            QUIT(3000);
         END;
   END;

<< Perform a generic FFINDBYKEY with a                    >>
<< partial key of length 1 and value "E".  The relational
<< operator will be 2 (greater or equal).                  >>
<< FFINDBYKEY will position the KSAM pointer at the        >>
<< first record starting with any kind of "E".             >>

   MOVE GEN'KEY := "E";

   FFINDBYKEY(FNUM, GEN'KEY, 0, 1, 2);
   IF <> THEN
      BEGIN
         PRINTFILEINFO(FNUM);
         QUIT(4000);
      END;

<< Read the subsequent entries and check by                >>
<< using NLKEYCOMPARE whether an exact match was found.    >>
<< When NLKEYCOMPARE returns a 3 as a result, the program
<< is beyond the range of valid keys.                      >>
<< If an exact match was found, the record is printed.

   RESULT := 0;
   DISPLAY
   "THE FOLLOWING RECORDS MATCH GEN-KEY (E) EXACTLY:"
   ON'STDLIST;
   WHILE RESULT <> 3 DO
   BEGIN
      FREAD(FNUM,L'RECORD,-10);
      IF <> THEN
         BEGIN
            PRINTFILEINFO(FNUM);
            QUIT(5000);
         END;

      MOVE KEY := B'RECORD(3),(4);
      NLKEYCOMPARE(GEN'KEY, 1, KEY, 4, RESULT, LANGNUM,
                                       L'ERROR, COLL'TABLE);
      ERROR'CHECK(9000 + L'ERROR(0));
      IF RESULT = 0 THEN           << exact hit >>
         BEGIN
            DISPLAY B'RECORD,(10) ON'STDLIST;
         END;
   END;

<< Close the KSAM file and purge it.                       >>

   FCLOSE(FNUM, 4, 0);

END.
```

Executing the program results in the following:

```
:RUN PROGRAM

THE FOLLOWING RECORDS MATCH GEN-KEY (E) EXACTLY:
003EaBCXXX
007EdCDxyx
011EZqrzyx

END OF PROGRAM
:
```

# K. Obtaining Language Information In A COBOLII Program

This program prints the User Interface, Data Manipulation, System Default, KSAM/3000 key sequence, VPLUS/3000 forms file, and IMAGE/3000 data base language numbers.

```
$CONTROL USLINIT
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    EXAMPLE.
* -------------------------------------------------
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP3000.
 OBJECT-COMPUTER. HP3000.
 SPECIAL-NAMES.
 CONDITION-CODE IS CCODE.
* -------------------------------------------------
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01   LANGUAGE             PIC S9(4) COMP.

 01   NLERROR.
      05 NLERR OCCURS 2    PIC S9(4) COMP.

 01   FILENUM              PIC S9(4) COMP.

 01   KSAMAREA.
      05 KSAMPARAM.
         10 FILLER         PIC X(20).
         10 KLANG          PIC S9(4) COMP.
         10 FILLER         PIC X(8).
         10 FLAGS          PIC S9(4) COMP VALUE 0.
         10 FILLER         PIC X(148).
      05 KSAMCONTROL       PIC X(256).

 01   COMAREA.
      05 COM-STAT          PIC S9(4) COMP VALUE 0.
      05 COM-LANG          PIC S9(4) COMP VALUE 0.
      05 COM-LENG          PIC S9(4) COMP VALUE 60.
      05 COM-FILL          PIC X(114) VALUE LOW-VALUE.

 01   RESULT.
      05 OPER              PIC X(10).
      05 LANG              PIC ZZZ9.
      05 FILLER            PIC X(6)  VALUE " Error".
      05 NERR              PIC ZZZ9.

 01   DBNAME.
      05 FILLER            PIC X(2)  VALUE "  ".
      05 FILENAME          PIC X(36).

 01   PASSWORD             PIC X(8).

 01   DBMODE               PIC S9(4) COMP VALUE 5.

 01   STAT.
      05 DBSTAT            PIC S9(4) COMP VALUE 0.
      05 FILLER            PIC X(18).

 01   DUMMY                PIC S9(4) COMP.
* ---------------------------------------------------
 PROCEDURE DIVISION.

 MAIN.
     PERFORM USER-LANG.
     PERFORM DATA-LANG.
     PERFORM SYST-LANG.
     PERFORM KSAM-LANG.
     PERFORM FORM-LANG.
     PERFORM BASE-LANG.
     STOP RUN.
* ...................................................
 USER-LANG.
     CALL INTRINSIC "NLGETLANG" USING 1 NLERROR
                                GIVING LANGUAGE.
     MOVE "USER lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     MOVE NLERR (1)    TO NERR.
     DISPLAY RESULT.
* ...................................................
 DATA-LANG.
     CALL INTRINSIC "NLGETLANG" USING 2 NLERROR
                                GIVING LANGUAGE.
     MOVE "DATA lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     MOVE NLERR (1)    TO NERR.
     DISPLAY RESULT.
* ...................................................
 SYST-LANG.
     CALL INTRINSIC "NLGETLANG" USING 3 NLERROR
                                GIVING LANGUAGE.
     MOVE "SYST lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     MOVE NLERR (1)    TO NERR.
     DISPLAY RESULT.
* ...................................................
 KSAM-LANG.
     DISPLAY "Enter KSAM file name:".
     ACCEPT FILENAME FREE.
     IF FILENAME NOT = SPACES PERFORM KSAM-OPEN.

 KSAM-OPEN.
     CALL INTRINSIC "FOPEN" USING FILENAME 1
                            GIVING FILENUM.
     IF CCODE = 0
        THEN PERFORM KSAM-INFO
        ELSE DISPLAY "Error in KSAM file OPEN".

 KSAM-INFO.
     CALL INTRINSIC "FGETKEYINFO" USING FILENUM
                  KSAMPARAM  KSAMCONTROL.
     CALL INTRINSIC "FCLOSE" USING FILENUM 0 0.
     IF FLAGS < 0 THEN ADD 32768 TO FLAGS.
     IF FLAGS - (FLAGS / 32) * 32 > 15
        THEN MOVE KLANG TO LANGUAGE
        ELSE MOVE ZERO  TO LANGUAGE.
     MOVE SPACES       TO RESULT.
     MOVE "KSAM lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     DISPLAY RESULT.
* ...................................................
 FORM-LANG.
     DISPLAY "Enter FORM file name:".
     ACCEPT FILENAME FREE.
     IF FILENAME NOT = SPACES PERFORM FORM-OPEN.

 FORM-OPEN.
     CALL "VOPENFORMF" USING COMAREA FILENAME.
     IF COM-STAT = 0
        THEN PERFORM FORM-INFO
        ELSE DISPLAY "FORMS file OPEN failed:" COM-STAT.

 FORM-INFO.
     CALL "VGETLANG" USING COMAREA LANGUAGE.
     CALL "VCLOSEFORMF" USING COMAREA.
     MOVE "FORM lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     DISPLAY RESULT.
* ...................................................
 BASE-LANG.
     DISPLAY "Enter DATA BASE name:".
     ACCEPT FILENAME FREE.
     IF FILENAME NOT = SPACES PERFORM BASE-OPEN.

 BASE-OPEN.
     DISPLAY "Enter PASSWORD:".
     ACCEPT PASSWORD FREE.
     CALL "DBOPEN" USING DBNAME PASSWORD DBMODE STAT.
     IF DBSTAT = 0
        THEN PERFORM BASE-INFO
        ELSE DISPLAY "Error in Data Base Open:" DBSTAT.

 BASE-INFO.
     MOVE 901 TO DBMODE.
     CALL "DBINFO" USING DBNAME DUMMY DBMODE STAT LANGUAGE.
     MOVE 1 TO DBMODE.
     CALL "DBCLOSE" USING DBNAME DUMMY DBMODE STAT.
     MOVE "BASE lang:" TO OPER.
     MOVE LANGUAGE     TO LANG.
     DISPLAY RESULT.
```

Executing the program results in the following:

```
:RUN PROGRAM;MAXDATA=12000

USER lang:   0 Error   2
DATA lang:   3 Error   0
SYST lang:   0 Error   0
Enter KSAM file name:
GERMANK
KSAM lang:   8
Enter FORM file name:
FRENCHFF
FORM lang:   7
Enter DATA BASE name:
SPBASE.TEST
Enter PASSWORD:
MANAGER
BASE lang:  12

END OF PROGRAM
:
```

## L. CATOPEN, CATREAD, CATCLOSE Intrinsics in a Pascal Program

This program opens a catalog, reads two messages, and prints them on the standard list device. It reads a third message into a buffer, prints the buffer, then closes the catalog.

```
$USLINIT$
$STANDARD_LEVEL 'HP3000'$

PROGRAM example (input,output);

TYPE int = -32768..32767;

VAR cat_index  : INTEGER;
    error      : PACKED ARRAY [1..2] OF int;
    cat_name   : PACKED ARRAY [1..8] OF CHAR;
    dummy,
    msg_len,
    set_num,
    msg_num,
    intr_id    : int;
    parm_n,
    parm_m     : STRING[40];
    buffer     : STRING[80];

FUNCTION  catopen: INTEGER; INTRINSIC;
FUNCTION  catread: int; INTRINSIC;
PROCEDURE catclose; INTRINSIC;

PROCEDURE show_error;  {a very simple "error printer"}
  BEGIN
    PROMPT(' error ',error [ 1 ]:1);
            { intr-id identifies the intrinsic called }
    CASE intr_id OF
      1 : WRITELN(' in CATOPEN');
      2 : WRITELN(' in CATREAD');
      3 : WRITELN(' in CATCLOSE');
    END;
  END;

BEGIN
              { Make sure that name ends with a space.}
  cat_name := 'EXAMPLE ';
  intr_id := 1;
  cat_index := catopen(cat_name,error);
  IF error [ 1 ] <> 0 THEN show_error;

  parm_n := '59';                     { set parameter 1 }
                              { append a null character }
  STRWRITE(parm_n,STRLEN(parm_n)+1,dummy,CHR(0));

  parm_m := 'thirty-three';           { set parameter 2 }
                              { append a null character }
  STRWRITE(parm_m,STRLEN(parm_m)+1,dummy,CHR(0));

  intr_id := 2;
```

```
   set_num := 3;          { set the message set number }
   msg_num := 17;             { set the message number }
   msg_len := catread(cat_index,set_num,msg_num,error,,,
                      parm_n,parm_m);
      { pass parameters 1 and 2, and print on $STDLIST }
   IF error [ 1 ] <> 0 THEN show_error;

   msg_num := 23;          { change the message number }
   msg_len := catread(cat_index,set_num,msg_num,error,,,
                      parm_n,parm_m);
      { pass parameters 1 and 2, and print on $STDLIST }
   IF error [ 1 ] <> 0 THEN show_error;

   set_num := 7;                 { change the set number }
   msg_num := 9;                 { set the message number }
                    { get the message into the buffer }
   msg_len := catread(cat_index,set_num,msg_num,error,
                      buffer);
   IF error [ 1 ] <> 0 THEN show_error;
                    { update the length of the buffer }
   SETSTRLEN(buffer,msg_len);
   WRITELN(buffer);                { now write the buffer }

   intr_id := 3;
   catclose(cat_index,error);
   IF error [ 1 ] <> 0 THEN show_error;

 END.
```

This program uses a message catalog file. To build this file, enter the following text into a text file:

```
$set 3    Comment describing this set's contents.
$
17    There is an error in line !1 on page !2.
23    On page !2 there is an error in line !1.
$
$set 7    Description of this set of messages.
$
09 Process completed successfully.
```

Use the GENCAT program to format this file into a catalog file called EXAMPLE. Executing the sample program results in the following:

```
:RUN PROGRAM
   There is an error in line 59 on page thirty-three.
   On page thirty-three there is an error in line 59.
Process completed successfully.

END OF PROGRAM
:
```

# Glossary

The following are definitions of NLS terms:

JISCII

The Japanese version of USASCII. It is a 7-bit character set identical to USASCII with the exception that the Japanese yen symbol replaces the "/" character.

KANA8

The Hewlett-Packard supported 8-bit character set for the support of phonetic Japanese (KATAKANA). It includes all of JISCII plus the KATAKANA characters. Refer to Appendix B for the table of KANA8 characters.

Limited Support

Refer to "Notes", in Appendix E, for each specific peripheral.

Old ROMAN8

USASCII plus Roman Extension. The manuals for terminals supporting old ROMAN8 contain this table.

Processing Standard

The internal Hewlett-Packard 8-bit processing standard for all Hewlett-Packard products. This standard was developed in anticipation of NLS and specifies standard character sets, escape sequences, character designations and invocations, and keyboard operation for peripherals and systems.

ROMAN8

The Hewlett-Packard supported 8-bit character set for Europe includes all of USASCII plus those characters necessary to support the major western European languages. Refer to Appendix B for the table of ROMAN8 characters.

Roman Extension

Part of the "old ROMAN8" as implemented on a number of older Hewlett-Packard terminals and printers. It is not a character set in itself but refers to an extension to USASCII. This extension is usually implemented as an alternate character set. The characters in Roman Extension form a subset of the non-USASCII characters in ROMAN8, and the same internal codes are used in both cases.

# Index