900 Series HP 3000 Computer Systems

MPE XL Native Language Programmer's Guide

HEWLETT PACKARD

HP Part No. 32650-90022
Printed in U.S.A. 1990

Second Edition
E0490

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 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.

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	**Date**	**Software Version**
First Edition	November 1987	A.01.00
Update 1	July 1988	A.10.00
Second Edition	April 1990	A.40.00

Preface

The Native Language Programmer's Guide is written for experienced programmers. It provides the HP 3000 programmer with the features necessary to produce localized application programs for end users without reprogramming for each country or language.

The following information is contained in this manual:

Chapter 1 **Introduction** introduces the subject matter of this manual.

Chapter 2 **Supported Native Languages** describes the character sets supported and the language-dependent characteristics of each.

Chapter 3 **Native Language Support in MPE XL** describes the utility programs, system intrinsics, and the Application Message Facility components of NLS.

Chapter 4 **NLS in the Subsystems** describes the NLS features within subsystems which provide the tools necessary for the design of local language applications.

Chapter 5 **Accessing NLS Features** describes how to access features through application programs or interactively by the user of a subsystem program.

Chapter 6 **Implicit Language Choices in Subsystems** describes how to designate a default language other than Native-3000 for the subsystems.

Chapter 7 **Application Programs Accessing NLS** describes the possible application models available for single language applications, multilingual applications, and subsystem utility programs.

Appendix A **Character Sets** identifies the characters sets supported by NLS.

Appendix B **Collating Sequences** explains and identifies the collating sequence used by NLS.

Appendix C **EBCDIC Mapping** identifies the mapping provided by NLS from supported character sets to various national versions of EBCDIC code.

Appendix D **Converting 7-Bit to 8-Bit Data** identifies the peripherals that must be converted and the conversion utilities available to convert from 7-bit to 8-bit operation.

Appendix E **Application Guidelines** identifies the supported programming languages and specific guidelines for each.

Appendix F **Example Programs** includes examples of programming languages with calls to NLS-related features.

Conventions

`UPPERCASE`

In a syntax statement, commands and keywords are shown in uppercase characters. The characters must be entered in the order shown; however, you can enter the characters in either uppercase or lowercase. For example,

```
COMMAND
```

can be entered as any of the following:

```
command      Command      COMMAND
```

It cannot, however, be entered as:

```
comm         com_mand     comamnd
```

italics

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

`COMMAND` *filename*

bold italics

In a syntax statement, a word in bold italics represents a parameter that you must replace with the actual value. In the following example, you must replace ***filename*** with the name of the file:

`COMMAND(`***filename***`)`

punctuation

In a syntax statement, punctuation characters (other than brackets, braces, vertical bars, and ellipses) must be entered exactly as shown. In the following example, the parentheses and colon must be entered:

`(`*filename*`):(`*filename*`)`

underlining

Within an example that contains interactive dialog, user input and user responses to prompts are indicated by underlining. In the following example, yes is the user's response to the prompt:

```
Do you want to continue? >>  yes
```

`{ }`

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

$$\texttt{COMMAND} \left\{ \begin{array}{l} \texttt{ON} \\ \texttt{OFF} \end{array} \right\}$$

Conventions (continued)

`[ ]`

In a syntax statement, brackets enclose optional elements. In the following example, `OPTION` can be omitted:

`COMMAND` *filename* `[OPTION]`

When several elements are stacked within brackets, you can select one or none of the elements. In the following example, you can select `OPTION` or *parameter* or neither. The elements cannot be repeated.

$$\texttt{COMMAND}\ \textit{filename}\ \begin{bmatrix}\texttt{OPTION}\\ \textit{parameter}\end{bmatrix}$$

`[ ... ]`

In a syntax statement, horizontal ellipses enclosed in brackets indicate that you can repeatedly select the element(s) that appear within the immediately preceding pair of brackets or braces. In the example below, you can select *parameter* zero or more times. Each instance of *parameter* must be preceded by a comma:

`[,`*parameter*`][...]`

In the example below, you only use the comma as a delimiter if *parameter* is repeated; no comma is used before the first occurrence of *parameter*:

`[`*parameter*`][,...]`

`| ... |`

In a syntax statement, horizontal ellipses enclosed in vertical bars indicate that you can select more than one element within the immediately preceding pair of brackets or braces. However, each particular element can only be selected once. In the following example, you must select `A`, `AB`, `BA`, or `B`. The elements cannot be repeated.

$$\begin{Bmatrix}\texttt{A}\\ \texttt{B}\end{Bmatrix}\ |\ \ldots\ |$$

`...`

In an example, horizontal or vertical ellipses indicate where portions of an example have been omitted.

`Δ`

In a syntax statement, the space symbol `Δ` shows a required blank. In the following example, *parameter* and *parameter* must be separated with a blank:

(*parameter*)Δ(*parameter*)

[ ] The symbol [ ] indicates a key on the keyboard. For example, [RETURN] represents the carriage return key or [Shift] represents the shift key.

[CTRL]*character* [CTRL]*character* indicates a control character. For example, [CTRL]Y means that you press the control key and the Y key simultaneously.

Conventions (continued)

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.

If no base is specified, decimal is assumed.

bits (*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 bits (*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|
|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
bits (0:1)                             bits (13:3)
```

Contents

A. Character Sets

B. Collating Sequences

C. EBCDIC Mapping

D. Converting 7-Bit to 8-Bit Data

E. Application Guidelines

F. Example Programs

Figures

Tables

1

Introduction

The Hewlett-Packard Native Language Support (NLS) is a feature of the MPE XL system and its associated subsystems. It enables the applications designer/programmer to write applications in the end user's local language.

Why NLS?

A well-written application program manipulates data and presents it appropriately for its use. Programs written with the intention of providing a friendly user interface often make assumptions about the local customs and language of the end user. Program interface and processing requirements vary from country to country, or possibly 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. The end user's native language, customs, and/or data processing requirements may differ from those in the environment of the software developer. End users benefit from application programs which interact with them in their native language and 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. 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 that depend on local language or custom 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.

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

Scope of Native Language Support

NLS consists of features within MPE XL and in the FCOPY, IMAGE, KSAM, QUERY, SORT-MERGE, VPLUS, and COBOL II/XL subsystems. These facilities allow application programs to be designed and written with a local language user interface for the end user and locally correct internal processing.

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

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

2

Supported Native Languages

Native Language Support (NLS) is based on languages and character sets that are predefined and built into the operating system. These are referred to as supported languages. Hewlett-Packard has assigned a unique language name and language ID number to each language supported in NLS. In some cases, Hewlett-Packard has more than one supported language corresponding to a single natural language (for example, NLS supports French and Canadian-French, because upshifting is handled differently). 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 that is applicable in several countries. German may be used in Germany, Austria, Switzerland, and any other place it is requested.

The NATIVE-3000 language (an artificial language) represents the way the computer dealt with language before the introduction of NLS. For example, the collating sequence for NATIVE-3000 is the same as the order of characters in the USASCII code. The NATIVE-3000 date format is returned by the existing MPE XL intrinsic `FMTDATE`. Anytime NATIVE-3000 is used in a native language function, the result is identical to the function performed prior to the introduction of NLS. NLS intrinsic calls using a language parameter of 0 always work correctly, even if native languages have not been configured on the system.

8-Bit Character Sets

Within NLS, each supported language is associated with an 8-bit character set. (One character set may support many languages.) 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.

It is possible to build supersets of USASCII permitting encoding and manipulation of characters required by languages other than American English, by using the eighth bit. These supersets are referred to as 8-bit or extended character sets. New characters are added with code values in the range 161-254.

Note

All character sets are supersets of USASCII, and are occasionally referred to as ASCII character sets.

Another method of providing foreign characters not supported by NLS involves 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. The NLS 8-bit character sets support all USASCII characters (except for \ in KANA8) in addition to the characters needed to support several Western European-based languages, Middle Eastern countries, and KATAKANA.

Note

Because 8-bit character sets are used in NLS, all bits of every byte have significance. Application software must take care to preserve the eighth bit (high-order), not allowing it to be modified or reused for any special purpose. No differentiation should be made between characters that have the eighth bit turned off or on, as all are characters of equal status in the extended character set.

Language-Dependent Characteristics

For each native language supported by NLS, a number of characteristics are known:

- 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 result of upshifting any character in the corresponding character set.
- Currency symbols (in their relationship to numbers), date, time, and number formats are country and local custom dependent.
- Data processing tables for ASCII-to-EBCDIC and EBCDIC-to-ASCII conversion are affected by language as the EBCDIC codes are different from country to country.

Within NLS, all these characteristics are 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 the format used in England rather than to the English language. American refers to the language, formats, and tables used in the United States.

The exact information on any particular installed language is available programmatically through the `NLINFO` intrinsic (refer to the *MPE XL Intrinsics Reference Manual* (32650-90028)) or in a report from the NLUTIL program (refer to the *MPE XL System Utilities Reference Manual* (32650-90081)).

3

NLS Components

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

NLS System Utilities

System managers use LANGINST to select and configure native languages to be supported on their system(s). NLUTIL is used to obtain the details of languages installed on a system. For a full description of the LANGINST and NLUTIL utilities, refer to the *MPE XL System Utilities Reference Manual* (32650-90081).

Configuring Native Languages

Before native languages can be used on a system (except NATIVE-3000), they must be configured by the system manager using the LANGINST utility program. (Refer to the *MPE XL System Utilities Reference Manual* (32650-90081) for the LANGINST dialog.) The system manager can select which supported language(s) to configure and modify formats associated with them (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). After a language has been installed, language-specific information available in NLS may be used by any application program requesting it.

Note

All language configuration changes are effective only after a system startup; at that time, the languages are installed.

NLS Intrinsics

Application programs and Hewlett-Packard subsystems call NLS to obtain language-dependent information for any language installed on a system. The following table lists the NLS intrinsics and their functions:

Table 3-1. NLS Intrinsic Categories

Function	Intrinsic	Description
Information Retrieving	ALMANAC	Returns numeric data 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.
	NLSTRANSLATE	Translates strings to/from EBCDIC.
	NLSUBSTR	Returns a string.
	NLSWITCHBUF	Converts a string of characters from phonetic order to screen order and vice versa.
Number Formatting	NLCONVNUM	Converts numbers from native to internal form.
	NLFMTNUM	Formats an internal number in native form.
	NLNUMSPEC	Returns information needed for formatting and converting numbers.

Table 3-1. NLS Intrinsic Categories (continued)

Function	Intrinsic	Description
Application Message Catalog	`CATCLOSE`	Closes a message catalog.
	`CATOPEN`	Opens a message catalog.
	`CATREAD`	Reads information from a message catalog.
	`NLAPPEND`	Concatenates a file name and a language number.

Refer to the *MPE XL Intrinsics Reference Manual* (32650-90028) for a complete description of each intrinsic.

NLS provides support features for language data and local custom formats in FCOPY, IMAGE, KSAM, QUERY, SORT-MERGE, and VPLUS. 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 interface (prompts, commands, and messages) of the subsystem utility programs are not localized (for example, FORMSPEC and DBUTIL).

Note

None of these changes are automatic. All existing applications and jobs work the same way they did prior to the NLS installation unless they are modified to request NLS functions.

Peripheral Support

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. (Refer to the *System Startup, Configuration, and Shutdown Reference Manual* (32650-90042) for procedures.) Limitations and notes are listed for each peripheral. All NLS features are available to users with 7-bit USASCII terminals and printers, provided that the data used contains only USASCII characters. For example, a user in the United States can use American (the Hewlett-Packard name for American English) for sorting, date formatting, and message handling. This is possible because USASCII is a subset of ROMAN8.

NLS has no direct control over the peripherals configured on a system. It is the user's responsibility to configure peripherals that support the character set(s) necessary for the desired languages.

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

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 can convert this data. A set of utilities is available to convert 7-bit data to 8-bit (ROMAN8) data in KSAM files, IMAGE databases, VPLUS forms files, and MPE XL files. Refer to appendix D for 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 without modifying or recompiling the program source code.

The Application Message Facility is an NLS tool that provides a programmer with the flexibility needed to create application catalogs for localized applications. 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 `CATOPEN`, `CATREAD`, and `CATCLOSE` intrinsics as shown in Figure 3-1. The GENCAT utility creates and maintains message catalogs that meet the NLS requirements for efficient storage and retrieval of messages. For more information, refer to *Message Catalogs Programmer's Guide* (32650-90021).

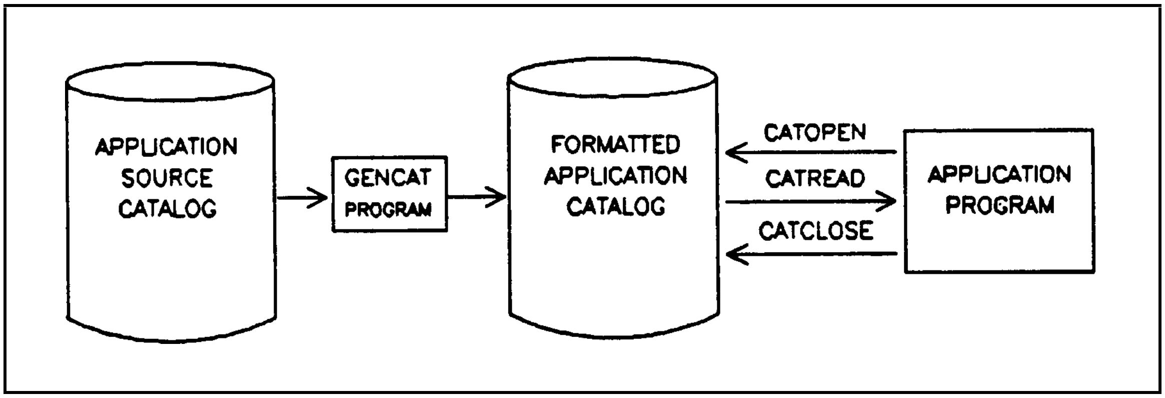


Figure 3-1. GENCAT Utility Program

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 message catalog 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 an MPE XL ASCII file (source catalog). The GENCAT program is used to read the source catalog and create a formatted catalog. The formatted catalog has an internal directory for efficient access and is compacted (for example, it deletes 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 file.
- 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). A suggested naming convention is discussed in chapter 7.

4

NLS in the Subsystems

NLS provides MPE XL intrinsic features in COBOL II/XL, FCOPY, IMAGE, KSAM, QUERY, SORT-MERGE, and VPLUS. NLS features in these subsystems provide the tools to design local language applications. The subsystems themselves are not localized. The application end user, not the programmer or subsystem user, sees the localized interface.

MPE XL Native Language Support intrinsics provide the means to implement NLS features of the subsystems. This means that native language definition is consistent within all the subsystems. Collating sequence is a good example of consistency within MPE XL and in the subsystems. The collating sequence defined for a specific native language can be used in MPE XL by calling the `NLCOLLATE` and `NLKEYCOMPARE` intrinsics. The same collating sequence is used by SORT-MERGE in ordering records, by KSAM in ordering keys, and by IMAGE in ordering sorted chains when these subsystems are dealing with sorted character strings that are associated with the same native language.

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

5

Accessing NLS Features

On HP 3000 systems using MPE XL 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

One option for obtaining NLS features from an application program is through calls to specific NLS intrinsics, primarily in MPE XL. To receive a local language date format, an application should call the `NLFMTDATE` intrinsic instead of the `FMTDATE` intrinsic.

Additional NLS features can be obtained by specifying values for extended or new parameters in existing intrinsics. For example, `SORTINIT` in SORT-MERGE has been extended to allow the specification of a character key and a native language ID number (*langnum*) that determines the collating sequence to be used.

Note

These additional parameters must be used in an application to sort according to native language values.

Native Language Attribute

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

Commands

Commands or keywords that make NLS features available on request have been added to certain subsystems. (For example, entering `LANGUAGE=FRENCH` within QUERY causes sorted character data of IMAGE types X and U to be sorted, in its output reports, according to the French collating sequence. If the language command is not entered, QUERY performs 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 E.

For information on how and when the individual subsystems are influenced, refer to the appropriate manual:

FCOPY Reference Manual (32212-90003)
TurboIMAGE/XL Reference Manual (30391-90001)
KSAM/3000 Reference Manual (30000-90079)
QUERY/V Reference Manual (30000-90042)
SORT-MERGE/XL Programmer's Guide (32650-90080)
Data Entry and Forms Management System VPLUS/3000 (32209-90001)

6

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, QUERY, and 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.

NLUSERLANG and NLDATALANG JCWs

NLUSERLANG and NLDATALANG are independent JCWs and are treated independently by NLS. In many cases, they specify the same language. Distinct values can be specified (for example, the HP Word product, which has the concepts of a user and a document language).

- NLUSERLANG designates the user interface and report output language for programs. If the subsystems are localized, this is the language of choice for prompts and messages. If user input data is modified (for example, upshifted by QUERY or VPLUS), this JCW determines which language's attributes are used. NLUSERLANG designates the default language for all language-dependent operations in QUERY and ENTRY.
- NLDATALANG designates the internal data manipulation language. This is distinct from NLUSERLANG because multiple subsystem users with different interface languages may share common internal data (for example, sorted according to one language). The data manipulation language is used in the SORT, MERGE, and FCOPY programs to control their language-dependent functions, such as collating, upshifting, and conversions to and from EBCDIC. Note that if the user interface of one of these programs is localized, it uses NLUSERLANG as its default for messages, prompts, and so on.

NLGETLANG Intrinsic

NLUSERLANG and NLDATALANG values are retrieved by the subsystems through calls to the `NLGETLANG` intrinsic. Application programs can 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` returns 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 the `SETJCW` command. The *MPE XL Commands Reference Manual* (32650-90003) lists the `SETJCW` command syntax.

User-Defined Commands (UDCs)

ENTRY, FCOPY, QUERY, SORT, and MERGE are often run from within user-defined commands (UDCs). NLUSERLANG and NLDATALANG give the application designer the option of establishing a native language within a UDC.

7

Application Programs Accessing NLS

The focus of HP 3000 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 as illustrated in Figure 7-1 through Figure 7-5. NLS capabilities can be used in single language applications, multilingual applications, in subsystem utility programs, or not at all.

File Naming Conventions

An application that is localized into several languages has separate message catalogs, VPLUS forms files, and other language-dependent data files for each of these languages. Establish a naming convention for these files that follows the language numbering used by NLS. To do this, use a file name that 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; the FILE008 would contain the same data modified for German; and FILE012 would contain SPANISH data.) 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.

General Application Program

The functions language can influence an application in terms of data manipulation (internals) and user interaction (externals), as illustrated in Figure 7-1. The core application program is flanked by functions that can differ according to language and local customs (local date, time, and currency formats).

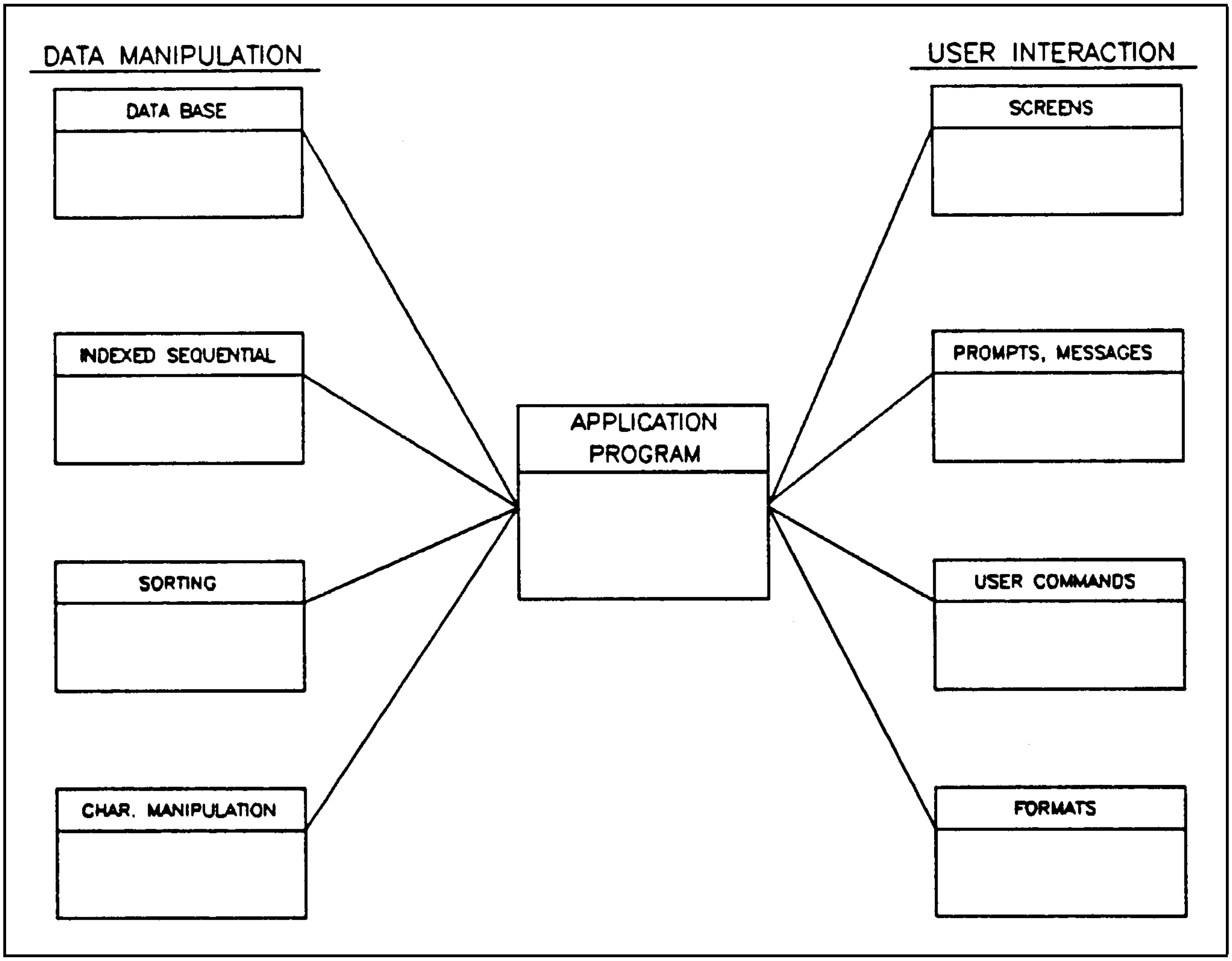
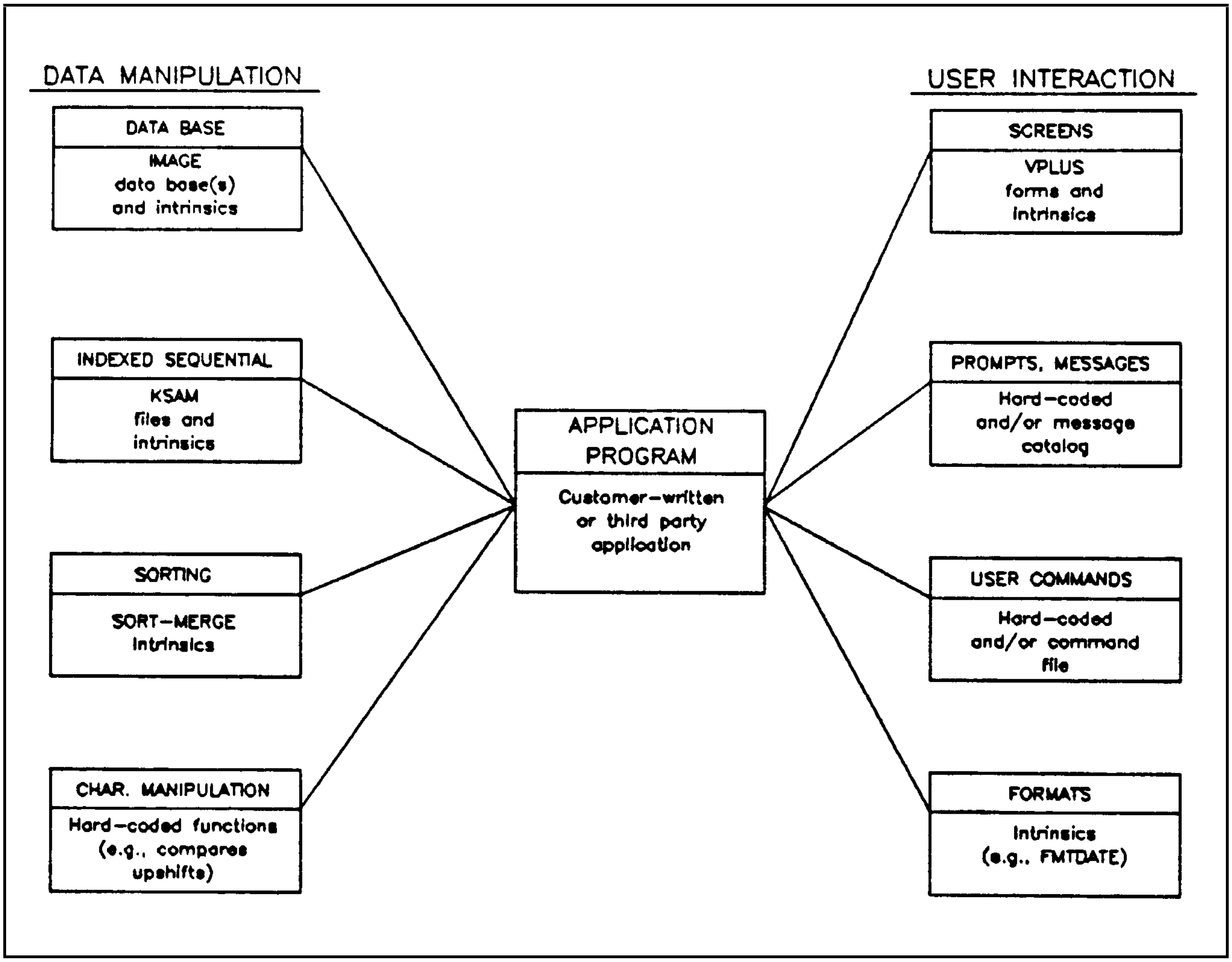


Figure 7-1. General Application Program Format

Application Program Without NLS

Figure 7-2 shows an application program that 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 7-2. Application Program Without NLS

Single Language Application

French is used as the single language application example in Figure 7-3. 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 databases, KSAM files, and VPLUS forms files with the French language attribute, and has expressed all prompts and messages in French. This use of NLS is for programs used in one country, location, or language only.

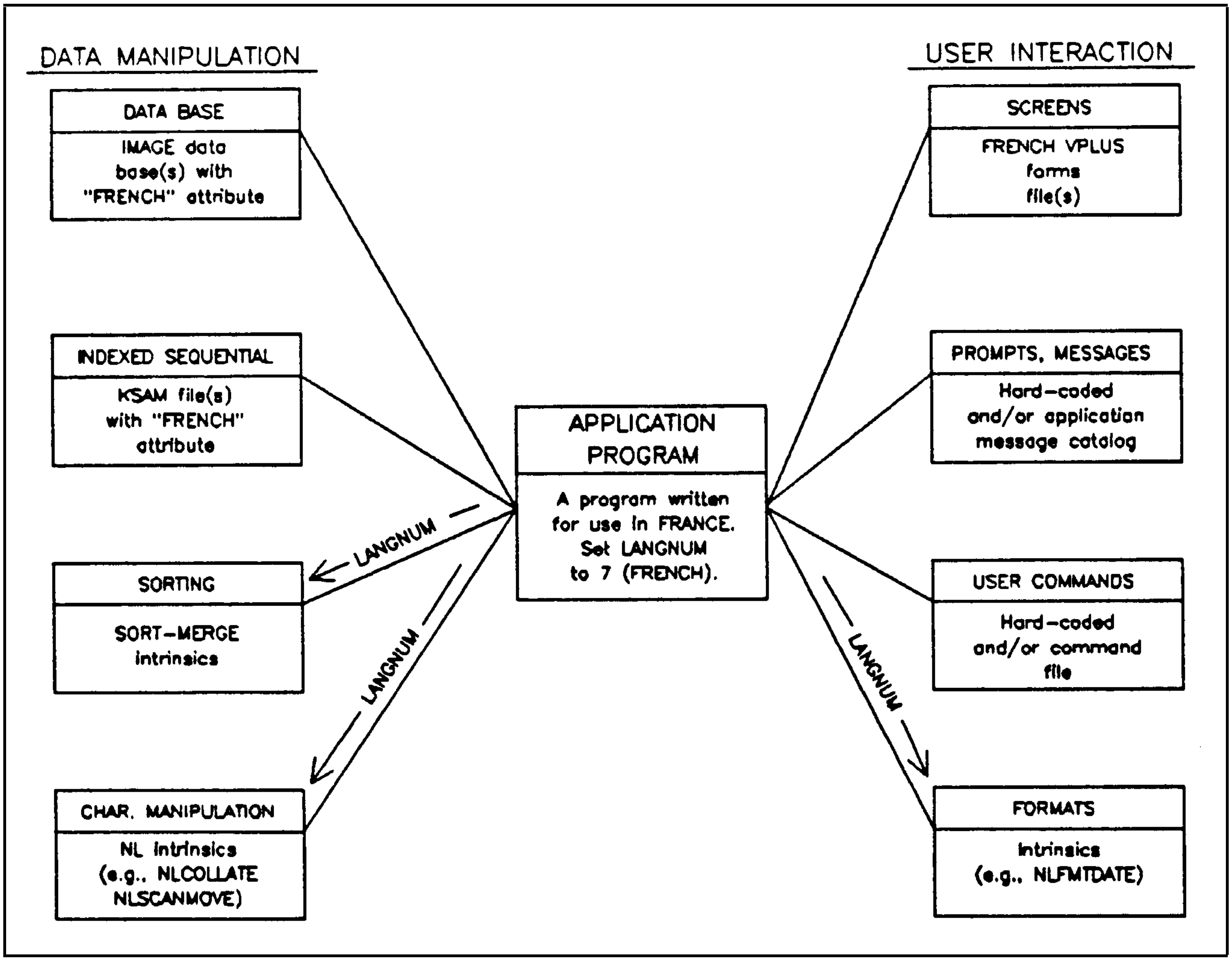

DATA MANIPULATION
USER INTERACTION
DATA BASE
IMAGE data base(s) with "FRENCH" attribute
INDEXED SEQUENTIAL
KSAM file(s) with "FRENCH" attribute
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
APPLICATION PROGRAM
A program written for use in FRANCE. Set LANGNUM to 7 (FRENCH).
LANGNUM
LANGNUM
LANGNUM
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)

Figure 7-3. Single Language Application

Multilingual Applications

The program in Figure 7-4 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 file(s) for that application. These files store information about various end-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 (which can be set by the system manager when native languages are configured), or it may prompt the end 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 too restrictive if many end users or transactions requiring different languages are handled from one job or session.

Once the languages have been determined, the program opens the appropriate VPLUS forms files, message catalogs, and/or command files, based on the user interface language choice. It also opens any needed IMAGE databases, KSAM 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 end 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.

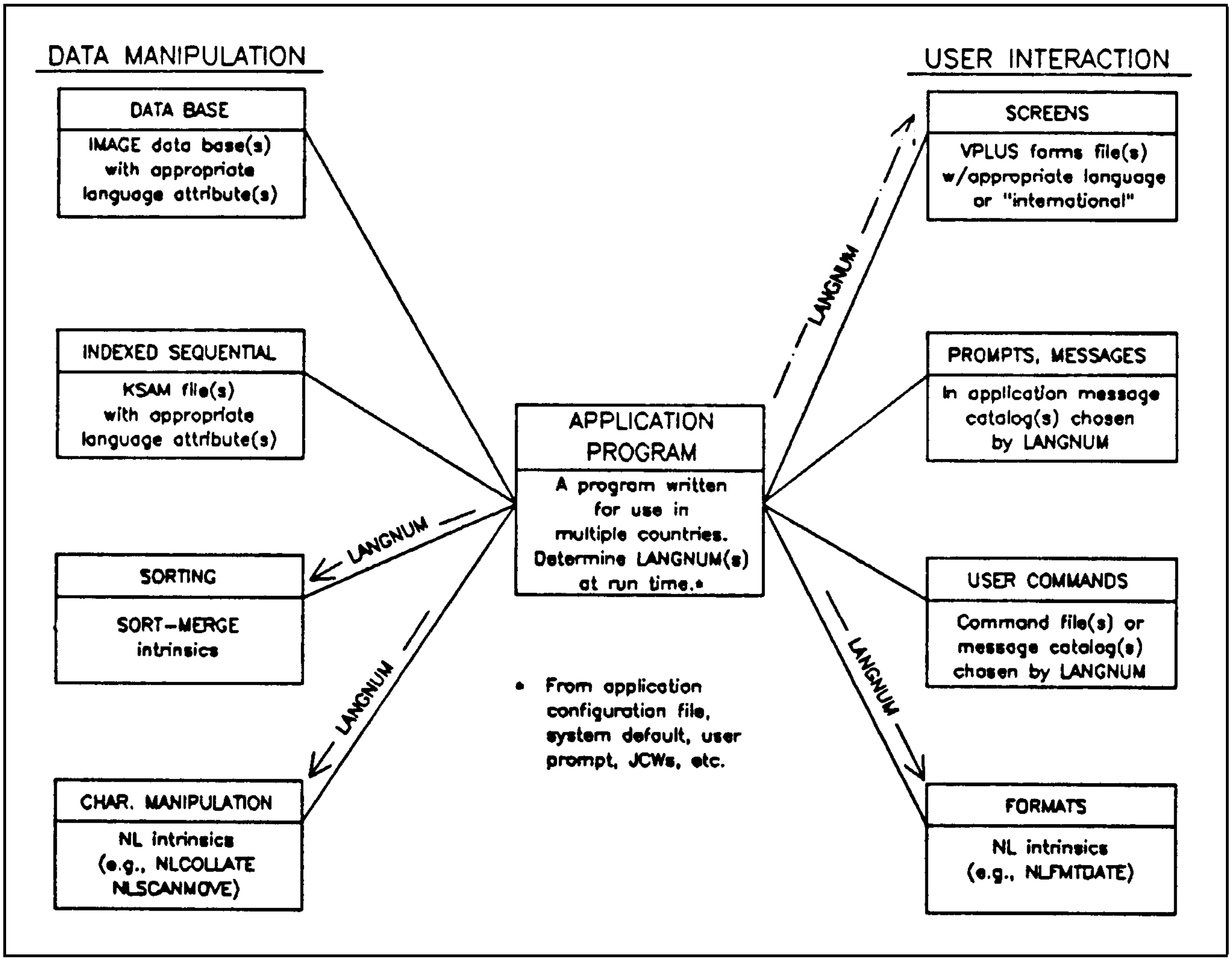

DATA MANIPULATION
USER INTERACTION
DATA BASE
IMAGE data base(s) with appropriate language attribute(s)
INDEXED SEQUENTIAL
KSAM file(s) with appropriate language attribute(s)
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
APPLICATION PROGRAM
A program written 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 7-4. Multilingual Application

Subsystem Utility Program

Figure 7-5 shows a special category of multilingual application, the Hewlett-Packard subsystem utility program. Many of these programs are not typically used by end users, but are used to manipulate end-user data in conjunction with application programs. They determine which language to use at run time through a user-entered keyword or command, or defaults.

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

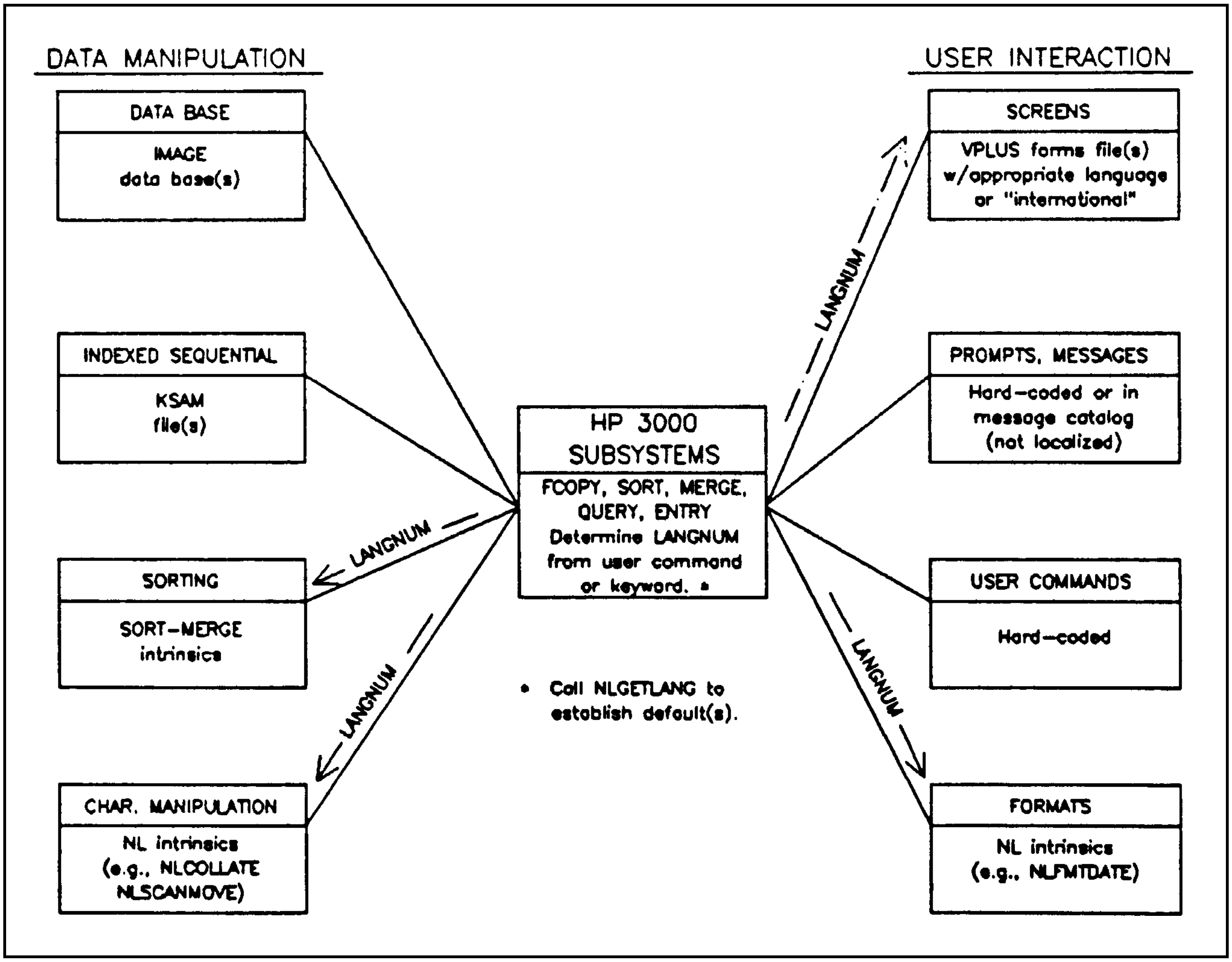

DATA MANIPULATION
DATA BASE
IMAGE data base(s)
INDEXED SEQUENTIAL
KSAM file(s)
SORTING
SORT-MERGE intrinsics
CHAR. MANIPULATION
NL intrinsics (e.g., NLCOLLATE NLSCANMOVE)
LANGNUM
LANGNUM
HP 3000 SUBSYSTEMS
FCOPY, SORT, MERGE, QUERY, ENTRY
Determine LANGNUM from user command or keyword. *
* Call NLGETLANG to establish default(s).
LANGNUM
LANGNUM
USER INTERACTION
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 7-5. Subsystem Utility Program

For every character set, a character attribute table is defined. This table of 256 entries holds an attribute (type) for every character.

A

Character Sets

NLS supports ten character sets containing the following native languages:

Table A-1. Languages Supported by NLS

Language Set Number	Name	Language ID Number	Language
0	USASCII	00	NATIVE-3000
1	ROMAN8	00	NATIVE-3000
		01	AMERICAN, ENGLISH
		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
2	KANA8	00	NATIVE-3000
		41	KATAKANA
3	ARABIC8	51	ARABIC
		52	WESTERN ARABIC
4	GREEK8	61	GREEK
6	TURKISH8	81	TURKISH
51	PRC15	201	SIMPLIFIED CHINESE (CHINESE-S)
56	ROC15	211	TRADITIONAL CHINESE (CHINESE-T)
61	JAPAN15	221	JAPANESE
66	KOREA15	231	KOREAN

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 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).
- The National Date table (where applicable).

The character sets supported by NLS are included on the following pages:

				b8	0	0	0	0	0	0	0	0
				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	″	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	′	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	∧	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Figure A-1. USASCII Character Set

				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	▨	▨	▨	—	â	Å	Á	Þ
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q	▨	▨	À	Ý	ê	Î	Ã	þ
0	0	1	0	2	STX	DC2	"	2	B	R	b	r	▨	▨	Â	ý	ô	Ø	ã	·
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s	▨	▨	È	°	û	Æ	Đ	µ
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	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	▨	▨	^	¤	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	β	Ÿ	±
1	1	1	1	15	SI	US	/	?	O	_	o	DEL	▨	▨	£	¢	ü	Ô	ÿ	▨

Figure A-2. ROMAN8 Character Set

				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 A-3. KANA8 Character Set

				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	▨	▨	▨	·	@	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	ئ	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	–		▨

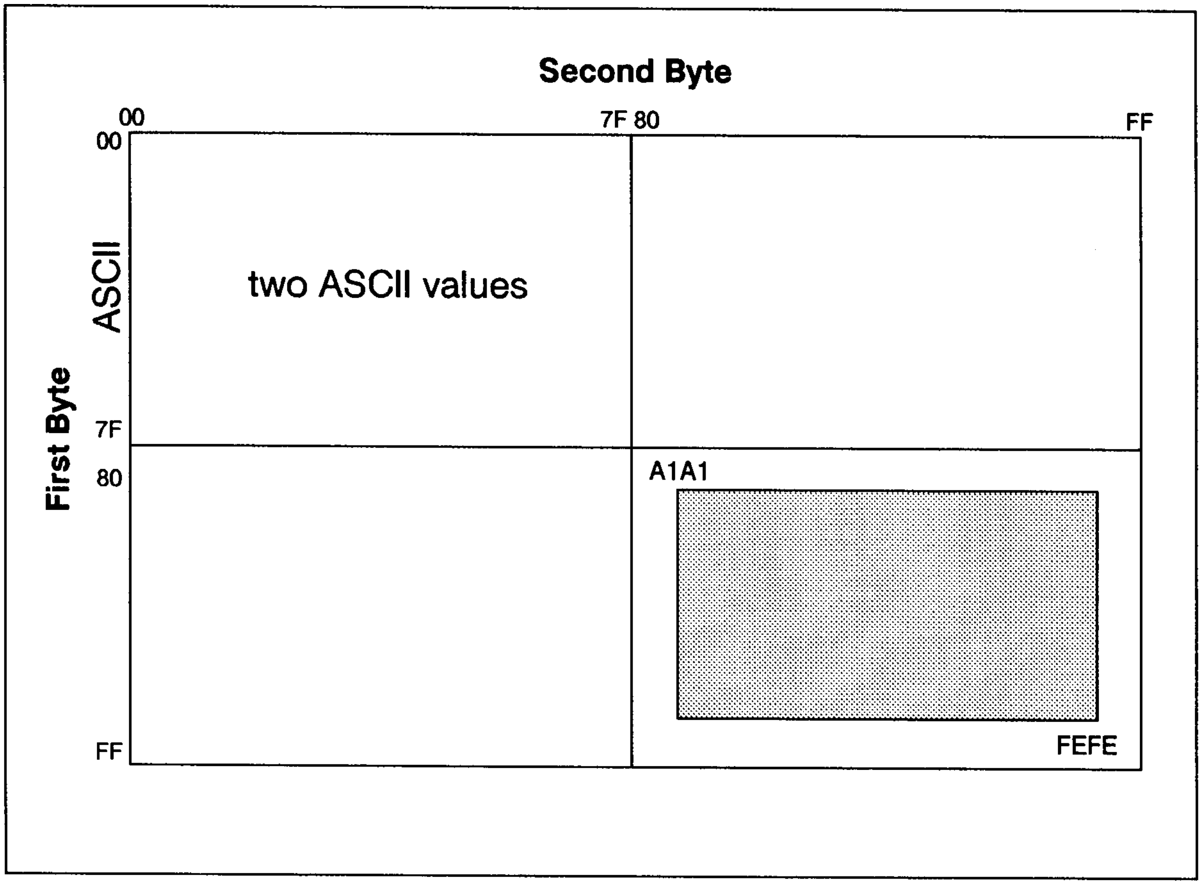
Figure A-4. ARABIC8 Character Set

				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		ω̇
1	0	1	1	11	VT	ESC	+	;	K	[	k	{	▨	▨			Κ	ά	κ	ξ́
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 A-5. GREEK8 Character Set

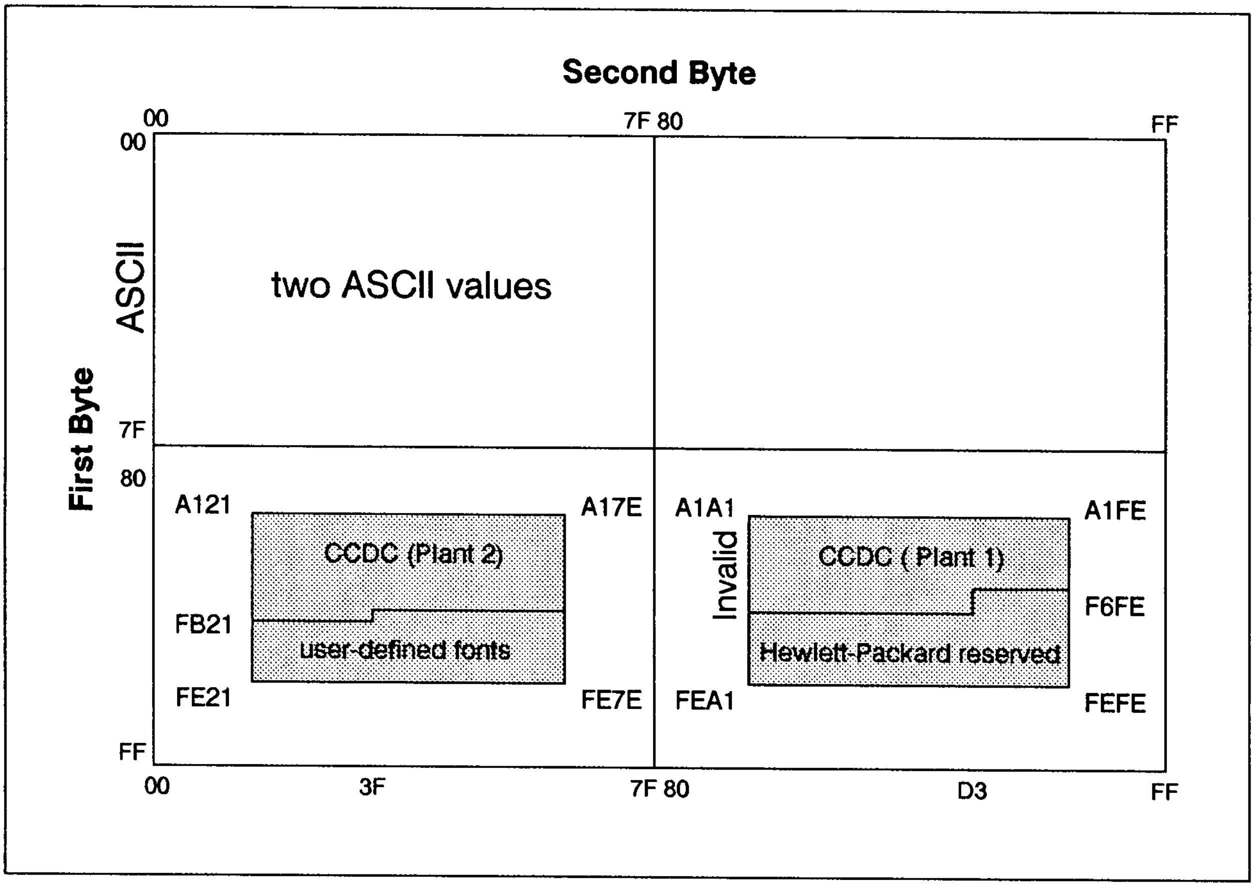
				b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p	▨	▨	▨			Å	ğ	Þ
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q	▨	▨	Ç	Ý	ê	î	Ã	þ
0	0	1	0	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	Đ	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	▨	▨	^	₺	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 A-6. TURKISH8 Character Set



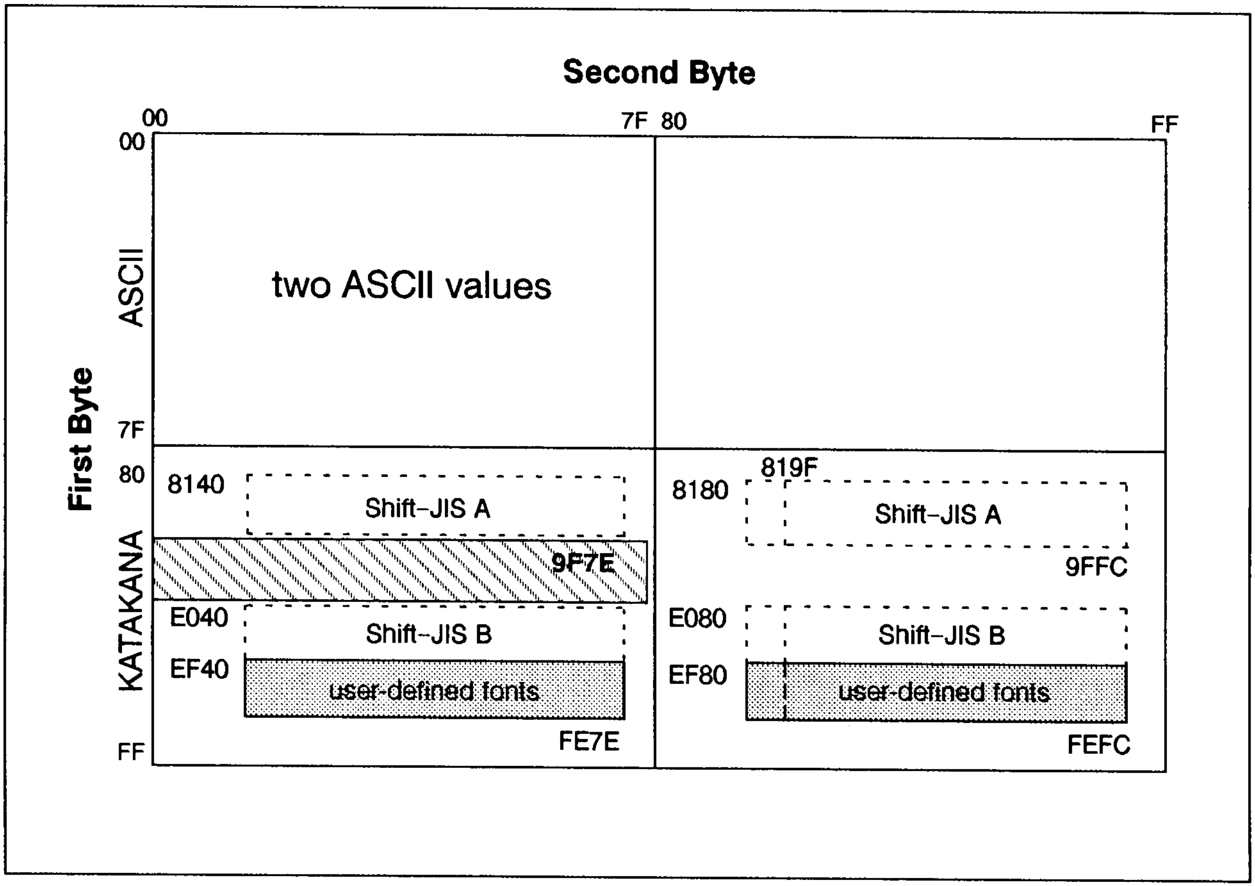
LG200165_002a

Figure A-7. PRC15 Character Set



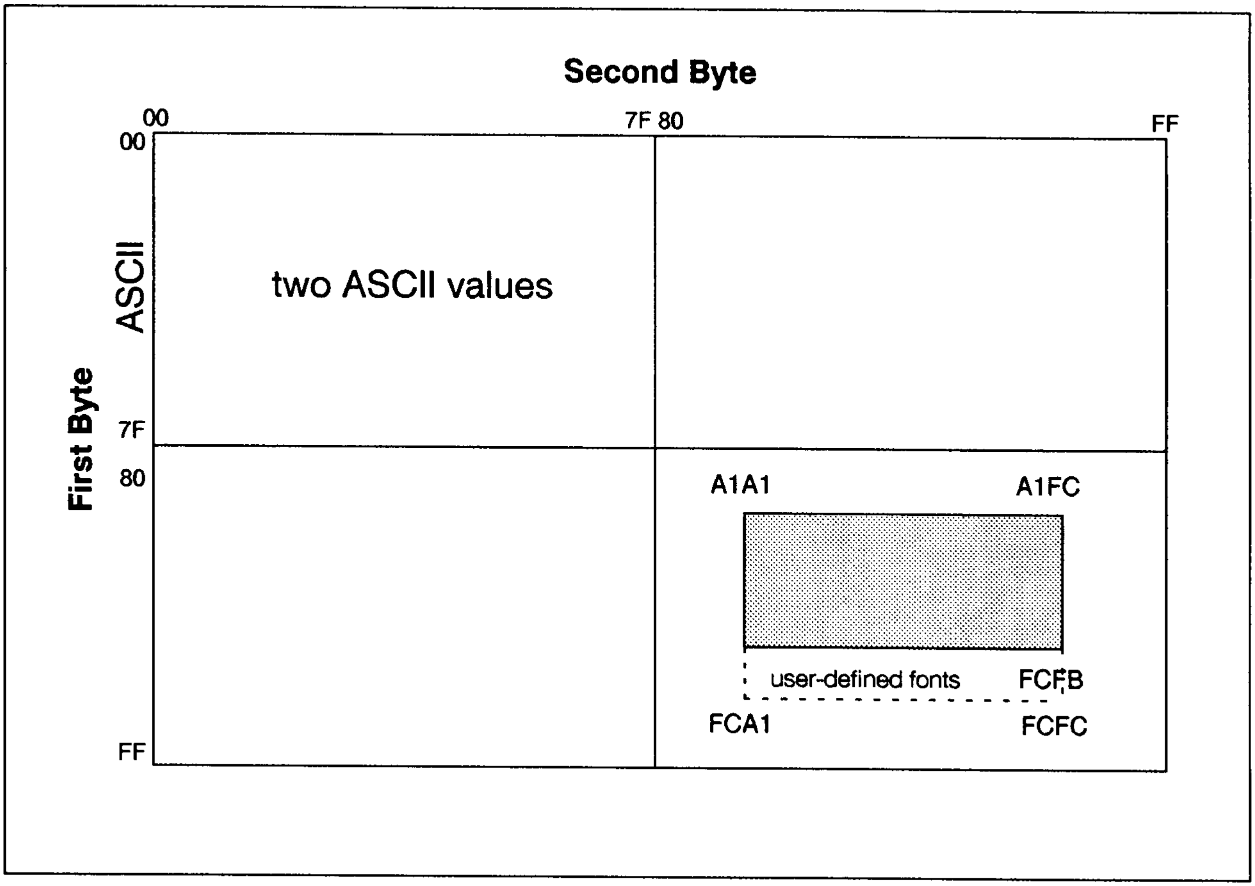
LG200165_001

Figure A-8. ROC15 Character Set



LG200165_003

Figure A-9. JAPAN15 Character Set



LG200165_002

Figure A-10. KOREA15 Character Set

B

Collating Sequences

Collating is defined as arranging character strings into order (usually alphabetic). 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.

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.

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. Table B-1 lists the collating sequence for American-English, Canadian-French, Danish, Dutch, English, Finnish, French, German, Italian, Norwegian, Portuguese, Spanish, and Swedish.

All characters in an alpha or numeric group collate the same. These characters usually differ only in uppercase versus lowercase priority, or accent priority. (Refer to Table B-2 for collating sequences.) In sorting, they are initially considered the same. If characters in the two strings do not determine which string comes first, then the priorities of characters are used to determine the order. Refer to Table B-1 for examples of collating sequence priority.

Table B-1. Collating Sequence Priority

Example Sorted Strings	Priority Explanation
aEb, aEc	The third character in each string is different. The “b” precedes the “c”.
aéb,aEb	The characters in the two strings are identical, so accent priority determines the order. The “é” precedes the “E”.
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. The “B” precedes the “b”.

Table B-2 displays the collating sequence in three ways:

- The graphic representation of the character.
- The decimal equivalent of the character’s binary value.
- A description of the character.

Table B-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

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
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
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

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
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
L	76	Uppercase L
l	108	Lowercase l
M	77	Uppercase M
m	109	Lowercase m
N	78	Uppercase N
n	109	Lowercase n
Ñ	182	Uppercase N tilde
ñ	183	Lowercase n tilde
O	79	Uppercase O
o	110	Lowercase o

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
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
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

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
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
	177-178	Currently undefined
	242-245	Currently undefined
(	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

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
+	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
¿	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

Table B-2. Collating Sequence (continued)

Character	Decimal Equivalent	Description
"	34	Double quote
‘	96	Opening single quote
’	39	Closing single quote
^	96	Caret
~	126	Tilde
´	168	Acute grave
`	169	Accent grave
^	170	Accent circumflex
¨	171	Umlaut/Diaeresis
~	172	Tilde accent
_	95	Underscore
—	246	Long dash
—	176	Overline
ª	249	Feminine ordinal sign
º	250	Masculine ordinal sign
■	252	Solid
	0-31	Control codes
	127	DEL
	128-159	Undefined control codes
	255	Do not use

Note

The Æ (uppercase AE ligature) and æ (lowercase ae ligature) are expanded for collating purposes to **AE** or **ae** and collates as:

`ad AE Ae aE ae AF`

The *β* (sharp s) is expanded for collating purposes to ss and collates according to the German standard as:

`sr ss st`

Table B-3 through Table B-6 show the language-dependent variations to the collating sequence.

Language-Dependent Variations

Following are the language-dependent variations to the collating sequences for Spanish, Danish/Norwegian, Swedish, and Finnish.

Spanish

The CH is considered a separate character, which collates between C and D:

`C@` <u>`CH Ch cH ch`</u> `D` *Where @ equals anything; therefore, CH comes after C followed by anything and before D.*

The LL is considered a separate character, which collates between L and M:

`l@` <u>`LL Ll lL ll`</u> `M` *Where @ equals anything; therefore, LL comes after L followed by anything and before M.*

`N` and `Ñ` are not considered the same when collating; they follow one another in the collating sequence:

Table B-3. Spanish Language-Dependent Variations

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:

Table B-4.
Danish/Norwegian Language-Dependent Variations

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

Swedish

The Å, Ä, and Ö collate at the end of the alphabet:

Table B-5. Swedish Language-Dependent Variations

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 collate at the end of the alphabet. Ø is considered the same as Ö and V, W, Y, and Ü are considered the same:

Table B-6. Finnish Language-Dependent Variations

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
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

Table B-6. Finnish Language-Dependent Variations (continued)

Character	Decimal Equivalent	Description
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

C

EBCDIC Mapping

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 that 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 (ˆ), 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 lower case **n**. 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, the empty space was used to accommodate characters from other languages, but the traditional German characters (L, N, O, and β) 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 for example, the character that has decimal code 161 (octal 241, hexadecimal A1). In original EBCDIC, it is the ˜ (tilde); in German, the sharp β; in French, the ¨ (diaeresis accent); in Swedish/Finnish and Norwegian/Danish, the lower case ü; in Italian, the lower case ì; and in Portuguese, the lower case ç.

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 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 that 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-alikes in ROMAN8. For these characters a one-to-one mapping has been defined as shown in Table C-1.

Table C-1. ROMAN8 to EBCDIC Mapping

Decimal	Octal	Hex	ROMAN8		EBCDIC	
169	251	A9	‘	Grave accent	|	Logical OR
170	252	AA	ˆ	Circumflex accent	¬	Logical NOT
172	254	AC	˜	Tilde accent	^{2}	Superscript 2
175	257	AF	₤	Italian lira sign	^{3}	Superscript 3
177	261	B1		Undefined	μ	MU character
178	262	B2		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		Undefined	♭	Cedilla
243	363	F3		Undefined	¶	Paragraph sign
244	364	F4		Undefined	®	Registered sign
245	365	F5		Undefined	$\frac{3}{4}$	Three quarters
246	366	F6	—	Long dash	**SHY**	Syllable hyphen
252	374	FC	■	Solid	•	Middle dot

The mapping to and from EBCDIC for the KANA8 character set is defined by Japanese Industrial Standards (JIS) and IBM.

In all languages, the character mappings defined and implemented are 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 the NLUTIL utility. (Refer to the *MPE XL System Utilities Reference Manual* (32650-90081).)

The mappings can be made available to a program by using `NLINFO` *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 or 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`.

D

Converting 7-Bit to 8-Bit Data

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 that 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. Eight bits are required 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 e in Spanish or French would be produced with an accent mark plus an e, whereas ROMAN8 contains the e 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 that support them:

```
DANSK/NORSK
DEUTSCH
ESPANOL
ESPANOL M
FRANCAIS
FRANCAIS M
ITALIANO  (On a few devices only.)
SVENSK/SUOMI
UK
```

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 HP 262X terminals in 7-bit operation for many years may have a substantial investment in data that is encoded in one of these 7-bit national substitution character sets. Hewlett-Packard makes several conversion utilities 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 need to configure their peripherals for 8-bit operation. (This is not possible for the HP 264X terminals.)

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 D-1 for a listing of these utilities. These utilities need to be run once for each file needing conversion, and peripherals need to be reconfigured for 8-bit operation.

Table D-1. Conversion Utilities

File Type	Conversion Utility
EDITOR files	N7MF8CNV (text option)
Other MPE files (all text)	N7MF8CNV (text option)
MPE files where text data is organized in fields and needs 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 databases	I7DB8CNV
VPLUS forms files	V7FF8CNV
HP WORD files	HP WORD internal files are based on a subset of ROMAN8; no conversion required
TDP files	Run N7MF8CNV, then change back the command backslashes that were converted in the chosen language (the command backslash is required for imbedded TDP commands)

Conversion Algorithm

The conversion utilities convert records or fields from files that 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 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 of certain characters 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. The conversion affects only the 12 characters shown in Table D-2 (all other characters remain unchanged). There are two rows of information opposite each national substitution set listed in Table D-2:

- The upper row indicates the graphic assigned in 7-bit operation.
- The lower row indicates the decimal value assigned the graphic in ROMAN8 after using the conversion algorithm.

Table D-2. Character Conversion Decimal Values

National Substitution Set	**Character and Conversion Decimal Value**											
USASCII	# 35	’ 39	@ 64	[91	\ 92	] 93	ˆ 94	‘ 96	{ 123	| 124	} 125	˜ 126
SVE/SUOMI	# 35	’ 39	É 220	Ä 216	Ö 218	Å 208	Ü 219	é 197	ä 204	ö 206	å 212	ü 207
DANSK/NORSK	# 35	’ 39	@ 64	Æ 211	Ø 210	Å 208	ˆ 94	‘ 96	æ 215	ø 214	å 212	˜ 126
FRANCAIS	£ 187	’ 39	à 200	° 179	ç 181	§ 189	ˆ 170	‘ 96	é 197	ù 203	è 201	¨ 171
FRANCAIS M	£ 187	’ 39	à 200	° 179	ç 181	§ 189	ˆ 170	‘ 96	é 197	ù 203	è 201	¨ 171
DEUTSCH	£ 187	’ 39	§ 189	Ä 216	Ö 218	Ü 219	ˆ 94	‘ 96	ä 204	ö 206	ü 207	β 222
UK	£ 187	’ 39	@ 64	[91	\ 92	] 93	ˆ 94	‘ 96	{ 123	| 124	} 125	˜ 126
ESPANOL	# 35	’ 39	@ 64	¡ 184	Ñ 182	¿ 185	° 179	‘ 96	{ 123	ñ 183	} 125	˜ 126
ESPANOL M	# 35	’ 39	@ 64	¡ 184	Ñ 182	¿ 185	° 179	‘ 96	{ 123	ñ 183	} 125	˜ 126
ITALIANO	£ 187	’ 39	@ 64	° 179	ç 181	é 197	ˆ 94	ù 203	à 200	ò 202	è 201	ì 217

Table D-3 shows the special character conversion for specified FRANCAIS M and ESPANOL M characters. If these characters are followed immediately by certain 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 D-3. Special Character Conversion

Language	Character Conversion
Francais M	ˆ (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), or 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).

Note

If these characters are followed by any other character, they are converted to their ROMAN8 equivalent as shown in Table D-2.

Conversion Procedure

To convert 7-bit substitution data to 8-bit ROMAN8 data, perform the following steps:

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 USASCII.
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. Refer to Table D-1 to determine which utility should be used to convert each file.
4. Back up all files to be converted (store to tape or SYSGEN).
5. Run each utility, supplying it with the language and file names 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 *System Startup, Configuration, and Shutdown Reference Manual* (32650-90042) for peripheral configuration.

Figure D-1 is a sample dialog 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
```

Figure D-1. N7MF8CNV Dialog

```
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
:
```

Figure D-1. N7MF8CNV Dialog (continued)

N7MF8CNV Utility

N7MF8CNV converts data in EDIT/XL 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 is 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 saved in its place. An exception to this is KSAM 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 does 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 where 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 files. Since these are converted in place, some records may already have been modified. KSAM 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 file name.

In addition to the text and data options, there is a test conversion option that 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, a (Return) exits the current program level. 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 that 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 database from any 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
   ```

   This unloads the database to tape.

2. Enter:

   ```
   RUN DBUTIL.PUB.SYS,ERASE
   ```

   This erases the database data.

3. Enter:

   ```
   RUN I7DB8CNV
   ```

   This converts the data and reloads it into the database.

I7DB8CNV requests the following:

1. The 7-bit national substitution set where 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?`. A `YES` or Return response converts all data fields of type U or X. A `NO` response prompts the user 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) it is linked to for 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 a `YES` answer; an `N` allows the user 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 that were selected have their data converted according to the algorithm for the 7-bit national substitution set selected at the beginning of the program.

I7DB8CNV does not allow 8-bit data (bytes with the high-order bit set) in the data fields it is trying to convert. The utility does not abort, but the field in question is not converted, and a warning is 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.

Figure D-2 shows the dialog from 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. UK
   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
:
```

Figure D-2. I7DB8CNV Dialog

V7FF8CNV Utility

V7FF8CNV converts text and literals in VPLUS/XL 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 SYSGEN), perform the following steps:

1. Configure your terminal for 8-bit operation. Refer to *System Startup, Configuration, and Shutdown Reference Manual* (32650-90042) for information on peripheral configuration.
2. Run `V7FF8CNV.PUB.SYS`, stepping through each form, field definition, save field, and 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 record VPLUS/XL 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 of handling all or part of ROMAN8, can be divided into two groups. The group differentiation is based on how they handle alternate character sets when configured for 8-bit operation.

Group 1 - 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 standard for new Hewlett-Packard terminals and printers.

Group 2 - 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 2 terminals use the eighth bit to switch back and forth between a 7-bit base character set and a 7-bit alternate character set. 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 and return to the alternate character set when the high-order bit is on.

Files using alternate character sets on one group of terminals do not display correctly on the terminals of the other group, even when terminals from both groups are configured for 8-bit operation.

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:

- Eliminate the use of alternate character sets (either with FORMSPEC or while running V7FF8CNV).
- Define alternate character sets to appear correctly on Group 1 terminals. This happens automatically when V7FF8CNV is run from a Group 1 terminal. Characters from these alternate sets appear as USASCII characters on a Group 2 terminal.

V7FF8CNV Operation

V7FF8CNV must be run on a terminal supported by VPLUS/XL, which supports display of all characters, enhancements, and alternate character sets used in the forms. 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 *System Startup, Configuration, and Shutdown Reference Manual* (32650-90042).
2. Run `V7FF8CNV.PUB.SYS`. Respond to prompts for the terminal group and the national substitution set.
3. Press (Next) to begin going through the forms file.
4. Press (Enter) after each screen until the end of the forms file is reached. Two exceptions are:
   a. Enter `Y` in `Function key labels` on each FORM MENU and the GLOBALS MENU to see and convert function key labels.
   b. On the field definition screen, if the processing specifications have converted data that you want to save, press the FIELD TOGGLE key then (Enter) to save that conversion.

Note

If you try to redisplay a screen that has already been converted and this conversion has been saved by pressing (Enter), a message `Form contains 8 bit data` is displayed. Do not press (Enter) again, but continue on through the forms file.

5. Compile your forms file as usual.

These conversion utilities are designed to be used once only to update existing data to 8-bit compatibility.

E

Application Guidelines

Currently, six conventional programming languages (SPL, FORTRAN, COBOL II/XL, Pascal, RPG, and BASIC) are supported. General and specific guidelines for each supported programming language are included in this appendix to aid the programmer in language selection for writing a local language or localizable application.

All Programming Languages

General guidelines for all languages supported include the following:

- 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 XL 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.

- Most 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 that 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` *item*=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 input something else (for example, a comma).

- 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/XL local edits. VPLUS/XL edit processing specifications and terminal edit processing statements are separate and are not checked for compatibility. There is no check specified as a terminal local edit consistent with the language-dependent symbol for the decimal point (`DEC TYPE EUR, DEC TYPE US`) in the configuration phase.

COBOL II/XL (HP 32233A)

Specific guidelines for COBOL II/XL, include the following:

- 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` *itemnum*=12. Do not use the COBOL II/XL ALPHABETIC and NUMERIC class tests to determine this (for example, IF data-item IS ALPHABETIC).
- Do not use input-output translation by COBOL II/XL from an EBCDIC character set by means of the ALPHABET-NAME and 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 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 COBOL II/XL.
- 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)

Specific guidelines for FORTRAN include the following:

- Format specifiers `N` and `M` 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 is required.
- Outputting logicals results 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)

Specific guidelines for SPL include the following:

- 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, then determines if 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 entry, and allows processing of 8-bit characters without problems.

F

Example Programs

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.

Sort from 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.
```

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
credt
DEBIT
:EOD

credit
character
DEBIT

END OF PROGRAM
:
```

Sort from 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
:
```

Sort from 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
:
```

Format Date and Time from 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
January
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
:
```

Format Date and Time from 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
January
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
:
```

Scan and Move Character Strings from 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!SP06dLe\1a23%&7"
     TO INSTR1.
     MOVE "a  123&i12fSXgVhklKLabCDASP06i"
     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.
*
*     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 be
*  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 ending condition
*  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
*        uppercase 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.

        33.8       DISPLAY
        33.9       "SCAN/MOVE/DOWNSHIFT WHILE ALPHA:  (EXAMPLE 3-2)".
        34         DISPLAY OUTSTRING.
        34.1
        34.2       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$E!SP06dLe\1a23%&7a 123&i12fSXgVhklKLabCDASP06i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBRCGJGF1F$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
```

Scan and Move Character Strings from 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!SP06dLe\1a23%&7",2;
   MOVE  *  := "a  123&i12fSXgVhklKLabCDASP06i";

   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 for
         uppercase 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$E!SP06dLe\1a23%&7a 123&i12fSXgVhklKLabCDASP06i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBRCGJGF1F$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$E!SP06dLe\1a23%&7a 123&i12fSXgVhklKLabCDASP06i
UPSHIFTED:  (EXAMPLE 1-1)
ABCDFG6IJKASXBVCGJGF1F$E!SP06DLE\1A23%&7A 123&I12FSXGVHKLKLABCDASP06I
UPSHIFTED:  (EXAMPLE 1-2)
ABCDFG6IJKASXBVCGJGF1F$E!SP06DLE\1A23%&7A 123&I12FSXGVHKLKLABCDASP06I
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
:
```

Translate and Relpace Characters from 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 that are nonprintable (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.

```

```
14.7       DISPLAY "IN-STRING:"
14.8       DISPLAY IN-STRING.
14.9       STOP RUN.
```

Compare Character Strings from a COBOLII Program

The example shows a new KSAM/3000 file built programmatically with a language attribute. This means that the keys are 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` is 3, 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\\aaYZZ015iABDYZY005eLDFyxy002BBCdxxx"
 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
:
```

Compare Character Strings from 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 are 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` is `3`, 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 |  >>
               "014hBBeZZZ",
               "011EZqrzyx",
               "001ABCDXXX", << This is the data, to>>
               "007EdCDxyx", << be written to the   >>
               "012IzzAzzz", << KSAM file.          >>
               "015iABDYZY", << The key starts in   >>
               "005eLDFyxy", << column 4 and is 4   >>
               "002BBCdxxx", << characters long.    >>
               "003EaBCXXX",
               "008\\aaYZZ",
               "004eABCYYY",
               "006EabcYYY",
               "009Ayzz",
               "010eaxfxyz",
               "013FGHIzqs";

<< The following DEFINE statement defines the layout of
   the KSAM parameter array; 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 realtional       >>
<< 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
:
```

Obtain Language Information from a COBOLII Program

This program prints the user interface, data manipulation, system default, KSAM/3000 key sequence, VPLUS/3000 forms file, and IMAGE/3000 database 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
:
```

Index