Character Data Representation Architecture

Character Data Representation Architecture (CDRA) is an IBM architecture which defines a set of identifiers, resources and APIs for identifying character data and maintaining data integrity during data conversion processes. The reference publication provides the reader with the framework for the identifiers, descriptions of the supporting resources and APIs, as well as methods for data conversions. CDRA is applicable to country specific data encodings such as the ISO-8859 series as well as large character sets such as Unicode.

Edition Notice: Updated in 2018, this document is a revised, on-line version of the original IBM publication. This version of the document contains many typographical and editorial corrections. A section on the Unicode Encoding Structure has been added in Appendix A. Also, Appendix K has been added to document the CDRA identifiers defined in support of Unicode. Likewise, Appendix J, has been updated with information on how to obtain CDRA conversion table resources on-line. Appendix L was added containing the EBCDIC control code definitions.

Overview

Architecture Definition

Appendices

- Appendix F. Character Sets and Code Pages
- Appendix G. Control Character Mappings
- Appendix H. CDRA and IBM i (Formerly OS/400)
- Appendix I. DFSMS/MVS Considerations
- Appendix J. CDRA Conversion Resources
- Appendix K. CDRA and Unicode
- Appendix L. EBCDIC control character definitions

Resources

- Glossary

Chapter 1. Introduction

This chapter introduces readers to the Character Data Representation Architecture. The architecture objectives, challenges, coverage and concepts are presented to form the basis for understanding the following chapters.

Definition of Character Data Representation Architecture

Character Data Representation Architecture (CDRA) is an IBM* architecture that defines a set of identifiers, services, supporting resources, and conventions to achieve consistent representation, processing, and interchange of graphic character data in data processing environments.

Objectives

The overall objective of CDRA is to define a method of assigning and preserving the meaning and rendering of coded graphic characters through various stages of processing and interchange.

The Objectives intro paragraph is followed by these paragraphs that you have already:

- Define the necessary supporting services (such as tagging) to allow consistent support of various coded graphic character sets, both within and across environments
- Select a minimum number of coded graphic character sets that satisfies most of today's text and data-processing applications and provide the necessary definitions for them so that they can be consistently supported in all applications, services, and devices within and across different environments

Data integrity challenges

Character Data Representation Architecture is used by products and systems to address the following data integrity challenges:

- Proliferation of Different Character Codes

  The primary problem in handling coded character sets is the variety of sets of characters and encoding schemes used to represent them. Technology has increased the variety of applications using computers, and the coded character support for these applications has been provided without an overall strategy.

- Graphic Character Data Recognition

The abilities to properly distinguish graphic character data in a universal manner and to attach a tag to the data are available only in some specific architected environments. The available architected methods are often inconsistent, have lagged behind worldwide requirements, and are constrained by the supporting products.

- Inconsistent or Incomplete Set of Identifiers

  Few applications have consistent character support. Operating system environments rarely provide the necessary services to identify coded character data, leaving this responsibility to the applications.

- Use of Absolute Values (Hard-coding)

  Many applications were designed to operate in specific environments with specific terminal characteristics. The internal representations of frequently used characters have been coded using absolute values. Character data misinterpretation occurs when environment changes are made and the initial character processing functions are no longer valid.

- Non-tagged Data

  Traditional data processing environments were closed systems, and the coded character support was primarily governed by the device character handling capabilities. Data was rarely tagged.

The most visible end-user impact of all these concerns is in data interchange within and across system environments. For example:

- The set of graphic characters supported on the IBM Personal Computer (PC) cannot be fully processed in non-PC environments. (Character set mismatch.)
- When a "dollar symbol" is sent in text from a U.S. mainframe computer to a U.K. mainframe computer, it often appears as a "pound sterling symbol" to the U.K. user. (Conversion based on byte integrity.)
- When a "lowercase a" is sent to a Katakana terminal in Japan, it appears as a "Katakana character" or gets irreversibly converted to an "uppercase A".
- Application and device code page differences may lead to users entering characters from the keyboard that are different from those specified by the application. For example, in Switzerland, the programmer must key "y-acute capital" (Ý) and "dieresis" (¨) instead of "left square bracket" ([ ) and "right square bracket" ( ]), even though the bracket symbols are supported and are engraved on the Swiss keyboard.

- The variety of existing code point conversion tables produces inconsistent and often unpredictable results between different environments. See Figure 1.

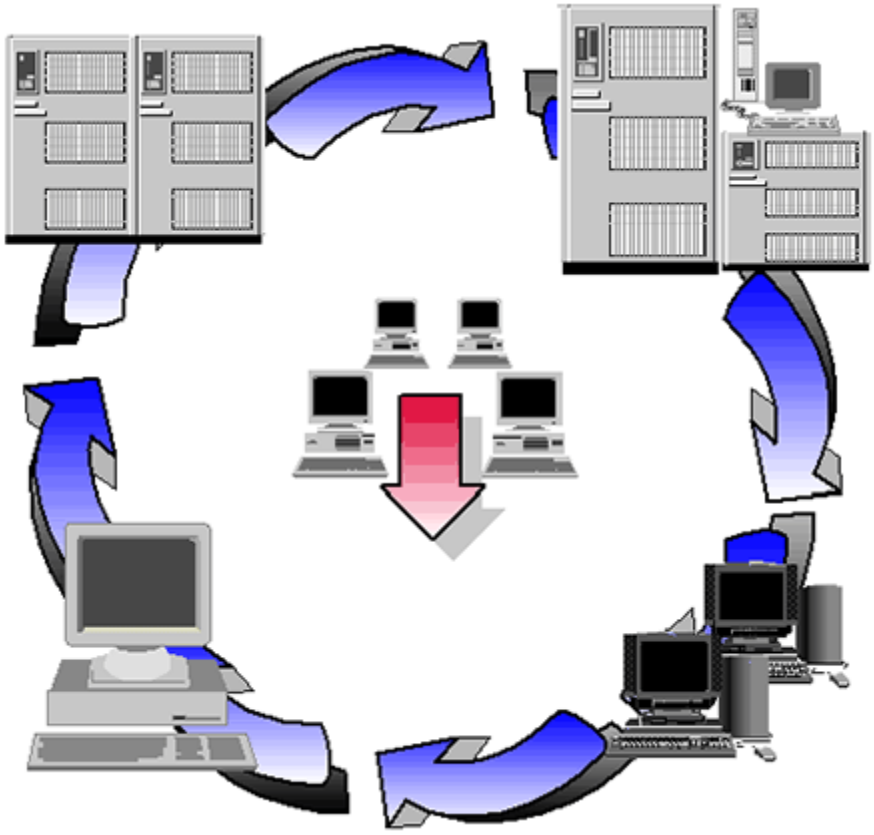


Figure 1. Character data platform domains

A character data domain may be described as an environment in which all character data has the same coded representation. This can be shown in a broad sense with respect to each category of system; midrange, mainframe, workstation and personal. Data domains may be the same within a system, or may differ within a system, but typically the view is one of a character data domain per system category. There are well known examples of the problems encountered as character data moves between data domains, leading to character data misrepresentation.
Businesses may spend a significant portion of their information technology budgets circumventing, repairing, and educating to resolve the data integrity problems.

Coverage of CDRA

Character Data Representation Architecture defines:

- An identification or tagging system to uniquely and reliably identify the representation of graphic character data
- A set of portable Application Programming Interfaces (APIs)
- A set of resources in support of the tags and services
- A set of conventions on the use of the tags and services
- A strategy for coded character set convergence.

This coverage is depicted in Figure 2.

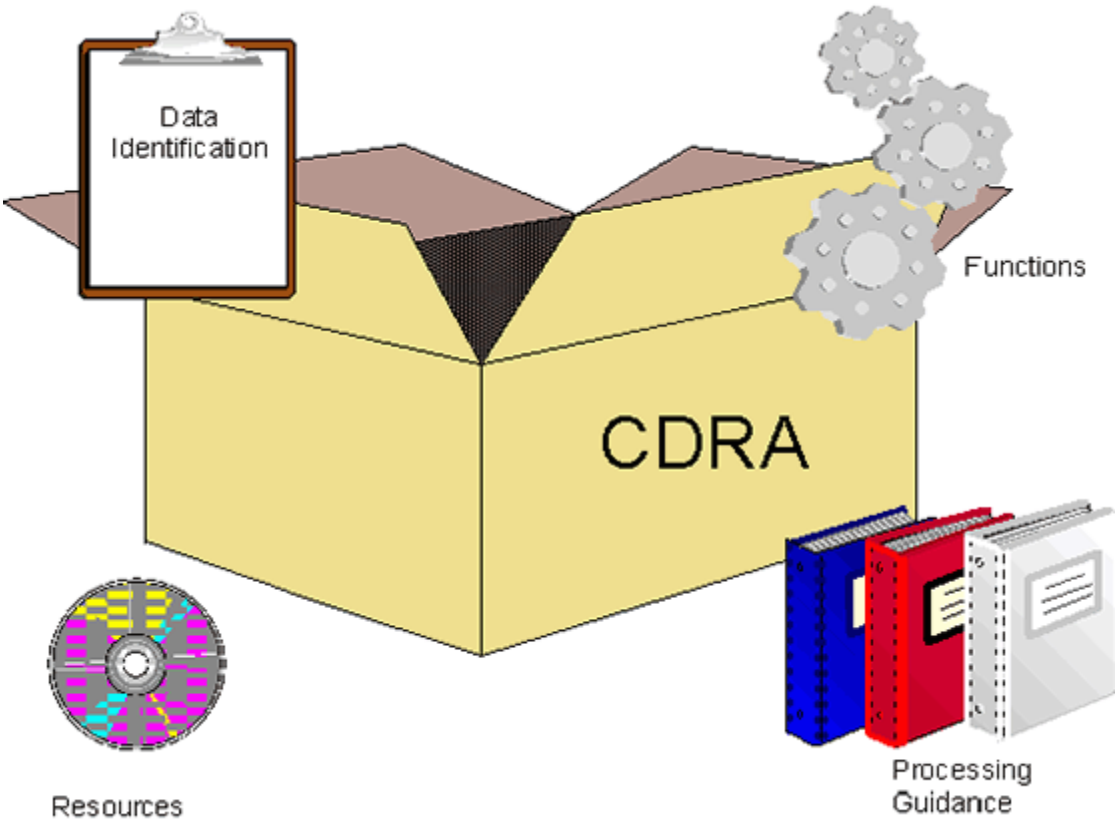


Figure 2. Components of CDRA.

CDRA components are categorized as the data identification mechanism, functions, resources and processing guidelines.

Character data focus

Data can be classified in many ways, such as *character* data, byte strings, integer numbers, or floating-point numbers. Character data is further classifiable into *control character* data and *graphic character* data. Control characters include, for example, Horizontal Tab and Line Feed, which perform specific functions. Graphic characters include uppercase and lowercase letters (with and without accent marks), numeric

digits 0 to 9, ideographs, and other symbols. Graphic character data streams can include embedded code extension controls, such as Shift-Out or Shift-In, used in the interpretation of the data following the controls. Figure 3 shows an example of these classifications.
CDRA deals with character data; primarily with graphic character data, and to a nominal extent with control character data.

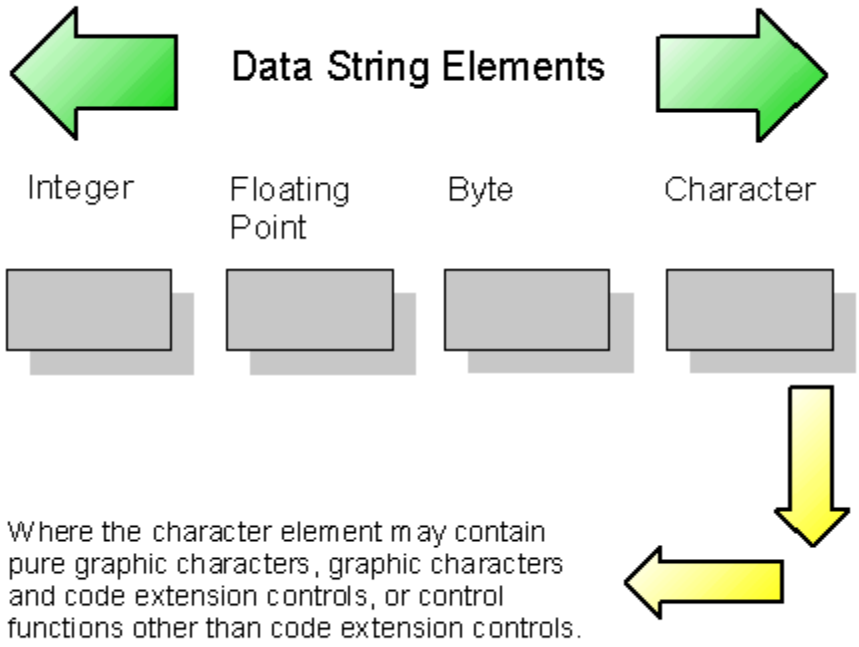


Figure 3. Types of data in a string.

Various types of data may be contained in a data string. CDRA focuses on the coded graphic character data.

Support for control functions

Control functions, as defined by either single control characters or sequences of code points, can appear intermixed with graphic character data. From the CDRA point of view, there are two categories of control functions:

- Code extension
  These functions modify the interpretation of subsequent code points

representing graphic characters. Examples are: Shift-Out (SO), Shift-In (SI), Single-Shift 2 (SS2).
- All other control functions
  Applications or architectures are responsible for handling specific control functions. CDRA provides an interface to query a set of control character encodings and uses the code point assignments for SPACE and SUB in its difference management functions.

CDRA conversion methods and functions support the concept of *string types* to handle space-padded and null-terminated strings. All other aspects of control functions are outside the scope of CDRA.

Architecture concepts

Tagging

Tagging is the primary method to identify the meaning and rendering of coded graphic characters. It is the method by which:

- One or more CDRA identifiers can be associated with a coded graphic character in a data object (such as a file, a database table, or a data stream)
- The graphic character handling capability of a device (such as a display terminal) can be identified or selected
- The graphic character handling capability associated with a piece of processing logic can be identified.

The tag field may be in a data structure that is logically associated with the data object (explicit tagging), or it may be inherited from tag fields associated with other objects or with the computing environment (implicit tagging).

Encoding scheme
Underlying each code used to represent graphic characters is an *encoding scheme*. Encoding scheme definitions specify the coding space (number and allowable values of code points), the allocation of the code space for control and graphic characters, and other characteristics such as the number of bytes per code point and code extension methods permitted in that scheme.

Graphic character integrity
The term *integrity* in CDRA means the preservation of a graphic character's meaning and rendering as identified by its graphic character global identifier (GCGID) or graphic character UCS identifier (GCUID).

Character sets

A character set is a specific collection of characters. There are many character sets in use today and the content of these sets may be quite similar or vastly different.

CDRA recognizes two categories of character sets: *interoperable sets* and *coexistence and migration sets*.

Interoperable sets are the largest character sets for a specific set of languages and countries that:

- Do not contain environment-specific characters
- Do not contain application-unique characters
- Do not contain device-specific characters
- Ensure a high level of processing environment interoperability.

Coexistence and migration sets are those that:

- May contain environment-specific characters
- May contain application-unique characters
- May contain device-specific characters
- May be a subset or superset of an interoperable set
- May not be widely supported.

Services

Services in support of CDRA are collections of functions such as setting and querying of tag values, manipulating tag values, defaulting tag values, or detecting differences in tag values. These services are not architected interfaces defined by CDRA.

CDRA defined services

The CDRA-defined functions have architected call interfaces, called *CDRA Application Programming Interfaces (CDRA APIs)*, that facilitate application code portability across environments. These services are callable using the conventions of any of the supported high-level languages.

Difference management

Difference management is the process of managing different representations of graphic character data. It involves the ability to determine if a difference exists, and to deal with the difference in a predictable and consistent manner.
CDRA describes the general principles of how to manage the representation

differences in coded graphic characters, and the criteria for creating character-data conversion tables. For consistency, a set of default conversion tables and conversion methods have been defined. Further, to minimize the differences and thereby minimize the potential data loss and data corruption problems, CDRA has identified character sets for interoperability.

Resources

Resources are machine representations of definitions associated with CDRA identifiers and supporting data for CDRA services. Collections of such CDRA resources are called CDRA Resource Repositories. The internal representation of the resources is implementation-specific.

Coexistence and migration

Coexistence and migration refers to the current customer environment containing various levels of tagged and non-tagged data, and different levels of application support. CDRA provides the following means by which the current environments can coexist, and at the same time allow for a reasonable migration to a more architected environment:

- Wherever possible, the CDRA-defined Coded Character Set Identifier (CCSID) values are assigned to be the same as the corresponding code page identifiers.
- CDRA has defined CCSIDs for many coded character sets that are currently in use but have not been identified as interoperable. These CCSIDs are called Coexistence and Migration CCSIDs.
- CDRA provides many conversion tables that convert between the Coexistence and Migration CCSIDs and the Interoperable CCSIDs.

Existing tagging methods

Some existing architectures and implementations have provisions for tagging. Some of these recognize code page identifiers (CP) only, while others recognize character set identifiers (CS) and code page identifiers (CP). These identification methods are considered *intermediate forms* of CDRA's long-form identification, which is composed of an encoding scheme, character set and code page pairs, and additional coding-related required information.

Chapter 2. Architecture strategy

This chapter describes the overall strategy used by Character Data Representation Architecture (CDRA) to address the data integrity concerns detailed in Chapter 1, and to provide a solution. Details of the solution are described in the following chapters. This solution can be used wherever graphic character data is handled.

Components of this strategy

The strategy used in CDRA:

- Categorizes and orders the overall problem of different character sets and the associated architectural and development solutions
- Provides a starting base for implementations from which support for other larger sets can be added in a controlled manner
- Recognizes the significant development effort that is needed to overcome the widespread single-byte per character limitation, to address large character sets on a global scale.

The three components of CDRA strategy, Architecture Base, Character Set Groups, and Levels, are shown in Figure 4, and are detailed below.

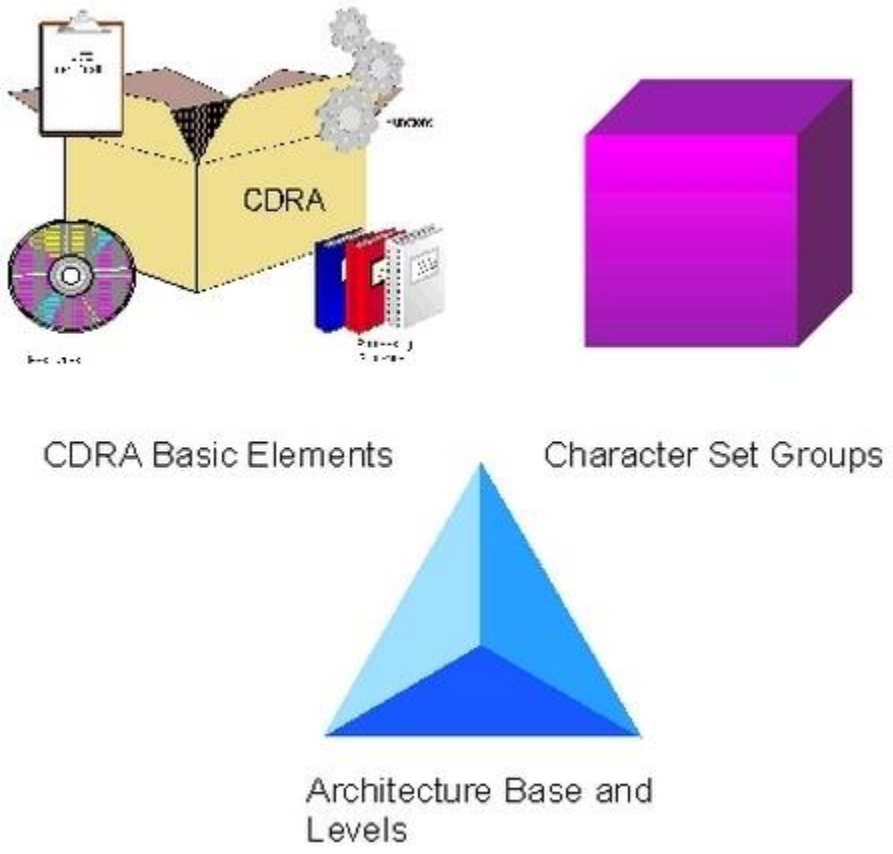


Figure 4. CDRA Strategy

CDRA strategy encompasses the four basic elements of CDRA, the character set groupings and levels of the architecture itself.

Architecture Base The first component of CDRA strategy is the architecture base. This component provides a framework to solve current problems, and can be extended to cover future requirements. It consists of:

- A comprehensive identification system and a set of identifiers for currently used character sets; can be extended as required
- An initial set of services facilitating the use of CDRA identifiers; additional services can be defined as the architecture evolves
- A set of resources required by the services; additional resources can be defined as the architecture evolves
- A set of processing guidelines for functions that are affected by the representation aspects of graphic character data, as an aid to users.

Character Set Groups The second component of CDRA strategy is the concept of character set groups. Graphic character sets used in different countries to support different languages have been grouped into sets with common properties. A selected few of these are defined as Interoperable Character Sets within each group. To reduce the proliferation of graphic character sets and code pages in use, IBM and various standards organizations have collected and classified commonly used graphic characters into a few specific sets. Each of these sets has the following characteristics:

- It is a superset of many existing smaller graphic character sets
- It contains a base set of graphic characters required in a group of countries or in a group of national languages having some common characteristics
- It can be used in a broad range of common applications
- It permits preservation of graphic character integrity for interworking applications within a specific group of countries that use the set
- It is the target for convergence and migration in each country or group of countries.

Special graphic character sets supporting specific applications (such as APL, scientific word processing, or desktop publishing) are treated as extensions to the base sets. Each graphic character set in all countries, with a few exceptions, contains a common set of graphic characters: the uppercase English letters A to Z, the lowercase English letters a to z, (4) the numerals 0 to 9, and 19 miscellaneous symbols. See Figure 45 in Appendix A for a complete list. The implications of supporting character set groups differ in the types of services and resources needed for each group. Character set groups are shown in Figure 5, and are described in the following sections.

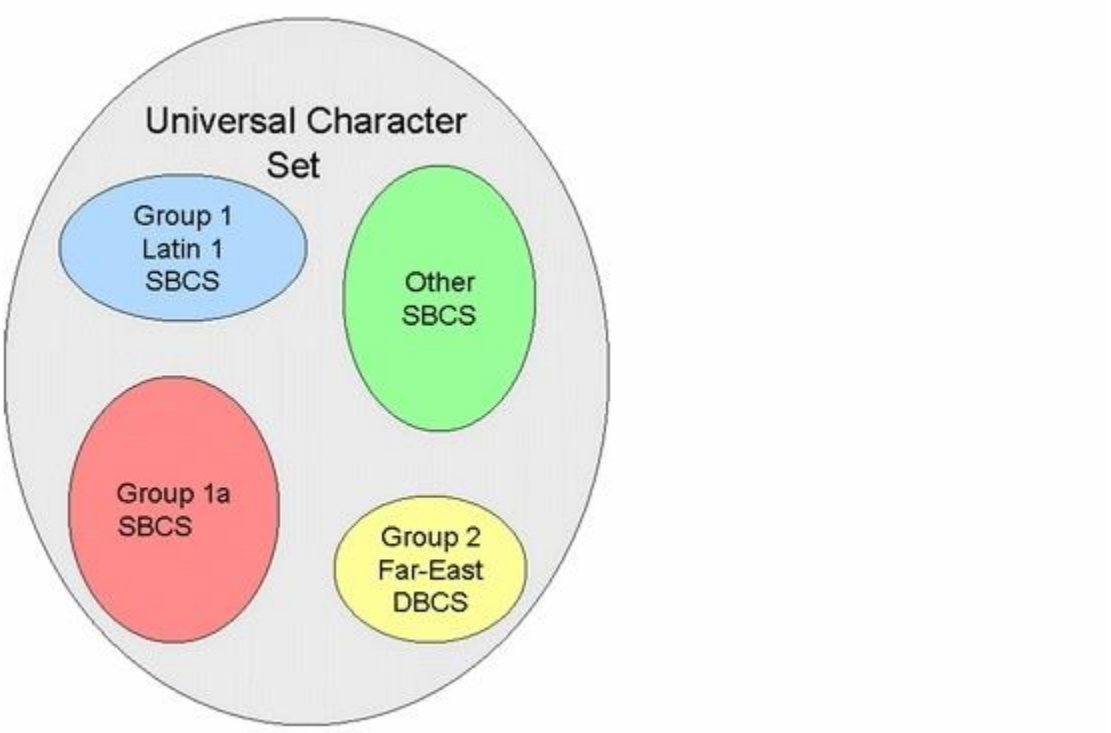


Figure 5. CDRA's Character Set Groupings

Commonly Used Character Sets

- Group 1 uses the Latin Alphabet Number 1 character set, which is represented using single-byte encodings (SBCS). It satisfies the basic graphic character requirements in the following geographic regions:
    - Americas:
      Canada
      Latin America: (Argentina, Belize, Bolivia, Brazil, Chile, Colombia, Costa Rica, Cuba, Ecuador, Guatemala, Guyana, Honduras, Mexico, Nicaragua, Panama, Paraguay, Peru, El Salvador, Surinam, Uruguay, and Venezuela)
      United States of America
    - Western Europe:
      Austria, Belgium, Denmark, Faroe Islands, Finland, France, Germany, Iceland, Italy, Liechtenstein, Luxemburg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, and United Kingdom
    - Other Parts of the World:
      Australia, Hong Kong (Latin), and New Zealand

- Group 1a consists of several different graphic character sets that can be represented using single-byte codes. These sets, called subgroups, include non-Latin characters and some Latin characters that are not included in Latin Alphabet Number 1. Each set is used in the following geographic regions:
    - Arabic Scripts
      Arabic: (Bahrain, Egypt, Iraq, Jordan, Kuwait, Lebanon, Libya, Oman, Qatar, Saudi Arabia, Sudan, Syria, United Arab Emirates, and Yemen)
      Urdu: Pakistan
      Farsi: Iran
      Arabic-French: Maghreb Region -- Algeria, Morocco, and Tunisia.
    - Latin Alphabet Number 2
      Albania, Czech Republic, Slovakia, Hungary, Poland, Romania, and the following republics of the former Yugoslavia: Croatia, Slovenia, and the Muslim-Croat part of Bosnia/Hercegovina.
    - Cyrillic
    - Eastern Europe
      Bulgaria and the following republics of the former Yugoslavia: the "Former Yugoslav Republic of Macedonia", Serbia/Montenegro, and the Serbian part of Bosnia/Hercegovina.
    - Russian
      Russia (and several other countries of the former Soviet Union)
    - Greek (Greece)
    - Turkish (Turkey)
    - Hebrew (Israel)
- Group 2 covers the geographic regions in the Far East. These graphic character sets are represented using multi-byte encoding (DBCS or MBCS). The following subgroups used in different countries are included in this group:
    - Japanese
    - Korean
    - Simplified Chinese (People's Republic of China, PRC)
    - Traditional Chinese (Republic of China, ROC)
    - Thai (7) (Thailand, device character sets).

  The following are single-byte sets (8) that have the characteristics of Group 1a:

    - Katakana (Japan), Korean (Korea, small set), Simplified Chinese (PRC, small set), and Thai (Thailand, processing and interchange set). The Katakana, Korean, and Simplified Chinese single-byte character sets are often used along with corresponding larger double-byte character sets, where a mixed single-byte and double-byte coding is used. Also, Traditional Chinese (ROC, small set) uses a subset of Group 1 characters.

- Group Universal covers all the 'large' character sets used in supporting Unicode and ISO-10646.

Universal character set

The IT industry largely supports the Universal Coded Character Set, known as Unicode. CDRA supports Unicode as a defined character set encoding. Unicode is a superset of the many earlier country or language specific character sets. The character repertoire of Unicode, developed by the Unicode Consortium is kept in synch with the ISO standard, ISO/IEC 10646, Information Technology-Universal Coded Character Set (UCS). This character set is applicable to presentation, processing, storage, transmission, interchange and representation of all of the world's written forms of language and symbols. UCS assigns a unique number to every character in all the living and archaic scripts and several symbols used in various application domains. The architecture itself:

- specifies an encoding space of 17 x 2^16 numbers or code points;
- describes the general structure of the coding space;
- defines the structure of the coding space as seventeen planes, numbered from 0 to 16, each containing 2^16 code points.
- specifies the coded representations for control functions;
- assigns characters to code points in the different planes; each assigned character is given a unique normative character name and a representative glyph.
- reserves ranges of code points for Private Use and for other specified purposes.
- specifies three basic encoding forms of encoding, a 4-byte format known as UCS-4 or UTF-32, a variable two or four-byte format known as UTF-16, and a variable number of bytes (one to four) format known as UTF-8. A subset of UTF-16 containing only the two-byte subset was called UCS-2 and is now deprecated in the standard.
- specifies how future additions to the coded character set will be managed

Earlier editions of ISO/IEC 10646 standard had defined ranges of code points called 'zones' in the BMP. These have now been removed from the standard. The standard also had definitions for 'levels of implementation' addressing ability to deal with 'combined character sequences' - these levels have also been removed from the standard.

The Unicode standard, in addition to being kept identical with ISO/IEC 10646 for the character set and assignments, defines properties and additional specifications on how to use these properties during text processing.

For detailed information on how CDRA handles Unicode see Appendix K, CDRA and Unicode.

CDRA Levels

The third component of the CDRA strategy shown in Figure 4 is the concept of Levels. Levels are used to distinguish between specific sets of available elements from the architecture base, as the architecture and the supporting implementations evolve over time. The relationship between the levels has been depicted in the diagram shown in Figure 6. Level 1 provided the initial seed of CDRA, which was substantially extended with the release of Level 2. The growth in Level 2, noted as extensions in the diagram, has been more of a series of enhancements rather than the pronounced type of change that was seen from Level 1 to Level 2.

CDRA Level 1

CDRA Level 1 defined an initial set of elements from the architecture base. It consisted of:

- A comprehensive identification system
- A set of CDRA identifiers for commonly used character sets
- A subset of these CDRA identifiers specifically for interoperability, to assist customers in identifying the strategic direction for coded character sets
- Generic concepts of tagging and difference management.

Character Data Representation Architecture - Registry, SC09-1391 contained the following:

- A set of graphic character data conversion tables for selected pairs of identifiers
- The principles used in creating these tables, and the specific mismatch management criteria associated with each.

CDRA Level 1 addressed all the commonly used character sets within:

- Latin Alphabet Number 1 in Group 1
- Single-byte graphic character sets in Group 1a
- Single- and double-byte graphic character sets in Group 2.

CDRA Level 1 satisfied these objectives:

- To achieve consistent character data processing and interworking between different systems or system components *within a country or within a specific group of countries* having a common character set. The graphic character sets within each country will be those that apply across a wide range of basic applications.
- To allow *coexistence* between countries or groups of countries with different character sets. Interchange of data for storage and retrieval purposes or for data pass-through will be possible.
- To allow *limited* interworking with systems outside the country or group of countries. The extent of correct interworking between two countries will be limited to the subset of characters that is common between the two character set groups or subgroups.

CDRA is primarily concerned with the coded graphic character set boundaries within and between different groups, rather than with political or geographical boundaries. However, these different types of boundaries are indirectly related to each other through the requirements for resources such as fonts, keyboards, and conversion tables.

CDRA Level 2

CDRA Level 2 included all Level 1 elements. In addition, it included definitions of functions called CDRA-Defined Services, along with the syntax for accessing these functions. These APIs were designed to be callable from any supported high-level language. Several CDRA resources were needed to support the functions defined in Level 2. Level 2 included descriptions of the elements of those resources and some general principles for managing them. The resource data structures and the resource maintenance functions are implementation-specific.

Extensions

CDRA extensions now include support for:

- Encoding scheme, character set, and code page identifiers for Extended UNIX Code (EUC), Transmission Control Protocol (TCP), and Universal Multiple-Octet Coded Character Set (UCS)
- New conversion tables
- New conversion method definitions
- New ESID definitions
- New CCSIDs registered

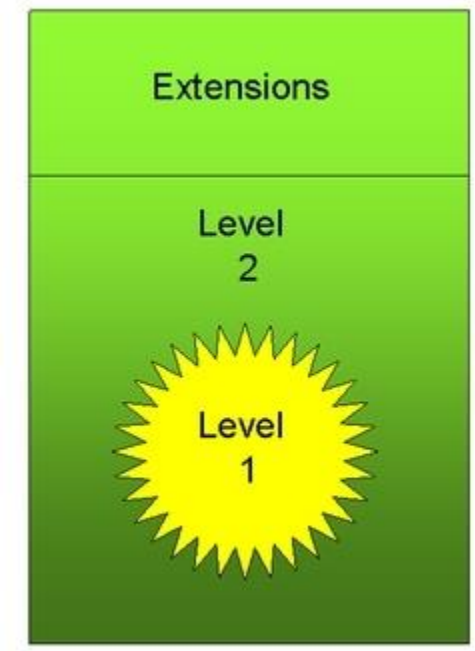


Figure 6. Architecture levels

Chapter 3. CDRA Identifiers

Character Data Representation Architecture deals primarily with graphic character data, and to a lesser extent with control character data. Graphic character data can include imbedded code extension controls that influence the interpretation of the data that follows. This chapter defines several identifiers related to graphic-character data representation, and what it means to tag with these identifiers.

The following identifiers are defined:

- Graphic Character Global Identifier (GCGID) and Graphic Character UCS Identifier (GCUID)
- Encoding Scheme Identifier (ESID)
- Coded Graphic Character Set Global Identifier (CGCSGID), consisting of:
    - Graphic Character Set Global Identifier (GCSGID)
    - Code Page Global Identifier (CPGID)
- Additional Coding-related Required Information (ACRI)
- Coded Character Set Identifier (CCSID).

These identifiers form the architectural basis for unique identification and interpretation of coded graphic character data.

Coding of Graphic Character Data

Character data is represented in machines as code points, consisting of one or more 7-bit bytes (septets) or 8-bit bytes (octets) of data. Underlying each code is an encoding scheme. In the terminology of coded character set standards, a code is a system of bit patterns to which a specific graphic or control meaning has been assigned. Each unique bit pattern defined by a code is called a code point. CDRA identifiers provide the ability to unambiguously determine the graphic character associated with a code point.

Elements of Character Data Representation

The identifiers associated with graphic character representation (see Figure 7) are:

- Graphic Character Global Identifier and Graphic Character UCS Identifier
- Long-form identification -- a set of identifiers consisting of:
    - Encoding Scheme Identifier
    - One or more Coded Graphic Character Set Global Identifiers, each of which is a concatenation of:

A Graphic Character Set Global Identifier
A Code Page Global Identifier.
    - Additional Coding-related Required Information, as specified by the Encoding Scheme Identifier (for example, a list of valid first bytes of double-byte code points, code points used for code extensions, or a set of floating accents and a valid associated character).
- Short-form identification -- an identifier called Coded Character Set Identifier (CCSID) is defined as an alternative to the variable-length long form.

These identifiers are detailed in the following sections.

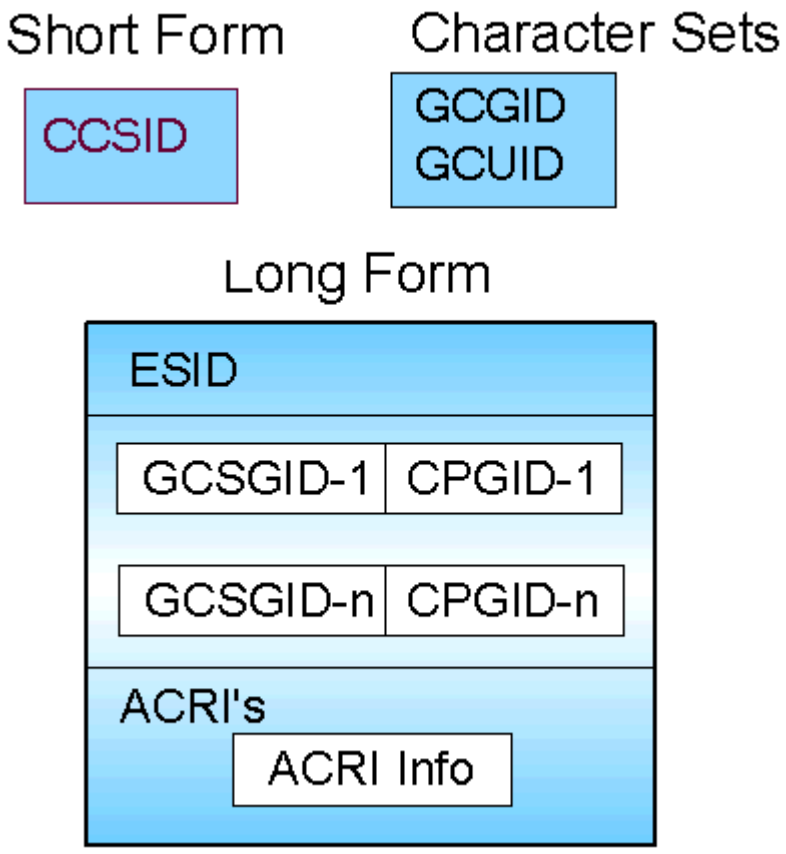


Figure 7. CDRA Identifier Forms

Graphic Character Global Identifier

IBM has an established system to uniquely and uniformly identify and name graphic characters. A graphic character global identifier (GCGID)or a graphic character UCS identifier (GCUID) is used to convey the meaning of a graphic character in a code-independent manner. They are used primarily in the representation of characters in

objects such as Character Set or Code Page resources. They are not used to tag the data directly.

A GCGID is a 4- to 8-character alphanumeric identifier assigned to a graphic character. A GCUID is an 8-character identifier of the form Unnnnnnn where nnnnnnn is a 7-digit hexadecimal value.

Each graphic character that is to be assigned a code point must have a GCGID or a GCUID. They are used wherever a graphic character is referenced in a code-independent manner. They are also the basis of establishing correspondences between code points in different representations.

Graphic Character Global Identifier (GCGID).
The GCGID identifies graphic characters defined by IBM. The GCGID definition uniformly associates arbitrary graphic character shape with an eight-character identifier GCGID. It also provides a short description for each GCGID.

Graphic Character UCS Identifier (GCUID).
The GCUID format is for defining additional characters and sets of characters that (mostly) exist in the Universal Coded Character Set (UCS) defined in ISO/IEC 10646 and Unicode standards and need to be used in IBM resource definitions such as IBM code pages. The format allows all current and future characters from UCS planes 0 through 16 to be described. It also allows for identifying characters and glyphs that are not defined in UCS as well as glyph variants of the unified Han area of UCS.

For those characters that exist in the UCS, the standardized graphic character name is used as a description for the GCUID.

Both GCGID and GCUID identifiers are compatible with each other since they share the same basic format and provide identifiers that are globally unique. Unless there is a special need to differentiate between GCGID and GCUID, both may be used interchangeably.

The GCGID and GCUID are identifiers from two different systems to identify individual members of the total collection of all characters. Since the two identifiers can point to the same character, this system also establishes an equivalence for some of them

In CDRA, the terms graphic character identifier, character identifier, meaning of graphic character, and rendering of graphic character are synonymous with GCGID.

IBM's GCGID system provides for distinguishing between two renderings of a graphic character. When no specific rendering is indicated, a "nominal" rendering is assumed

with the character. Specific renderings can further be specified using another identifier such as Font Global Identifier, FGID.

The rendering part of a GCGID is of significance primarily for presentation processing such as formatting, displaying, or printing. GCGIDs with different renderings typically appear in character sets and code pages that are primarily presentation-oriented. However, some of these character sets and code pages are used for all aspects of processing of graphic characters. If they are encountered by functions such as comparison, depending on the context of use, graphic characters with two different renderings may be equated.

When graphic characters with two different renderings are part of a character set and are included in the same coded character set, different code points are assigned different GCGIDs to represent the different renderings. Some examples of use of GCGIDs for graphic characters with different renderings are:

- The different shapes of Arabic characters coded from CS 00235 in CP 00420 or CP 00864
- The wide Latin alphabets (such as A through Z) of Far East double-byte code pages (such as CP 00300), to distinguish them from "nominal" width A through Z in single-byte code pages (such as CP 000290). CP 00290 and CP 00300 are used together in mixed single-byte and double-byte codes (such as CCSID 05026).

Long-Form Identification

The long-form identification consists of an Encoding Scheme Identifier, one or more Coded Graphic Character Set Global Identifiers (each consisting of a Graphic Character Set Global Identifier and a Code Page Global Identifier), and any Additional Coding-related Required Information that is required to complete the specification of the representation.

Encoding Scheme Identifier
The Encoding Scheme Identifier, ESID, is a 4-digit hexadecimal number that identifies the scheme used to code graphic character data. The following 3 elements have been used where possible in ESID definitions.

The basic encoding structure (x)
This element identifies the basic structural characteristic that differentiates various encoding schemes such as EBCDIC, ISO-8, IBM-PC Data, or others.
The number of bytes per code point (y)

When the encoding scheme permits a different number of 7-bit or 8-bit bytes per code point, this element identifies the selection used.
The code extension method (zz)
Code extensions are techniques used to encode more characters than can be accommodated in the basic encoding structure. An example is the use of SO (Shift-Out) and SI (Shift-In) as controls to access an alternative assignment of graphic characters to code points, and to show whether one byte or two bytes of the data constitute a code point, in the EBCDIC mixed single-byte and double-byte encoding. This element of the ESID identifies the method of code extension used from among the many that may be allowed in the encoding scheme.

Note to developers: While efforts have been made to define ESIDs using these elements, not all ESIDs follow the above pattern. It is essential that all encoding scheme identifiers be defined by the owner of CDRA prior to being used.

Figure 8 shows the three components of the ESID. The component values and their meanings are detailed in the following sections.

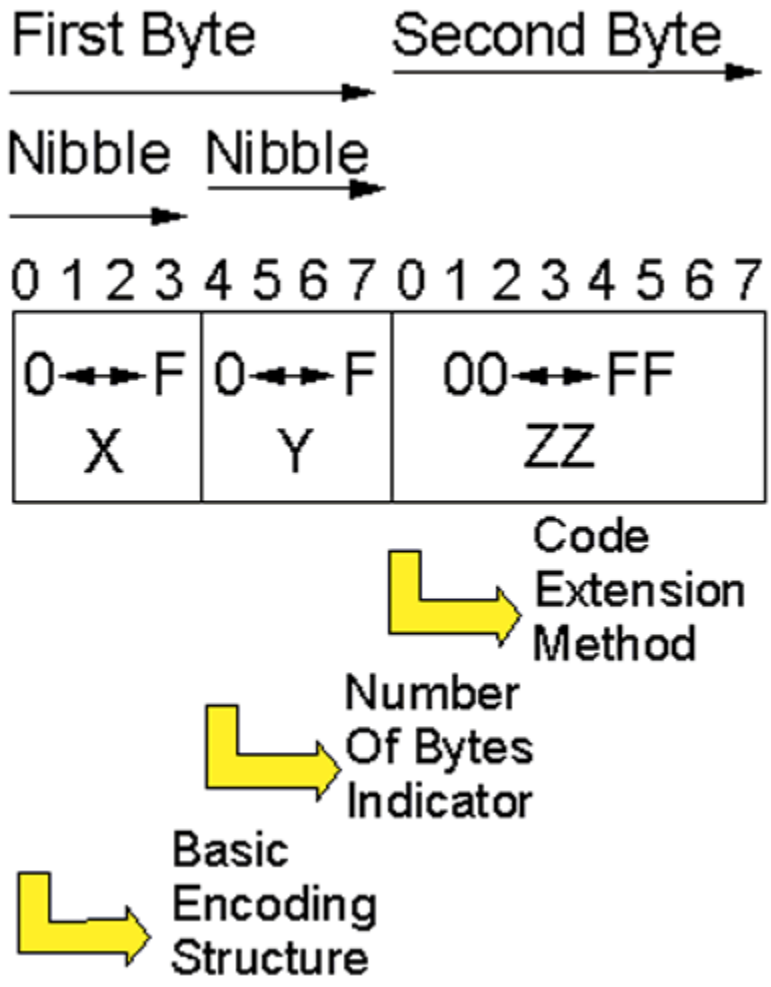


Figure 8. Encoding Scheme Identifier Format

The ESID makes the following possible:

- The selection of the correct algorithms (such as parsing) to be invoked to process graphic character data.
- Identification of reserved code point(s) for allocation to some most-frequently used characters such as SPACE (GCGID SP010000).

The ESID also determines the number and types of other CDRA identifiers needed in the long form.

The term Encoding Scheme (ES) is synonymous with ESID.

Basic Encoding Structure (x)

The following values are defined for the first nibble (10) of the ESID to identify the structure. The properties of each structure are detailed in Appendix A. Encoding Schemes.

Hex	Structure
0	Defaults to higher level in hierarchy
1	EBCDIC
2	IBM-PC Data
3	IBM-PC Display
4	ISO 8
5	ISO 7
6	EBCDIC presentation
7	UCS
8	UCS Display
9	8 bit, for a standalone, 7-bit EUC G-set that has been shifted into the right half of the encoding space
A-C	Reserved for future allocation by CDRA
D	Unique encoding. Details of the encoding structure are found in the related CP and CCSID definitions.
E	Reserved for extending ES id, when needed
F	For Private Use. Use of this value must be accompanied by a specification of the structure, and the rules for usage with specific values of the other parts of the ESID. Definition of the Private Use values is outside the scope of CDRA.

Number of Bytes Indicator (y)

An encoding scheme may permit specific variations in the number of bytes associated with a code point (for example, EBCDIC single-byte versus EBCDIC double-byte). These variations are shown using the second nibble of the ESID. The value of this nibble is not the number of bytes per code point, but rather a pointer to the definition. The value does not equate to the number of bytes in the code point. The values representing a variable number of bytes identify what is allowed to appear in a string, not what actually appears. The encoding scheme defines permitted values of this nibble for the encoding structure used.

If the value of the first nibble defining the basic encoding structure element is zero, the second nibble identifying the number of bytes must also be zero.

The following values are defined:

Hex	Number of Bytes per Code Point
0	Reserved for use with zero value for the basic encoding structure
1	Fixed single-byte, SBCS
2	Fixed double-byte, DBCS (including ISO/IEC 10646-1 UCS-2)
3	IBM Far East style, mixed single-byte and double-byte
4	ISO 2022 schemes (EUC, TCP/IP)
5	UCS-4 or UTF-32
6	Reserved for future allocation by CDRA
7	Fixed triple-byte
8	UTF-n variable number of bytes, self describing (37)
9	Fixed 4-byte
A	Mixed 1-byte, 2-byte, 4-byte (for GB 18030)
B	BOCU-1, SCSU and similar Stateful Compression Schemes
C-E	Reserved for future allocation by CDRA
F	For Private Use. The specification of Private Use must include the values (and the specific meaning) of the encoding structure nibble with which it can be used. Definition of the Private Use values is outside the scope of CDRA.

Code Extension Method (zz)

The code extension method is described by the second byte of the ES identifier. This byte indicates that a code point from an extended coded character set may appear in the data; it does not mean that the extension method has actually been used in a specific character string.

When the first two nibbles of the ESID are zeros, the code extension byte value must be zero.

The following values are defined:

Hex	Code Extension Method	Hex	Code Extension Method
00	No extensions are specified	0C	Unicode Standard Code Compression Scheme
01	Locking Shifts (SO and SI, or LS1 and LS0 (11) or UC and LC locking controls)	0D	Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)
02	Reserved for future allocation by CDRA	0E	Binary Ordered Compression for Unicode (BOCU-1)
03	IBM EUC scheme (ISO-2022-based)	0F	UCS with Byte Order Mark (BOM) to indicate Endianness; BE is assumed in absence of BOM

04	TCP/IP scheme (ISO-2022-based)	10 to 49	Reserved for future allocation by CDRA
05	ISO-8 with possible graphics in C1 area (X'80' to X'9F')	50	ISO-7 with possible graphics in the C0 area (X'00'to X'1F')
06	Reserved for future allocation by CDRA	51 to 54	Reserved for future allocation by CDRA
07	UTF-8 Universal Transformation Format	55	ISO-8 with possible graphics in C0 and C1 areas (X'00' to X'1F' and X'80' to X'9F')
08	UTF-EBCDIC Universal Transformation Format	56 to FD	Reserved for future allocation by CDRA
09	Used for an individual Unicode plane	FE	Reserved for Private Use of Code Extension. Definition of the Private Use value is outside the scope of CDRA.
0A	Reserved for future allocation by CDRA	FF	Code Extension consideration does not apply.
0B	Used to indicate Little Endian Order for UCS		

Code Extension States

When an encoding scheme uses an extension technique, it uses more than one elementary coded character set to create a composite coded character set. The scheme specifies one code extension switching state for each coded character set used. While in a given state, the associated coded character set is used for representing and interpreting the character data. The method for switching between these states can be implicit or explicit, locking or single shifting. The number of switching states and the method of switching between the states in a coded character set are specified by the encoding scheme. State numbering begins at 1 and increases by 1 for each coded character set. For example, in mixed single-byte, double-byte encodings there are 2 states; the single-byte coded character set is state 1 and the double-byte coded character set is state 2. Encoding schemes which define a single coded character set have a single state; state 1.

The second nibble and the last byte of the ESID together identify the number of switching states. The last byte of the ESID identifies the switching method employed in an encoding scheme. The first and second nibbles identify the nature of the elementary code structures used in the resulting composite structure.

ESID Values

ESID values and their semantics are listed in Figure 9.

ESID hex	Interpretation
1100	EBCDIC, SBCS, No code extension is allowed Number of States = 1.
2100	IBM-PC Data, SBCS, No code extension is allowed. Number of States = 1.
3100	IBM-PC Display, SBCS, No code extension is allowed. Number of States = 1.
4100	ISO 8, SBCS, No code extension is allowed. Number of States = 1.
4105	ISO 8 (ASCII code), SBCS, Graphics in C1 Note that graphic characters may be present in the area normally reserved for the C1 control codes. (ie X'80' to X'9F') Number of States = 1.
4155	ISO 8 Presentation (ASCII code), SBCS, Graphics in C0 and C1. Number of States = 1.
5100	ISO 7 (ASCII code), SBCS, No code extension is allowed. Number of States = 1.
5150	ISO 7 Presentation (ASCII code), SBCS, Graphics in C0. Number of States = 1.
6100	EBCDIC Presentation, SBCS, No code extension is allowed. Number of States = 1.
8100	8 bit, SBCS, used with a 7-bit code page, characters are shifted into the right hand side of the encoding space, used only for single-byte EUC G-sets when each G-set is treated as a standalone code. Number of States = 1.
D100	PTTC/BCDIC – 6 bit encoding, no code extension is allowed. Number of States = 1.
D101	Paper Tape Transmission Code (PTTC), 6 bit encoding, uppercase/lowercase support using UC/LC code extension method. Number of States = 2.
1200	EBCDIC, DBCS, No code extension is allowed. Number of States = 1.
2200	IBM-PC Data, DBCS, No code extension is allowed. Number of States = 1.
3200	IBM-PC Display, DBCS, No code extension is allowed. Number of States = 1.
5200	ISO 7 (ASCII code), DBCS, No code extension is allowed. Number of States = 1.
6200	EBCDIC Double-byte Presentation Number of States = 1.
7200	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, Big Endian order. No code extension is allowed. Number of States = 1.
7209	Unicode pure double-byte. Used for any standalone, individual Unicode plane. Number of States = 1.
720B	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, Little Endian order. No code extension is allowed. Number of States = 1.
720F	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, endianness is determined by byte order mark (BOM), assumed to be Big Endian in absence of BOM. No code extension is allowed Number of States = 1.
8200	Unicode Display Number of States = 1.
9200	8 bit, DBCS, used with a 7-bit code page, characters are shifted into the right hand side of the encoding space, used only for double-byte EUC G-sets when each G-set is treated as a standalone code. Number of States = 1.
1301	EBCDIC, Mixed single-byte and double-byte, using SO/SI code extension method. Number of States = 2.

2300	IBM-PC Data, Mixed single-byte and double-byte, with implicit code extension. Number of States = 2.
2305	PC Data, Mixed single-byte and double-byte, with implicit code extension, single-byte is Windows encoding. Number of States = 2.
3300	IBM-PC Display, Mixed single-byte and double-byte, with implicit code extension. Number of States = 2.
4403	IBM EUC Number of States = 2-4.
5404	ISO 2022 TCP/IP using ESC sequences to designate code sets to G0. Number of States = 2-4.
5409	ISO 2022 TCP/IP using SO/SI Number of States = 2.
540A	ISO 2022 TCP/IP using SO, SI, SS2, and SS3. Number of States = 3-4.
7500	Unicode UTF-32, Big Endian order. No code extension is allowed. Number of States = 1.
750B	Unicode UTF-32, Little Endian order. No code extension is allowed. Number of States = 1.
750F	Unicode UTF-32, endianness is determined by byte order mark (BOM), assumed to be Big Endian in absence of BOM. No code extension is allowed Number of States = 1.
5700	ISO 7 Triple-byte Code Set, No code extension is allowed. Number of States = 1.
1808	UTF-EBCDIC, as defined in Unicode Technical Repot 16. Number of States = 1.
7807	UTF-8, UCS-2 transform, No code extension is allowed. Number of States = 1.
780D	Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8), as defined in Unicode Technical Report #26 . Number of States = 1.
2900	PC Data, fixed 4-byte Number of States = 1.
2A00	PC Data, mixed single-, double- and four-byte (Note: IBM PC or Windows code pages may be used as the single-byte component of a CCSID using this ESID.) Number of States = 3.
7B0C	Standard Compression Scheme for Unicode (SCSU) as defined in Unicode Technical Standard 6.
7B0E	Binary Ordered Compression for Unicode (BOCU-1) as defined in Unicode Technical Note 6.
Fxxx	Private Use. User-defined encoding scheme.
xFxx	Private Use. User-defined encoding scheme.
xxFE	Private Use. User-defined encoding scheme.

Figure 9. ESID values

Coded Graphic Character Set Global Identifier

The Coded Graphic Character Set Global Identifier, CGCSGID, is a ten-digit decimal number representing the concatenation of the Graphic Character Set Global Identifier (GCSGID) followed by the Code Page Global Identifier (CPGID). GCSGID and CPGID are described in the following sections. CGCSGID identifies a specific collection of graphic characters and their assigned code points using an encoding scheme.

Many architectures and supporting implementations, such as Document Interchange Architecture (DIA), have traditionally supported the CGCSGID. It has been assumed that the encoding scheme information can always be reliably derived from the code page identifier alone, but this assumption is not true for many registered PC code pages. It will also be invalid if schemes such as the mixed single-byte and double-byte encodings used by the IBM PCs in the Far East have to be represented.

The term GCID, used in some IBM architectures, is synonymous with CGCSGID.

Graphic Character Set Global Identifier

A Graphic Character Set Global Identifier, GCSGID, is a 5-digit decimal identifier assigned to a collection of characters that is to be processed as an entity (a Graphic Character Set). It uniquely identifies a specific collection of GCGIDs that are valid in the set.

The range of GCSGID values is 00001 (X'0001') to 65534 (X'FFFE'). The values X'FE00' to X'FEFF' are reserved for Request for Price Quotation (RPQ) use by IBM products. The values X'FF00' to X'FFFE' are reserved for customer use.

A GCSGID is assigned to every Registered Graphic Character Set by IBM (or by a customer organization).

See Special-Purpose Values for GCSGID and CPGID for use of 00000 (X'0000'), 65,535 (X'FFFF') and other special-purpose values for GCSGID.

The term Character Set (CS) is synonymous with GCSGID.

SPACE as a special character

By itself, the GCSGID does not specify either the inclusion or the exclusion of the SPACE (GCGID = SP010000) character. Each encoding scheme reserves one or more code points for allocation to the SPACE character. There are two possible code points for it when using mixed SBCS and DBCS encoding schemes.

Code Page Global Identifier

A Code Page Global Identifier, CPGID, is a 5-digit decimal number assigned to a code page.

A code page is a specification of code points from a defined encoding structure for each graphic character in a collection of one or more graphic character sets.

A CPGID identifies a unique assignment of the graphic code points in an encoding scheme to a specific set of GCGIDs. Many character sets may be contained in a code page. When all of the code points in the graphic encoding space of a code page have been assigned, then the character set containing this collection of GCGIDs is defined to be full. Often, when a code page is first created and registered, some of the assignable graphic code points may not have assigned GCGIDs. The character set containing these assigned characters is defined to be maximal. As more code point assignments are made, the maximal character set will change. Once all code points have been assigned, the maximal set will be the full set.

A CPGID is assigned to every Registered Code Page by IBM. In some cases, the same CPGIDs have been used when the encoding structures are similar.

The range of CPGID values is 00001 (X'0001') to 65534 (X'FFFE'). The values X'FE00' to X'FEFF' are reserved for Request for Price Quotation (RPQ) use by IBM products. The values X'FF00' to X'FFFE' are reserved for customer use.

The term Code Page (CP) is synonymous with CPGID.

Special-Purpose Values for GCSGID and CPGID

IBM standards reserve the values X'0000' and X'FFFF' for future assignments. In practice, these identifier values have been used for a number of different special purposes. Some values other than X'0000' and X'FFFF' that have been reserved for special-purpose use are also included in this section. In the interest of providing consistency between various implementations, the semantics of use of these values, either in current use or for future use, are defined here.

Some known definitions are listed below, along with their semantics. Others will be added as they become known to CDRA.

The CS value of X'0000' is used in several IBM architectures, such as Formatted Data Object Content Architecture (FD:OCA), Mixed Object Document Content Architecture (MO:DCA), and Document Interchange Architecture (DIA) Profiles, to facilitate migration and coexistence between the use of only a CGCSGID (CS, CP pair) prior to

the advent of CDRA and the use of the CCSID identifier in different architecture definitions.

In these architectures, if the CS portion of a structured field carrying a CGCSGID has a value of X'0000', the value of the CP portion is interpreted as a CCSID. The following definitions then apply:

1. CS X'0000' with CP X'0000'
   The CP value of X'0000' is interpreted as CCSID X'0000'. This CCSID value means that the tag value is to be inherited from a higher level in a hierarchical structure.
2. CS X'0000' with CP X'FFFE'
   The CP value of X'FFFE' is interpreted as CCSID X'FFFE'. This CCSID value means that the tag value is to be obtained from a lower level in a hierarchical structure.
3. CS X'0000' with CP X'FFFF'
   The CP value of X'FFFF' is interpreted as CCSID X'FFFF'. This CCSID value means that the tagged data is to be interpreted as "not graphic character data" or "actual representation is unknown".
4. CS X'0000' with all other CP values
   The CP value is interpreted as a CCSID.

The CS value of X'FFFF' can have the following special-purpose definitions.

1. CS X'FFFF' with CP X'0000'
   Reserved for future definition in CDRA.
2. CS X'FFFF' with CP X'FFFF'
   In FD:OCA, the combination is used to indicate inheritance from a higher level in the structured object.
3. CS X'FFFF' with all other CP values
   A CS value of X'FFFF' used with CP values from X'0001' to X'FFFE' identifies a growing character set.

In the Intelligent Printer Data Stream* (IPDS*), both the GCSGID and CPGID are carried but are not treated as a CGCSGID construct. In this case, the following special-purpose values for GCSGID and CPGID are defined:

1. CS X'0000'
   The CS value of X'0000' means that no value is supplied.
2. CP X'0000'
   The CP value of X'0000' means that no value is supplied.
3. CP X'FFFF'
   The CP value of X'FFFF' implies that the device default code page should be used.

In IPDS and in MO:DCA the following special-purpose value is defined:

1. CS X'FFFF'
   The CS value of X'FFFF' implies that the set of characters with assigned code points in the resource definition of the selected code page is to be used.

Special CS and CP values are used to indicate "No CS, No CP" in the ACRI-EUC structure defined in c. the following special-purpose value is defined:

1. CS X'FDFF'
   The CS value of X'FDFF' implies that there is no character set, that is that the corresponding G set is not used for this particular EUC CCSID.
2. CP X'FDFF'
   The CP value of X'FDFF' implies that there is no code page, that is that the corresponding G set is not used for this particular EUC CCSID.

Within CDRA the following CS/CP pair have been used in the definition of Unicode CCSIDs.

1. CS X’FFF0’ (65520) and CP X’FFF0’ (65520)
   This CS/CP pair is used to represent an empty plane of Unicode. By definition CS 65520 is an empty set containing no characters and CP 65520 is a Unicode plane with no characters defined.

Additional Coding-Related Required Information

Some encoding schemes require specifications beyond the CS and CP elements to complete their definitions. Such specifications are called Additional Coding-related Required Information (ACRI) elements.

Three types of ACRI are defined below.

ACRI PC Mixed Byte (ACRI-PCMB)

This type of ACRI applies to ES values X'2300', X'2305' or X'3300' (see semantics of these ES values in Figure 9). It cannot be specified with any other ES values. It consists of the specification of ranges of valid first bytes of double-bytes associated with particular CS, CP pairs that are used with this encoding scheme. An ACRI-PCMB has the following format:

N S1 E1 S2 E2 -- -- Sk Ek -- -- Sn En

where N is the number of ranges of valid first bytes, Sk is the starting byte and Ek is the ending byte in the kth range, for all values of k from 1 to N. Sk and Ek are each in the range 128 to 255.

For example, ACRI-PCMB associated with CCSID 00942 in the CCSID Registry (see Appendix C: CCSID Repository) will be represented as:

2 129 159 224 252

In this example, there are two sets of valid first bytes (shown as their decimal values). The first set of 31 values is in the range 129 to 159 (X'81' to X'9F'), and the second set of 29 values in the range 224 to 252 (X'E0' to X'FC'). Thus, a total of 60 double-byte wards (14) can be defined using this ACRI-PCMB.

Other formats, such as a bit-pattern representation, are also possible.

ACRI Type EUC (ACRI-EUC)

This type of ACRI applies to ES value X'4403' only. It specifies the number of coded character sets and the width of each. It has the following format:

N W1 W2 W3 W4

where N is the number of coded graphic character sets, and Wn is the width of the nth set. If a G set is not used then the value of W is 0 and the corresponding CS/CP entries will be X'FDFF'.

ACRI Type TCP (ACRI-TCP)

This type of ACRI applies to ES value X'5404' only. It specifies the number of coded character sets followed by a triplet for each consisting of the width of the code points for the set, the length of the escape sequence used to designate the set into G0, and the actual escape sequence. The format is as follows:

n W1 LD1 D1 W2 LD2 D2 ... Wn LDn Dn

where

n =number of CGCSGIDs associated with the CCSID

W =width of code points in the code page

LD =length of the designation escape sequence

D =actual designation sequence

For example, the ACRI-TCP for CCSID 00965 (TCP for Traditional Chinese) is:

03 01 03 ESC 28 42 02 04 ESC 24 29 30 02 04 ESC 24 29 31

In this example the ESC mnemonic is shown, rather than the hex value 1B, to allow for ease of readability.

The format of the Escape Sequences is defined in ISO 2022. The "final byte," which defines the actual coded character set to be used, is defined in the ISO document International Register of Coded Character Sets to be used with Escape Sequences.

Short-Form Identification

Many implementations and architectures cannot accommodate variable-length tags like the long-form identifier. To address this problem, an alternative short-form fixed-length identifier called the Coded Character Set Identifier (CCSID) is defined.

Coded Character Set Identifier

A CCSID is a 16-bit identifier defined by CDRA. A CCSID, by definition, uniquely defines a data encoding. Given a CCSID tag and a valid code point, the character associated with that code point can be precisely identified. This is because the definition of the CCSID is linked to the definition of the code page in the IBM corporate registry. The definition of the control characters associated with a CCSID are inherited from the definition of controls defined for the related encoding scheme.

CCSIDs can be defined as growing. A growing CCSID is defined when the related code page is expected to be expanded. When the CCSID grows (i.e., more characters are added to the related code page and character set), a non-growing, fixed, CCSID is defined for the existing resources and the growing CCSID takes on the characteristics of the expanded resources. The range of CCSID values is 00000 (X'0000') to 65535 (X'FFFF'). The bit allocations in a CCSID are shown in Figure 10.

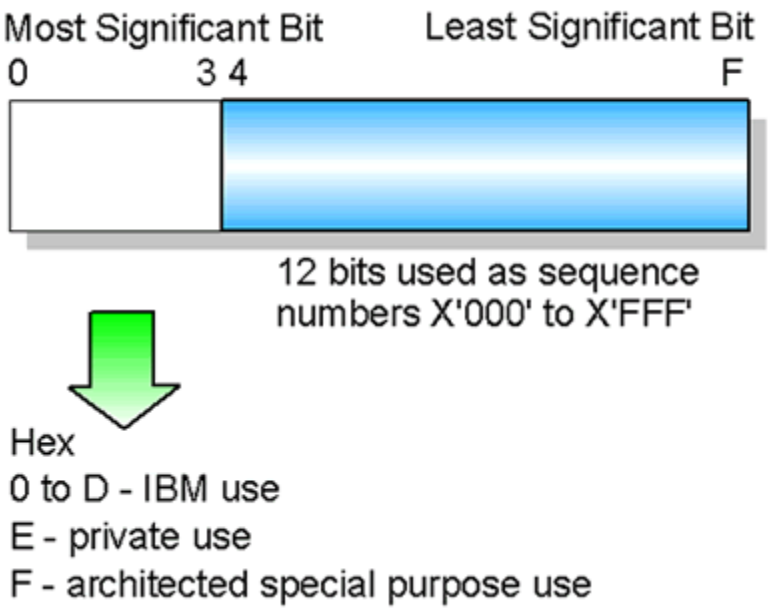


Figure 10. Bit Allocations in the Coded Character Set Identifier (CCSID)

Each CCSID has a corresponding long-form identifier or has a predefined special meaning. Figure 11 shows the allocation of CCSID values.

Value	Purpose/Meaning
X'0000'	Inheritance This value is reserved to show that the value of CCSID is defaulted and is to be taken from the next higher level in a defined hierarchy. It cannot be used if there is no hierarchy or no higher level. The highest level in the hierarchy cannot use this value. If a CCSID value of X'0000' is used when there is no higher level or no hierarchy, it will resolve to X'FFFF' (CCSID is not applicable).
X'0001' to X'DFFF'	IBM Registered CCSIDs These values are for IBM use. They will be registered and published in the CDRA documentation.
X'E000' to X'EFFF'	Private-use CCSIDs These values are reserved for private use. Customers must maintain their own organizational registries.
X'F000' to X'F0FF'	Reserved for future allocation by CDRA
X'F100' to X'F1FF'	Global Use CCSIDs These values are reserved for global use common character sets, such as the Syntactic character set, associated with specific encoding structures. This avoids the need to issue specific CCSIDs for usage of these character sets with every code page registered. The CCSID Repository contains a list of Global Use CCSIDs. Note: The use of Global Use CCSIDs is optional; it is determined individually by each implementation.
X'F200' to X'F2FF'	Reserved for RPQ use by products. Values in this range are specific to a product and must be completely defined by that product.
X'F300' to X'FFEF'	Reserved for future allocation by CDRA
X'FFF0'	CCSID for Empty Code Page
X'FFF1' to X'FFFB'	Reserved for future allocation by CDRA
X'FFFC' to X'FFFD'	Special value CCSIDs reserved for use in DB2.
X'FFFE'	Lower Level in Hierarchy This value is reserved to show that a value for CCSID at this level is not relevant. It should be obtained from the tag fields of elements at a lower level in the defined hierarchy. If a hierarchy does not exist, or if a CCSID value of X'FFFE' is specified at the lowest level, then the CCSID resolves to X'FFFF' (CCSID is not applicable).
X'FFFF'	CCSID is Not Applicable This value means that the tagged data is to be interpreted as "not graphic character data" or "actual representation is unknown".

Figure 11. Allocation of CCSID values

Representation of CDRA Identifiers

Internal Representation of CCSID, GCSGID, and CPGID

The representation of the identifier values, the syntax, is specified by providers of the tag fields that hold these identifier values. Each of these CDRA identifiers is a 16-bit binary number. The CDRA recommendation is that the internal representations be unsigned binary integers, rather than numeric character strings. If they are stored as alphanumeric strings, they must be tagged (implicitly or explicitly) like any other graphic character data.

Internal Representation of GCGID

The GCGID values are made up of uppercase A to Z, the digits 0 to 9, and a SPACE (trailing). The method of encoding them in an object must be identified in the object definition.

Internal Representation of ACRI

A variable-length array containing the value of each ACRI is needed to store the information. Each element in the array is a positive integer with a maximum value of 255. These numbers should be stored as binary values rather than strings of digits, to eliminate the need for tagging.

External Representation of Identifiers

CDRA identifiers may appear in documentation, display panels, program statements, or other textual strings. For consistency, CDRA recommends:

- The CCSID, CPGID, and GCSGID values be represented as 5-digit decimal numeric character strings; leading zeros may be replaced with spaces for presentation
- The ES value be represented as a 4-digit hexadecimal character string
- The ACRI be presented as a variable-length array of hexadecimal numbers
- The GCGID values be represented as 4- to 8-character alphanumeric strings; trailing zeros may be replaced with spaces for presentation.

As an aid to users, a descriptive name associated with each of the identifiers can also be presented.

CCSID Values

The CCSID values are categorized as follows:

- Interoperable CCSIDs:
  Interoperable CCSIDs have the following characteristics:

    - The character set is an interoperable set

  Supporting interoperable CCSIDs allows for:

    - Data interchange across various environments within a country/language (16) without data loss
    - Data interchange across various environments and countries within Group 1 without data loss.
    - See Appendix C: CCSID Repository for access to a complete list of CCSIDs.
- Global Use CCSIDs
  Some CCSIDs are defined with character sets that are globally applicable. These typically use the Syntactic Character Set (CS 640).
- Universal
  This category encompasses all the encoding forms of UCS, it is a Large Multi-Script Character set covering all the living languages of today, is the character set of the world-wide web and is expected to be supported in all computing environments. Its character set is a super set of the character sets of the non-UCS CCSIDs.
- Coexistence and Migration CCSIDs
  All other CCSIDs are classified as Coexistence and Migration CCSIDs. They may be widely used within a country or environment but not have the properties of an interoperable CCSID, or they may have a very specific, limited use such as a 7-bit symbols set.

Tagging in CDRA

When data is tagged with a CCSID, the GCGIDs assigned to the graphic character code points must be those defined by the CCSID.

When a graphic character is represented in data using a CCSID tag:

- It is in one of the CS elements found in one of the CS,CP pairs identified by the CCSID.
- The number of bytes in the code point is defined in the ESID element associated with the CCSID. In the case of mixed encodings, the number of bytes in the code point is defined by the respective CP element.
- The encoding scheme indicates if code extension controls (such as SO and SI) are required.

When data is to be interpreted according to a CCSID value:

- Parsing logic that respects the ESID element of the CCSID tag is needed to correctly process the data based on the number of bytes in each code point.
- The code point should be verified to be located in the graphic character encoding space with an assigned GCGID for the appropriate CP element of the CCSID.
- The GCGID must exist in the character set identified by the appropriate CS element of the CCSID.

When data with no assigned graphic character meaning is found, it should be treated as bytes.

These concepts are explained using two examples.

Example 1: Pure Single-Byte Case

In this example, let ESa, CSa, and CPa (in a single-byte encoding scheme) be the Encoding Scheme, Character Set, and Code Page elements of CCSIDa. See Figure 12 .

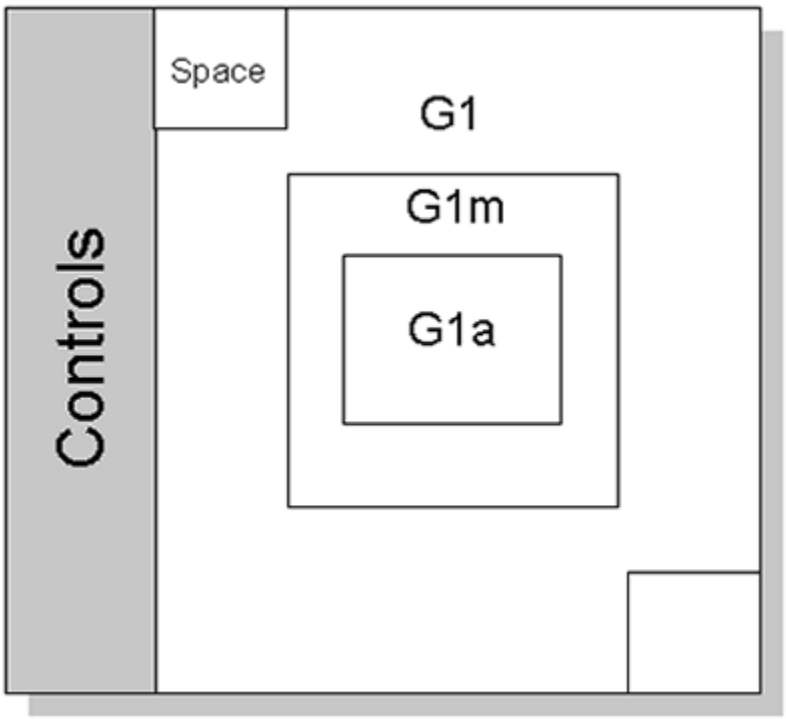


Figure 12. Meaning of Tagging: A Single Byte Example

The encoding space defined by ESa is composed of C and G1, where C is the control area and G1 is the graphic area. G1a represents all code points that have been assigned to the GCGIDs found in character set CSa. Only code points found within G1a

can have graphic character meaning according to the definition of CCSIDa. G1m represents all code points within CPa that have assigned GCGID values.

Example 2: Case of Mixed Single-Byte Double-Byte in PC

The example shown in Figure 13 uses a PC mixed single-byte and double-byte encoding. Let the elements of CCSIDa be ESa, CSa1, CPa1, CSa2, CPa2, and Fa (=ACRI-PCMB, ranges of valid first bytes).

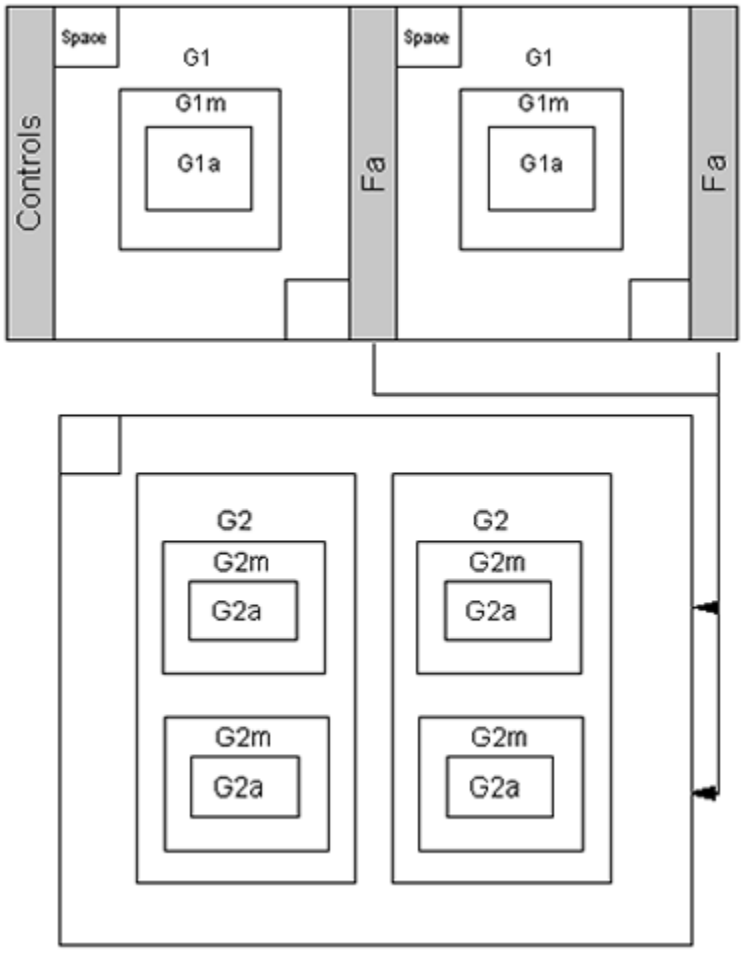


Figure 13. Meaning of Tagging: A PC Mixed SB/DB Example

The encoding space defined by ESa is composed of C, G1, and G2, where C is the control area, G1 is the single-byte graphic area, and G2 is the double-byte graphic area. G1a represents all of the single-byte code points that have been assigned to the GCGIDs found in character set CSa1. G2a represents all of the double-byte code points that have been assigned to the GCGIDs found in character set CSa2. Only code points found within G1a or G2a can have graphic character meaning according to the definition of CCSIDa. G1m and G2m represent all code points within CPa1 and CPa2,

respectively, that have assigned GCGID values. Fa represents the set of valid first bytes for double-byte code points found in G2a.

Meaning of Tagging in CDRA

CDRA has a dependency on other architectures, processes, or functions to provide proper graphic character data processing. The tags can be used to set the meaning or derive the meaning of code points in data to the extent defined above, when:

- Proper validation or filtering mechanisms to separate graphic character data from others are in place
- The CDRA tags are not erroneously applied to interpret the meaning of the bytes that do not have any graphic character meaning.

Relationship of Tags to Data Path

The data along with its tag may traverse many different systems through networks. In the process the tag value may get changed to reflect any conversion of the data. The tag values do not have any relationship to the data path.

Chapter 4. Services

This chapter describes several functions that are related to using the CDRA identifiers. CDRA-defined services are functions that are needed to consistently and correctly process graphic character data. These are detailed in "Chapter 5. CDRA Interface Definitions". Other related services are also discussed.

Data Flow Models

To describe the needed functions and to understand where the CDRA services and other related services may be used, a generic data flow model is used.

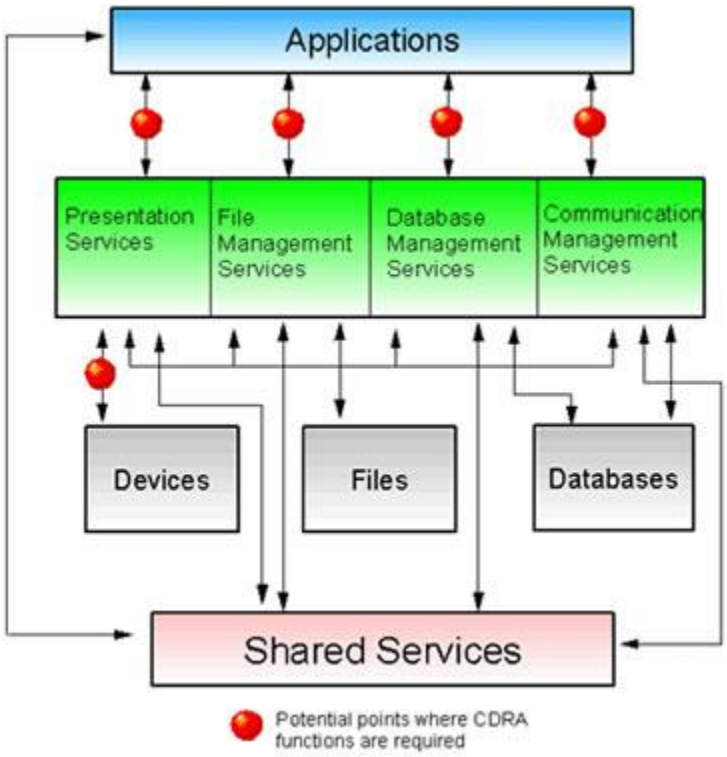


Figure 14. Data Flow Model

Figure 14 shows a model containing generic processing modules. This model does not imply any product implementation. Its elements and their individual roles are described below, identifying the CDRA functions that are needed.

Applications
A collection of processing functions that serve to execute a user application. Graphic character data is exchanged between the application and the operating system, or between the application and the providers of various types of services such as

presentation services, file management services, database management services, and communication services.

Shared Services
A collection of functions provided in the operating system that is useful to several components or applications in a system.

Presentation Services
A collection of functions that allow data exchanges between devices and applications. The services that handle graphic character data have the following capabilities:

- They support graphic character data interchange at the application interface.
- They have logic to recognize the device capabilities and to manage any difference between the encodings of application data strings and device capabilities
- They provide their own specific interface to applications; and provide the necessary tagging-related functions (such as set and query of tags) for graphic character data entities (such as fields, panels, windows, or sessions) at the application interface.

File Management Services
A collection of functions that allow applications to place data into organized data units called files. The graphic-character handling capabilities of these services are:

- Using their own interfaces to applications, they allow the application to set and query character data entities such as files, records, fields, or strings.
- They may provide automatic difference management functions to applications.
- In support of automatic difference management, they may have capabilities to support a specific set of encodings in the application or data views.

Database Management Services
A collection of functions that allow organizing and managing data as well-defined structures such as tables, columns, and rows. The graphic-character-data related aspects of these services include the following:

- Using their own method and interfaces, they allow applications or database administrators to set and query tags associated with graphic character data.
- They may provide transparent access to data from the database in a specified list of encodings at the application interface.

Communications Services
A collection of functions that allow various modules to communicate with peer modules or other modules in the same system or a different system.

- A minimum of a transparent data path is provided by these services.
- No automatic difference management functions are assumed to be present in the communications services in this model.

Applications have a choice of using the difference management services provided in the various service providers, where available. The list of encodings supported by each of these must be known to the application. The application must restrict its data across the service provider interfaces to one of these supported encodings. The potential interface points where CDRA functions are required are shown in Figure 14. If the service providers do not have support for a specific encoding the application must manage the difference.

CDRA-Defined Services and Other Related Services

An implementation of CDRA will include a collection of CDRA-defined services and other related services. Together they will provide the environment with a method of identifying graphic character data and managing the differences in graphic character data representations.

CDRA-defined services

CDRA-defined services have the following properties:

- Their semantics are definable and are equally applicable in all environments; the function can be defined in a manner independent of the system's or environment's unique methods or organizations.
- The function can be provided once per system, and will be of maximum benefit to many callers from different layers in the system.
- The syntax of the functions can be common across all environments; the common-service implementations will provide at least one function interface using the defined syntax. They may provide other system-specific function interfaces as well.

The CDRA-defined services can be grouped into four categories:

1. Functions for querying CCSID information
2. Functions for querying CCSID relationships
3. Functions related to difference management
4. Functions for identifying exception conditions.

A brief description of the functions in each group follows. Chapter 5, CDRA Interface Definitions details each of the APIs.

Functions for querying CCSID information
To assist in migrating from the use of an intermediate form of tagging, the caller of a CDRA service may obtain the various elements of a CCSID. Similarly, a caller may have an intermediate form and need to find the equivalent CCSID. Additionally, the caller may obtain control function assignment information. The following functions provide these services.

CDRGESP Get Encoding Scheme, Character Set, and Code Page Elements
CDRSCSP Get Short Form (CCSID) from Specified ES (CS, CP)
CDRGESE Get Encoding Scheme Element and its Sub-elements
CDRGCTL Get Control Function Definition
CDRSMXC Get Short Form (CCSID) with Maximal CS for Specified ES, CP

Functions for querying CCSID relationships
When an environment deals with multiple encoding schemes (such as PC-Data and PC-Display in the OS/2 environment, or the pure single-byte or mixed single-byte and double-byte codes in systems supporting Group 2 character sets), special query functions are needed to find different CCSIDs that have specific relationships. For example, when data is received in a CCSID that is not native to a system environment, such as PC-Data encoded data in an EBCDIC supporting iSeries* (aka AS/400*) system, it is necessary to find a supported CCSID that best relates to the CCSID of the received data. The following functions provide information on the relationships between CCSIDs:

CDRGRDC Get Related Default CCSID
CDRGCCN Get CCSID for Normalization

Functions related to difference management
When different data representations, as described by the CCSIDs of two entities is detected (using query functions), a conversion service can be called to convert data in one CCSID to another. The conversion service is a collection of conversion methods and supporting conversion tables. The concepts and criteria associated with difference management, the selection of an appropriate conversion method, and the creation of the contents of conversion tables are described in Chapter 6, Difference

Management and in Appendix B, Conversion Methods. The following functions are defined for difference management:

CDRCVRT Convert a Graphic Character String
CDRMSCI Multiple-Step Convert Initialize
CDRMSCP Multiple-Step Convert Perform
CDRMSCC Multiple-Step Convert Clean Up

Function for identifying exception conditions
A service is provided for callers of the CDRA services who cannot use the feedback structure. This service will provide the user with a status code and reason code as separate elements. The following function is defined for identifying exception conditions:

CDRXSRF Extract Status and Reason Codes from Feedback Code

Related Services

The related services include those functions that can potentially have a common syntax, but whose semantics cannot be defined in a common manner across all environments; for example, querying system defaults or the current session CCSID.

The resource management functions are contained in the group of related services. These functions are system-specific because they deal with system-specific resource structures, access, and storage; and they provide system-specific administration facilities (utilities and their end-user interfaces).

Functions that are not common within a system are object-manager-specific (for example, querying the CCSID of a relational database table). These functions may have a common object-manager-specific syntax across systems. The functions and their syntaxes cannot be supported as common services in all environments.

Brief descriptions of related services follow.

Setting Tag Values
Setting is the process of entering one or more tag values into the tag fields associated with different graphic character data elements. The set function is provided as part of the various object-manager-specific interfaces, or associated service functions. Setting of tag fields on entities owned by the operating system is provided by the system-specific interfaces. Setting is usually performed at object create time, and may be modified if the data encoding changes.

Querying Tag Values

Querying is the process of reading or obtaining one or more tag values associated with graphic character data elements. The query function is provided as part of various object-manager-specific interfaces, or associated service functions. Querying of tag fields in entities (or objects) owned by the operating system is provided by services in the operating system, as part of the system-specific services.

The query function also resolves any unknown or default tag values such as a CCSID value of X'0000'. Any hierarchy used for organizing the different elements owned by the object managers is object-manager-specific. Different types of queries are applicable to different object managers. For example, a file manager may provide for querying all the CCSIDs that may be present within a file; a presentation manager may provide for querying the CCSID used in a currently active window.

To assist in resolving the defaults in a hierarchy, a function may be provided to create and maintain a resource of default values for an application, a user, or another module. This resource is system-specific.

Resource Management Functions

Different systems have different structures and different methods of storing information resources. The content of the CDRA-supporting resources is defined by the architecture; however, the way the resources are stored is system-specific as are the functions for maintaining (creating, updating, and querying) them. These functions, when provided, are classed as related services.

The related services include those functions that can potentially have a common syntax, but whose semantics cannot be defined in a common manner across all environments; for example, querying system defaults or the current session CCSID.

The resource management functions are contained in the group of related services. These functions are system-specific because they deal with system-specific resource structures, access, and storage; and they provide system-specific administration facilities (utilities and their end-user interfaces).

Functions that are not common within a system are object-manager-specific (for example, querying the CCSID of a relational database table). These functions may have a common object-manager-specific syntax across systems. The functions and their syntaxes cannot be supported as common services in all environments.

Brief descriptions of related services follow.

Setting Tag Values

Setting is the process of entering one or more tag values into the tag fields associated with different graphic character data elements. The set function is provided as part of the various object-manager-specific interfaces, or associated service functions. Setting of tag fields on entities owned by the operating system is provided by the system-specific interfaces. Setting is usually performed at object create time, and may be modified if the data encoding changes.

Querying Tag Values

Querying is the process of reading or obtaining one or more tag values associated with graphic character data elements. The query function is provided as part of various object-manager-specific interfaces, or associated service functions. Querying of tag fields in entities (or objects) owned by the operating system is provided by services in the operating system, as part of the system-specific services.

The query function also resolves any unknown or default tag values such as a CCSID value of X'0000'. Any hierarchy used for organizing the different elements owned by the object managers is object-manager-specific. Different types of queries are applicable to different object managers. For example, a file manager may provide for querying all the CCSIDs that may be present within a file; a presentation manager may provide for querying the CCSID used in a currently active window.

To assist in resolving the defaults in a hierarchy, a function may be provided to create and maintain a resource of default values for an application, a user, or another module. This resource is system-specific.

Resource Management Functions

Different systems have different structures and different methods of storing information resources. The content of the CDRA-supporting resources is defined by the architecture; however, the way the resources are stored is system-specific as are the functions for maintaining (creating, updating, and querying) them. These functions, when provided, are classed as related services.

Chapter 5. CDRA Interface Definitions

This chapter contains the specifications of the CDRA-defined functions (Application Programming Interfaces or APIs). These functions are provided as procedure calls that are independent of programming languages.

Function Templates and Defined Conventions

The CDRA functions are described using a common template containing the following information:

CDRXXXX - Descriptive Function Name
A single line title for the function definition, consisting of a short name and a descriptive name for the function.
Function Description
A short description of what the function does.
Resources Used
A list of any resources used by the function.
Function Syntax
Conventions used for syntax are detailed in the next section. The following information is included:

CDRXXXX (Input and Output Parameters, Feedback)

Input:
List of Input parameters
IPARM1: a description, variable type, and permitted values
IPARM2: - - - - -
- - - : - - - - -

Input/Output
List of Input/Output parameters
IOPARM1: a description, variable type, and permitted values
IOPARM2: - - - - -
- - - : - - - - -

Output:
List of Output parameters
OPARM1: a description, variable type, and permitted values
OPARM2: - - - - -
- - - : - - - - -

FB:
Feedback codes and their meanings

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	description of condition reported

Usage Notes

Information that is considered to be useful to the caller of the function to provide a better understanding of the function.

Syntax Conventions

The syntax for the CDRA functions has been defined in general terms so that the functions and their associated parameters can be coded using all IBM high-level languages (HLLs).

The following conventions are used:

FUNCTION-NAME (Parameters, Feedback)

with the following explanations:

1. FUNCTION NAME: name of the function to be performed. The function names all begin with the prefix "CDR" standing for "Character Data Representation"

   Function names can be up to seven characters, consisting only of uppercase letters A to Z and digits 0 to 9. The first character of the name cannot be a digit. This restriction accommodates the limitation on function names of all known HLLs.

2. PARAMETERS: used to specify the desired input and output elements based upon the function to be performed and the output desired.
    1. Parameters are positional.
    2. A comma (,) is used as a delimiter between parameters.
    3. No optional parameters are permitted.

4. All the parameter values are passed by reference. The parameters are of the type "variable" or "address" (of a location in the caller's address space). Passing constants as parameters is not permitted (for example, X'01F4' or 500 as CCSID parameter value, or a literal string).
5. Space for variables and buffers is allocated by the "caller" of the function. The caller always specifies the variable names or buffer names in the function call. Any data that is exchanged between the caller and the function is always passed in these variables or buffer areas.
6. Parameters that contain the input values are in general kept separate from those that contain the output values or feedback code.
   Note: If the same variable or buffer name is used for more than one call parameter, the results of the function are not predictable.
7. The parameters are passed (in the input output stack between the caller and the function) in the same order (sequentially from first to last) as they appear in the function call.
8. The terms Input, Output, and Input/Output in the syntax descriptions are based on what generates the data that is passed across the function interface in the parameters -- variables or buffers.
    1. Input variables or buffers contain data that is supplied by the caller for use by the function.
    2. Output variables or buffers contain data that is generated by the function and returned to the caller.
    3. Input/Output variables or buffers contain data that is supplied by the caller for use by the function, and data that may be modified and returned by the function.
9. CDRA does not place any restrictions on the number of characters or the characters used in parameter names. The rules specified by the appropriate programming language or other syntax parser used to parse the caller's program source statements apply.
10. Parameter variables containing an integer value will be of the type "32-bit, signed, two's complement, binary". This type is supported across HLLs.
    Length values (data-typed as integers) interchanged across a call interface are constrained by programming language support considerations to a maximum of +999,999,999.

3. FEEDBACK: The feedback code, FB, is a 96-bit structure (it can also be viewed as an array of 12 bytes or an array of three 32-bit two's complement binary numbers). It is used to communicate to the caller the conditions arising during the execution of the function. FB is a required parameter in all function calls.
    1. Of the 12 bytes of FB (see Figure 15), the first four are used to indicate a status code (also known as a class code) and reason code (also known as

a cause code) associated with each condition that may arise. The remaining eight are reserved for use by CDRA.

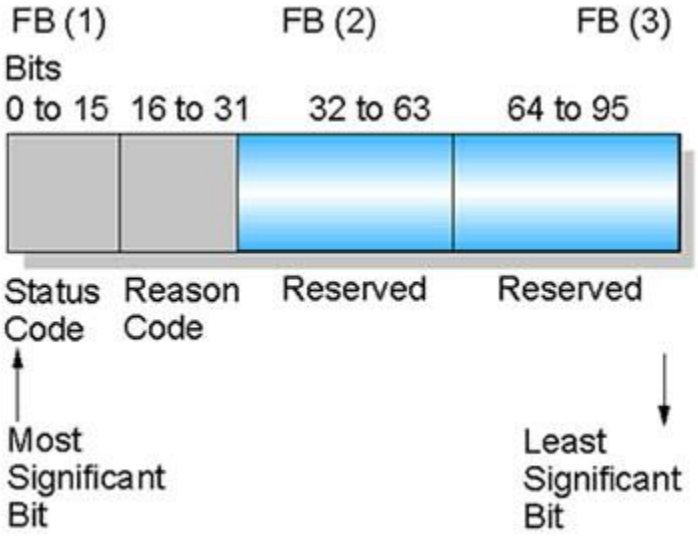


Figure 15. Structure of the feedback code (FB)

2. The status codes are described below under Status Code Convention . The reason codes are function-specific. See the list of status and reason code combinations under each function interface definition.
3. A zero value for each of the status and reason codes indicates that the function performed as expected with no detected conditions. If the status code is zero, the reason code must also be zero. A nonzero value in the first 16 bits indicates there was some condition in execution and the function may not have completed successfully.
4. The status code part of the feedback code indicates the class of problem encountered. The associated reason code part of the feedback code gives more information for error analysis and reporting.
5. Note:
    1. The level of severity of the error and the follow-on action to be taken are determined by the caller, depending on the specific nature of the problem associated with the called function.
    2. No explicit attempt is made in the CDRA definition to classify the feedback codes into different classes such as INFORMATION, WARNING, or ERROR.
    3. Depending on the function and the feedback code value, the calling function is responsible for handling the feedback code and taking appropriate action, such as making the appropriate values

available to a message service: for example, CCSID= X'5234' was not found.

6. All the feedback codes defined in CDRA are non-negative numbers. As a result, the status code value that forms the most significant 16 bits of the feedback code is restricted to a maximum of X'7FFF'.

4. Violations of the syntax for the functions are expected to be detected and handled by the appropriate parser that is used to parse the function call statements of the caller. An indication of a parsing error is expected to be returned by the parser.

Status Code Convention

The status codes have the following generic meanings, and each function description expands on the specifics of their use.

Status (Hex)	Meaning
0000 to 00FF	Common to several functions and environments; specific values used in this document are described later. The following are specific values in this range with their assigned meanings:
	Status (Hex) Meaning 0000 function completed successfully
0100 to 07FF	Common to all environments but function-specific; use of these status code values is documented in this chapter, along with the feedback code values of the functions that use them.

Depending on the function, the status code is accompanied by a function-specific reason code. The reason codes are detailed along with each function. A nonzero status code is always accompanied by a nonzero reason code.

An attempt has been made to assign unique feedback codes for significantly different conditions across the APIs. There are, however, several exceptions due to changes in the architecture specifications.

Note: Status and reason code values that are not specifically listed under each function are reserved for future allocation by CDRA.

Data Overflow Convention

Parameter lists for routines that have caller-allocated areas or arrays for receipt of data (or for variable-length character strings) must contain a minimum of two counts: one for the space allocated by the caller, and one for the space used by the function. The size of the allocated area is passed to the called function, and the function returns the size actually used in the caller-allocated area for the returned data.

When there is insufficient space allocated, a methodology is needed to deal with the overflow situation. A common convention (followed by both the caller and the function implementation) to deal with overflow situations is assumed for most CDRA callable functions (the exceptions are conversion-related functions where an overflow situation is dealt with by returning a truncated converted string).

The method described below is suitable for handling overflows when all the data to be returned by the function is available to the function, and is static data as opposed to dynamically computed data. (For example, it is information from a CCSID resource definition versus the result of a conversion process, which is dynamically generated.)

For dynamically generated data, the output will have to be computed from the beginning in order to return the next piece of data, especially since the assumptions for the CDRA callable functions are that input, and output data transferred across the call interface is all in the caller-allocated area (except for some local storage needs of the function). Such re-computations can cause severe degradation of performance for operations, such as the conversion of a long string. For this reason, the overflow-handling method described here is applied only to those CDRA functions that deal with static data.

In the following example the allocated space (variable N1) is 4, and the required space is 15. The variable N2 contains the actually-used space. The resource has N2max (N2max = 15 in this example) elements to be returned. Figure 16 shows the values of N1, N2, and the remaining number of elements to be retrieved.

Event		N1	N2	Remaining
First Call	Invocation	4	0	0
	Return	4	15	11
Second Call	Invocation	4	15	11
	Return	4	11	7
Third Call	Invocation	4	11	7
	Return	4	7	3
Fourth Call	Invocation	4	7	3

	Return	4	3	-

Figure 16. Example of Data Overflow Handling

The variable N2 in this example acts as an input/output variable. N2 is initialized to zero by the caller. The roles of the performing function and that of the caller are explained below:

The role of the performing function:

- It examines the value of N2 to determine where to start the output. If N1 is insufficient to hold all the returned data, the data will be returned in segments of size N1 until all of the data to be returned is exhausted.
- The function should see one of the following two relationships between N1 and N2 when it is called:
    - N2 is zero (set by the caller). The function returns the first piece of maximum N1 elements of data and returns the value N2max in N2. N2max is the total required space (N2max = 15 in the example shown in Figure 16). When the returned value of N2 is greater than N1, there is more data to be returned. A nonzero feedback code is returned, indicating that there is more data to be returned.
    - N2 is greater than N1. The function returns the next piece of maximum N1 elements of requested data starting at S, where "S = N2max - (N2 - N1) + 1" (elements are numbered starting at 1), and decrements N2 by N1. If the resultant N2 is greater than N1, a nonzero feedback code indicates that there is more data to be returned. If N2 is less than or equal to N1, the last block of data containing N2 elements is returned, along with a feedback code value of zero.

The caller has the following role:

- It calls the function with N2 initialized to zero.
- When the returned value of N2 is greater than N1 (along with the corresponding nonzero feedback code indicating that there is more data), the caller processes the returned segment of N1 elements and calls the function again for the remaining data.
- The caller must not change the values of N1 or N2 between function calls.
- When the returned value of N2 is less than or equal to N1, the function has returned all the remaining elements (consisting of N2 valid

elements) of requested data. The feedback code returned with the last block of data should be zero (unless some error condition other than overflow is encountered by the function).
- The caller should examine the feedback code from each call to ensure that there are no other conditions reported by the performing function. It is insufficient to merely rely on the relationship between N1 and N2.

Error situations can arise. Specific errors are documented with the individual function definitions.

Functions for Querying CCSID Information

CDRGESP - Get Encoding Scheme, Character Set, and Code Page Elements

Function Description
The most frequently accessed elements of a CCSID are the Encoding Scheme and the CS/CP elements. This function returns the value of the Encoding Scheme associated with CCSID1 in ES, and the values of the CS and CP elements in CSCPL.

Resources Used
CCSID Resource – see the section in Chapter 7 on the CCSID resource.

Function Syntax
CDRGESP (CCSID1, N1, N2, ES, CSCPL, FB)

Input:

CCSID1: this variable contains the CCSID value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

N1: this variable contains the size of the allocated area starting at CSCPL to contain the return data. N1 is specified as a number of elements, and each CS, CP pair is counted as two elements. Field type: 32-bit two's complement binary. It is an even number greater than or equal to 2.

Input/Output:

N2: this variable contains the number of values (each pair of CS and CP is counted as

two values) associated with CCSID1 and returned in CSCPL providing that sufficient space (N1) was allocated; field-type: 32-bit two's complement binary. The first invocation of this function must have N2 initialized to zero. The function's handling of the output and value of N2 returned is explained in section Data Overflow Convention Data Overflow Convention Data Overflow Convention.

Output:

ES: this variable contains the ES associated with CCSID1; field-type: 32-bit two's complement binary. It is a positive number in the range 4352 (X'00001100') to 65,534 (X'0000FFFE').

CSCPL: the field type of variable CSCPL is an array of 32-bit two's complement binary numbers whose format is CS1, CP1, CS2, CP2, ... CSn, CPn.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	CCSID1 value is not in the CCSID resource repository
0002	0001	CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. The invoker must resolve the default before invoking this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF').
0004	0001	the allocated length (value of N1) for the area to contain returned values was insufficient to contain all the output data that is to be returned. See Data Overflow Convention
0005	0002	N2 is greater than N1; however, the start of the next block of data to be returned is outside the valid range 1 to N2max
0005	000A	N2 is less than or equal to N1, but is not 0
0006	0001	the CCSID resource repository was not found
0006	0002	the CCSID resource repository is currently unavailable
0007	0001	the system CCSID resource repository accessed by the function was found to be invalid in structure

Status	Reason	Meaning
0007	0004	there was no ES element definition in the CCSID resource for CCSID1
0007	0006	there was no definition for CS, CP elements in the CCSID resource for CCSID1
0008	0001	CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0002	N1 value is greater than the maximum allowed in this implementation, or N1 is odd
0008	0003	N1 is less than 2

Usage Notes

The maximum number of CS, CP values depends on the ES. Most CCSIDs have only one CS, CP pair. See the section on Encoding Scheme Identifiers in Chapter 3 for further information. A caller can set N1 to 32, to accommodate up to 16 CS, CP pairs without overflow.

CDRSCSP - Get Short Form (CCSID) from Specified ES (CS, CP)

Function Description
This function gets the CCSID associated with the specified (CS, CP) pair(s) and ES. It aids in coexistence and migration for products that deal with the short form (CCSID) of identification on one side and the intermediate form (CGCSGID) on the other. The ES is further required to distinguish between usage of the same CS, CP with two different encoding schemes, (such as CS 00697 and CP 00850, with ES values of X'2100' and X'3100'), and when more than one CGCSGID is associated with a CCSID (such as with ES X'1301', X'2300', X'2305' and X'3300', for the CCSIDs registered to date. See Appendix C for information on the CCSID Resource Repository.

Resources Used
Repository of CCSID Resources (see section on the CCSID Resource in Chapter 7).

Function Syntax
CDRSCSP (CSCPL, N1, ESIN, CCSIDR, ESR, FB)

Input:

CSCPL: this variable is an array of 32-bit two's compliment binary numbers whose

format is CS1, CP1 CS2, CP2, ... CSn, CPn. Each CS is a positive number in the range 1(X'00000001') to 65,535(X'0000FFFF'). Each CP is a positive number in the range 1(X'00000001') to 65,534 (X'0000FFFF'). Each is placed in a single CSCPL array element.

N1: this variable contains the number of elements in CSCPL; field-type: 32-bit two's complement binary; a positive number in the range 2 to 32 (can accommodate up to 16 CS and CP pairs).

ESIN: this variable contains the ES value referenced; field-type: 32-bit two's complement binary; a zero, or a positive number in the range 4352 (X'00001100') to 65,534 (X'0000FFFE').

ESIN	Meaning
0	the caller does not know the ESID value; the CCSID returned is the first occurring in the CCSID resource repository whose CS and CP values match those specified in CSCPL.
Other	the user specifies the ESID value. See Figure 9 for a complete list of ESIDs and their associated meanings.

Output:

CCSIDR: this variable contains the returned CCSID value; field-type: 32-bit two's complement
binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF'). A value of 65,535 (X'0000FFFF') is returned when the function could not find the requested CCSID.

ESR: this variable contains the ES value of the returned CCSID; field-type: 32-bit two's complement binary; a zero, or a positive number in the range 4352(X'00001100') to 65,534(X'0000FFFE').

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully

Status	Reason	Meaning
0001	0001	no entry was found in the CCSID resource repository for the specified ESIN and CS, CP pair(s)
0001	0002	a single CCSID value with the specified CSCPL was found, but not with the specified ESIN. The values of the CCSID found and its associated ES are returned.
0002	0001	a CP value in CSCPL is zero
0002	0002	a CS value in CSCPL is zero
0003	0001	a CP value in CSCPL is 65,535 (X'0000FFFF')
0003	0002	Reserved
0005	0001	N1 is odd
0006	0001	the CCSID resource repository was not found
0006	0002	the CCSID repository is currently unavailable
0007	0001	the system CCSID resource repository accessed by the function was found to be invalid in structure
0008	0001	a CS or CP value in CSCPL array is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0002	N1 is greater than the maximum permitted in this implementation
0008	0003	N1 is less than 2
0008	0004	ESIN value is nonzero and not in the range 4352 (X'00001100') to 65,534 (X'0000FFFE')

Usage Notes

1. Often it is required to find the CCSID when the ES and (CS, CP) values are known. CS, CP (also known as CGCSGID or GCID) is used in many existing IBM architectures and data streams and supporting products. Together with Get Character Set and Code Page Elements (CDRGCSP), this function aids in coexistence and migration for products that deal with the short form (CCSID) of identification on one side and the intermediate form (CGCSGID) on the other. Because of the intermediate forms are by themselves incomplete when used in some encoding schemes, the function can return only a default value as defined in the installation's resources, when the ESIN information is not known.
2. For the CCSIDs defined to date, the maximum number of CS, CP pairs is 4 (up to eight values can be specified in CSCPL). Future CCSIDs may have more CS, CP pairs.
3. When an ESIN value of zero is specified the function will return the first CCSID encountered in the resource repository with matching CS, CP pairs. There may be additional CCSIDs that meet the specified criteria.

CDRGESE - Get Encoding Scheme Element and its Sub-elements

Function Description
This function gets the values of the Encoding Scheme identifier (ESID) element and each of its sub-elements for a given CCSID value (CCSID1) from the CCSID resource repository (see section "CCSID Resource" in Chapter 7). ESID has a two-byte hexadecimal format. The two bytes are decomposed as follows: the first nibble (first 4 bits of the first byte) is the basic encoding structure, the second nibble is the number of bytes indicator, and the second byte contains the code extension mechanism. The first two are defined in terms of X'0' to X'F', and the third as X'00' to X'FF'. The composite field is documented as its hexadecimal value and its unsigned decimal equivalent (see section "Encoding Scheme Identifier" in Chapter 3 for possible values).

Resources Used
CCSID Resource (see section "CCSID Resource" in Chapter 7).

Function Syntax
CDRGESE (CCSID1, ESEL, FB)

Input:

CCSID1: this variable contains the CCSID value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

Output:

ESEL: the function returns the values of ES identifier and its three sub-elements in this array of four elements; each element is a 32-bit two's complement binary number:

	ESEL
Element	Content and Range
1	value of ESID 4352 (X'00001100') to 65,534 (X'0000FFFE')
2	value of basic encoding structure sub-element 1 to 15 (X'00000001' to X'0000000F')
3	value of number of bytes indicator sub-element 1 to 15 (X'00000001' to X'0000000F')

ESEL

Element	Content and Range
4	value of code extension method sub-element 0 to 254 (X'00000000' to X'000000FE')

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	the CCSID1 value is not in the CCSID resource repository
0002	0001	the CCSID1 value is 0, which is reserved for indicating a default in hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000F00') to 65,535 (X'0000FFFF')
0006	0001	the CCSID resource repository was not found
0006	0002	the CCSID resource repository is currently unavailable
0007	0001	the system CCSID resource repository accessed by the function was found to be invalid in structure
0007	0004	there was no ES element definition in the CCSID resource for CCSID1
0008	0001	the CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')

Usage Note

The caller selects the appropriate element(s) of ESEL array for his or her purposes.

CDRGCTL - Get Control Function Definition

Function Description
This function gets a requested control function definition associated with a given CCSID from the CCSID resource repository (see section "CCSID Resource" in Chapter 7 for a model of the repository). The following control function definitions are defined in the CCSID resource repository model:

- Substitute

- New Line
- Line Feed
- Carriage Return
- End of File

The SPACE (SP01) definition is included in this function. Each control function definition is found as a triplet consisting of:

- The code point value allocated to the requested control function definition
- Its width in number of bytes
- The state number in which the code point is to be used

A triplet for each control function may be defined for each of the possible code extension switching states associated with the CCSID.
A selection parameter (SEL) is used to identify which control function definition is to be returned by the function.

Resources Used

CCSID Resource (see section "CCSID Resource" in Chapter 7).

Function Syntax
CDRGCTL (CCSID1, SEL, N1, N2, CTLFDF, FB)

Input:

CCSID1: this variable contains the CCSID value referenced; field-type: 32-bit, two's complement binary; a positive number in the range 1 ('00000001') to 65,279 (X'0000FEFF').

SEL: this variable containing the selection specification; field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255. If the selected control function element is available in the resource definition for CCSID1, the triplet(s) are returned in the area starting at CTLFDF. The following values are currently defined for SEL:

SEL	Selected Control Function
0	Space
1	Substitute

SEL	Selected Control Function
2	New Line
3	Line Feed
4	Carriage Return
5	End of File
6 to 255	Reserved for CDRA

N1: this variable contains the size of the allocated area starting at CTLFDF to contain the returned data. N1 is specified as a number of elements, each triplet is counted as 3 elements. field-type: 32-bit two's complement binary. It is a non-zero positive number whose minimum value is 3.

Input/Output:

N2: this variable will contain the number of values returned in CTLFDF; field-type: 32-bit two's complement binary. The first invocation of this function must have N2 initialized to zero. It is a non-negative integer and is a multiple of 3 (corresponding to each triplet in CTLFDF). If no definition is found in the CCSID resource for the requested element, a value of 0 is returned in N2. The function's handling of the output and value of N2 returned is explained in section Data Overflow Convention .

Output:

CTLFDF: this variable contains the start of the area reserved for the return definition element(s). Each element is a triplet of 3, field-type: 32-bit two's complement binary. For each triplet the first value is the code point, the second is the code point width, and the third value contains the switching state number. There is one triplet returned for each switching state for CCSID1. An undefined element is indicated by a zero state number in the corresponding CTLFDF entry.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	the CCSID1 value is not in the CCSID resource repository

Status	Reason	Meaning
0001	0004	one or more of the requested control function definitions are undefined (as indicated by a zero value for its corresponding state number in CTLFDF)
0001	000A	the requested control function definition element in the CCSID resource for CCSID1 was not found
0002	0001	the CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. The invoker must resolve the default before invoking this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0004	0001	the allocated length (value of N1) for the area to contain returned values was insufficient to contain all the output data that is to be returned. See Data Overflow Convention .
0005	0002	N2 is greater than N1; however, the start of the next block of data to be returned is outside the valid range 1 to N2max
0005	0003	the value specified in the SEL parameter is not supported
0005	000A	N2 is less than or equal to N1, but is not 0
0006	0001	the CCSID resource repository was not found
0006	0002	the CCSID resource repository is currently unavailable
0007	0001	the system CCSID resource repository accessed by the function was found to be invalid in structure
0008	0001	CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0002	N1 is greater than the maximum permitted in this implementation
0008	000A	N1 is less than 3.
0008	000B	the SEL value is not in the range 0 to 255.

Usage Notes

1. The maximum number of code extension states (and the associated corresponding code pages) for the CCSID depends on the ES. Most CCSIDs have only one state. The maximum is four for the CCSIDs registered to date, though some future CCSIDs may have more. An invoking function can set N1 to 48, to accommodate up to 16 triplets of information without overflow.
2. The code point value for any control function definition can be in the range X'00000000' to X'7FFFFFFF', only up to four-byte code points can be defined. The code point width values can be 1 to 4 (bytes).

CDRSMXC - Get Short Form (CCSID) with Maximal CS for Specified ES, CP

Function Description

This function gets the CCSID with the largest CS, either maximal or full, for a given CP. The function aids in coexistence and migration; it allows a caller to get an appropriate CCSID when only the CP is known. The ES parameter may be specified to distinguish between usage of the same CP with two different encoding schemes, such as PC Display and PC Data.
The function is restricted to pure single-byte pure double-byte CCSIDs that have only one CS, CP pair associated with them (for those registered to date).
The CCSID value returned by the function may differ from one implementation to another, as it is dependent on the content of the CCSID resource and the various implementations may support different CCSIDs.

Resources Used

Repository of CCSID Resources (see section "CCSID Resource" in Chapter 7).

Function Syntax

CDRSMXC (CPIN, ESIN, CCSIDR, ESR, FB)

Input:

CPIN: this variable contains the CP value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,534 (X'0000FFFE').

ESIN: this variable contains the ES value referenced;

ESIN	Meaning
0	the caller does not know the ES value, and expects the first CCSID encountered in the CCSID repository, with the specified CP and the "Full" or "Maximal" CS, to be returned.
Other	the invoker specifies the ESID value; field-type: 32-bit two's complement binary; a positive number in the range 4352 (X'00001100') to 65,534 (X'0000FFFE'). Only ESIDs that have a single (CS, CP) pair associated with

ESIN	Meaning

them are valid for this function. See Figure 9 for a complete list of ESIDs and their associated meanings.

Output:

CCSIDR: this variable contains the returned CCSID value; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF')
ESR: this variable contains the ES value of the returned CCSID; field-type: 32-bit two's complement binary; a zero, or a positive number in the range 4352 (X'00001100') to 65,534 (X'0000FFFE').

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	no entry was found in the CCSID resource repository for the specified CPIN, ESIN combination
0001	0003	ESIN was specified as 0; the first CCSID encountered in the CCSID repository, with the specified CP and the "Full" or "Maximal" CS was returned; additional CCSIDs meeting the criteria may exist.
0001	0009	the ESIN specified indicates that more than one pair of CS, CPs are associated with it, which is invalid for this function
0002	0001	the CPIN value is 0
0003	0001	the CPIN value is 65,535 (X'0000FFFF')
0006	0001	the CCSID resource repository was not found
0006	0002	the CCSID resource repository is currently unavailable
0007	0001	the system CCSID resource repository accessed by the function was found to be invalid in structure
0008	0001	the CPIN value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0009	the ESIN value is nonzero and not in the range 4352 (X'00001100') to 65,534 (X'0000FFFE')

Usage Notes

1. Some code page identifiers in use in the Far East refer to "pseudo" or "combined" code pages with the PC Mixed and Host Mixed encoding schemes. These identifiers are to be used as CCSIDs rather than CPGIDs. The CDRSMXC function will not return the corresponding CCSIDs for these combined code page identifier values.
2. When an ESIN value of zero is specified the function will return the first CCSID in the resource with the specified CP and a Full or Maximal size. There may be additional CCSIDs that meet the specified criteria.

Functions for querying CCSID relationships

CDRGRDC - Get Related Default CCSID

Function Description

A given CCSID may not be directly usable in many situations. This function allows the invoker to get a nearest equivalent or best-fit *related CCSID*. The related default is made available in the form of a resource table called *Related Default CCSID Table (RDCT)* (see section "Related Default CCSID Table (RDCT) Resource" in Chapter 7 for details). The caller supplies an ES value as an additional key to select the appropriate related CCSID.

Resources Used

Related Default CCSID Table (RDCT) - see section "Related Default CCSID Table (RDCT) Resource" in Chapter 7.

Function Syntax

CDRGRDC (CCSID1, ESIN, SEL, CCSIDR, FB)

Input:

CCSID1: this variable contains the CCSID value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

ESIN: this variable contains the ES value referenced; field-type: 32-bit two's complement binary; a positive number in the range 4352 (X'00001100') to 65,534 (X'0000FFFE'). See Figure 9 in Chapter 3 for the list of valid ESIDs and their associated meanings.

SEL: This variable is reserved to identify any specific selection criteria as additional input, for example, to select among two equally valid related defaults; field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255.

SEL	Meaning
0	Installation default
1 to 127	Reserved for use by CDRA
128 to 255	Reserved for customer use

Output:

CCSIDR: this variable contains the returned CCSID value; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF'). If no related default is found, CCSIDR is set to CCSID1.
FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	no entry was found in the Related Default CCSID Table(RDCT) resource for the CCSID1, ESIN, and SEL combination specified. The CCSID1 value is copied and returned in CCSIDR.
0002	0001	the CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. It must be resolved before this function is called.
0003	0001	CCSID1 has one of the special- purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF'); it cannot have a related default
0005	0001	the value of SEL specified is not supported
0006	0001	the RDCT resource was not found
0006	0002	the RDCT resource is currently unavailable
0007	0001	the system RDCT resource accessed by the function is found to be invalid in structure
0008	0001	the CCSID1 value is not in the range 0 (X'0000000') to 65,535 (X'0000FFFF')
0008	0002	the ESIN value is not in the range 4352 (X'00001100') to 65,534 (X'0000FFFE'). The CCSID1 value is copied and returned in CCSIDR.

Status	Reason	Meaning
0008	000B	the SEL value is not in the range 0 to 255.

CDRGCCN - Get CCSID for Normalization

Function Description

When certain operations, such as concatenation or comparison, are performed on graphic character strings, the two strings are both in the same CCSID, or they are normalized first to a single CCSID before concatenation. This function assists in determining the CCSID for normalization given two CCSIDs. The returned CCSID may equal one or both the input CCSIDs.

Resources Used

Normalization Support CCSID Table (NSCT) (see section "Normalization Support CCSID Table (NSCT) Resource" in Chapter 7).

Function Syntax

CDRGCCN (CCSID1, CCSID2, CCSIDN, HINTV, FB)

Input:

CCSID1: this variable contains the CCSID value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

CCSID2: this variable contains the second CCSID value referenced; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

Output:

CCSIDN: this variable contains the returned CCSID value for normalization; field-type: 32-bit two's complement binary; a positive number in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

HINTV: the function returns in this variable a number (field-type: 32-bit two's complement binary) that conveys information to assist the calling function in its subsequent processing. The following values and meanings are defined:

HINTV	Meaning
0	No hints
1	CCSID1 and CCSID2 have both the same value for their CP element. The returned CCSIDN has a character set which is a superset of or equals the larger of the character sets of CCSID1 and CCSID2.
2	CCSIDN has the same CP element as CCSID2. The character set of CCSIDN is a superset of or equals the character set of CCSID2. Only the string with CCSID1 needs to be converted to CCSIDN.
3	CCSIDN has the same CP element as CCSID1. The character set of CCSIDN is a superset of or equals the character set of CCSID1. Only the string with CCSID2 needs to be converted to CCSIDN.
4-127	Reserved for use by CDRA.
128-255	Reserved for customer use.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	there is no entry in the resource -- Normalization Support CCSID Table (NSCT) -- for the pair CCSID1, CCSID2
0002	0001	the CCSID1 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0002	0002	the CCSID2 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0003	0002	CCSID2 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0006	0001	the NSCT resource was not found
0006	0002	the NSCT resource is currently unavailable
0007	0001	the system NSCT resource accessed by the function is found to be invalid in structure

Status	Reason	Meaning
0008	0001	the CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0002	the CCSID2 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')

Usage Notes

The values returned by this function are implementation specific and may vary from system to system.

Functions for character data conversion

CDRCVRT - Convert a Graphic Character String

Function Description

This function converts a graphic character data string of the identified string type (ST1) represented in a specified "from" CCSID (CCSID1) to a graphic character data string of the required string type (ST2) that is represented in another specified "to" CCSID (CCSID2).
The function assumes that the entire string to be converted is known and is passed to the function. Also, the caller has provided sufficient space for the returned converted string. In case of an overflow situation, an orderly truncation would result.
To perform the conversion, a (CCSID1, ST1) to (CCSID2, ST2) entry must exist in the Graphic Character Conversion Selection Table (GCCST), along with the conversion method and conversion tables. See the description and model of this table in chapter 7.
An installation may need to support more than one conversion method or conversion table(s) from a given (CCSID1, ST1) to (CCSID2, ST2). Each of these alternatives must have an entry in the GCCST. The Graphic Character Conversion Alternative Selection Number (GCCASN) differentiates the alternatives, if available, for pairs (CCSID1, ST1) to (CCSID2, ST2) from one another.

Note:
The GCCASN is not to be confused with the options used in creating the conversion tables following different criteria for mismatch management; it is used for selecting the appropriate conversion method and associated table(s).

Resources Used

- CCSID Resource
- Graphic Character Conversion Table (GCCT) repository
- Graphic Character Conversion Selection Table (GCCST)

Function Syntax

CDRCVRT (CCSID1, ST1, S1, L1, CCSID2, ST2, GCCASN, L2, S2, L3, L4, FB)

Input

CCSID1: this variable contains the CCSID value for the input graphic character data string being converted; field-type: 32-bit two's complement binary; a positive value in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

ST1: this variable contains the type of input string (types of strings are defined in chapter 6); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255. The following types are defined:

Type	Explanation
0	A Graphic Character String, as semantically defined by CCSID1.
1	A Graphic Character String, as semantically defined by CCSID1, and null-terminated. Null-terminated strings are defined in chapter 6.
2	A Graphic Character String, as semantically defined by CCSID2, and SPACE-padded. Padded strings are defined in chapter 6.
3	Special Newline Nextline Handling, a complete description is found in chapter 6 under Types of Strings.
4-15	String types for bi-directional languages. Details of the characteristics of these string types are found in chapter 6 under Types of Strings.
16-255	Reserved for future use by CDRA

S1: this variable contains the starting address of the area in the caller's address space containing the graphic character data to be converted

L1: this variable contains the length (in number of bytes) of:

- the string to be converted when ST1=0 or
- the input buffer when ST1=1

contained in the area starting at S1; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific

CCSID2: this variable contains the CCSID value for the converted graphic character

data string; field-type: 32-bit two's complement binary; a positive value in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

ST2: this variable contains the type of output string (types of strings are defined in chapter 6); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255. The following types are defined:

Type	Explanation
0	A Graphic Character String, as semantically defined by CCSID1.
1	A Graphic Character String, as semantically defined by CCSID1, and null-terminated. Null-terminated strings are defined in chapter 6.
2	A Graphic Character String, as semantically defined by CCSID2, and SPACE-padded. Padded strings are defined in chapter 6.
3	Special Newline Nextline Handling, a complete description is found in chapter 6 under Types of Strings.
4-15	String types for bi-directional languages. Details of the characteristics of these string types are found in chapter 6 under Types of Strings.
16-255	Reserved for future use by CDRA

GCCASN: this variable contains a number that identifies which conversion alternative is to be selected to convert graphic character data from (CCSID1, ST1) to (CCSID2, ST2); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255.

Value	Nature of the Conversion Alternative Selected
0	is used to select the designated "installation default" conversion method and table(s) (see chapter 7 for a model).
1	is used to select the CDRA-defined default method and associated conversion table(s). The difference management criterion used in the creation of the selected tables is based on country requirements to serve most applications using the selected CCSID pairs.
2-9	are reserved for future allocation by CDRA.
10-55	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the round-trip mismatch management criterion.
56-101	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the enforced subset mismatch management criterion.

Value	Nature of the Conversion Alternative Selected
102-147	are reserved to select other CDRA-defined alternatives. These alternatives may include conversions where: • the mismatch management criterion used in creating any of the selected tables is other than round trip or enforced subset • multiple conversion tables are selected and unequal criteria have been used when creating the different tables.
148-255	are reserved for selecting customer-defined alternatives. A customer organization may establish and control ranges of GCCASN to distinguish between different mismatch management criteria, similar to the IBM-defined ones described above.

L2: this variable contains the byte-length of the allocated area starting at S2 to contain the converted graphic character data; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific.

Output

S2: the converted graphic character data is placed in the area (in the invoker's address space), whose starting address is specified by the invoker in S2; the area's allocated length is given in L2.
Under certain error conditions the output may contain the results of converting only a part of the input string.

L3: this variable contains the byte-length of the converted string returned in S2; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific.
The byte-length includes any null termination or padding characters necessary to retain the semantics of CCSID2 and ST2.

L4: this variable contains a byte-number in the input string; field-type: 32-bit two's complement binary; a non-negative number whose maximum value is implementation-specific. The value of L4 is dependent upon the manner in which the convert function terminates. The values that may be returned are as follows:

- When the function detects an output buffer overflow condition, L4 is set to the first byte of the code point representing the next character to be converted in the input string S1.
- If the function detects an error in the input string, L4 contains the byte number in the input string S1 that is being processed when the error is detected.
- When the conversion is error-free, a value of zero is returned in L4.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	the requested conversion is not supported (there is no entry in GCCST for the specified (CCSID1, ST1), (CCSID2, ST2), GCCASN combination)
0001	0005	the requested conversion algorithm specified by GCCASN does not support the specified (CCSID1, ST1) to (CCSID2, ST2) combination
0001	0006	the GCCASN value is 0; but an "installation default" was not found in the GCCST for the pair (CCSID1, ST1) to (CCSID2, ST2)
0002	0001	the CCSID1 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0002	0002	the CCSID2 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0003	0002	CCSID2 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0004	0001	the length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to the first byte of the code point representing the next character to be converted in the input string S1.
0004	0002	the encoding scheme of CCSID1 is X'1301' (mixed Host SB/DB encoding). The length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to the first byte of a double-byte character (between SO and SI code points) that would have been converted next in the input buffer.

Status	Reason	Meaning
0005	0001	a pure double-byte CCSID1 was specified and either • ST1=0 and L1 is odd, or • vST1=1 and an orphan byte was found
0005	0004	ES of CCSID1 is X'1301', and a malformed string -- an odd number of bytes between SO, SI code points -- was encountered
0005	0005	a null-terminated input string was specified using ST1=1; however, there was no null-termination character in S1 within the length L1 specified.
0005	0006	a null-terminated output string was specified using ST2=1; however, the output string contains one or more characters matching the null-termination character, resulting from using the selected conversion tables and methods.
0005	0007	a SPACE-padded output string was specified using ST2=2; however, the definition for SPACE character could not be obtained (the CCSID resource definition did not have an entry for SPACE character definition, or the CCSID resource definition could not be found).
0005	0008	a pure double-byte CCSID2 with ST2=1 was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns only an even number of bytes (maximum L2-1 bytes), including the null-termination character in S2.
0005	0009	a pure double-byte CCSID2 with ST2=2 (SPACE-padded string) was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns L2-1 bytes, including the SPACE-padding characters, in S2.
0005	000C	ES of CCSID1 is X'1301', and a trailing SI bracket is missing.
0005	000D	ES of CCSID1 is X'1301', and a trailing SI code point was encountered without first encountering its corresponding leading SO code point (the number of intervening code points may have been odd or even; the code points would have been treated as single-byte code points because the leading SO was missing)
0006	0001	the selection table (GCCST) could not be found.
0006	0002	a CDRA resource is currently unavailable.
0006	0003	the conversion method identified in the GCCST for the specified selection is currently unavailable.
0006	0004	a conversion table identified in the GCCST for the specified selection could not be found.
0007	0001	the system GCCST resource accessed by the function is found to be invalid in structure.
0007	0002	the system GCCT resource accessed by the function is found to be invalid in structure.

Status	Reason	Meaning
0007	0003	the table type of GCCT does not match the method selected from GCCST.
0008	0001	the CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF').
0008	0002	the CCSID2 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF').
0008	0003	the ST1 value is not in the range 0 to 255.
0008	0004	the ST2 value is not in the range 0 to 255.
0008	0005	L1 is outside the range permitted by this implementation.
0008	0006	L2 is outside the range permitted by this implementation.
0008	0007	GCCASN is not in the range 0 to 255.
0100	0001	one or more input graphic characters were replaced with a "SUB" character specified for the output string.
0100	0002	the conversion specified has resulted in character mismatches.

Usage Notes

1. Some of the above status and reason code values are possible only when the method selected and the tables used have the capabilities to indicate that a character replacement has occurred (using shadow flags or other equivalent means) or that a substitution with a SUB character was done.
2. When CDRCVRT terminates with a feedback code indicating that the area allocated for output was insufficient, it is the responsibility of the caller to ensure that the remaining portion of the input string is semantically correct prior to making a subsequent call to complete the conversion. For example, in the case of input data with an encoding scheme of X'1301' (mixed Host SB/DB encoding), an SI is added at the end of the truncated output string if required; however, no alteration is made to the input string. If a subsequent call is made with the remainder of the input string, the function will terminate unsuccessfully as an SI will be encountered without a leading SO.
3. When the encoding associated with a CCSID is such that all graphic character code points are a fixed number of bytes (for example, two for pure double-byte), the assumption is that there are no characters (control or graphic) with a code point width different from that called for by the encoding scheme (other than the termination characters appropriate for the specified input string type) in the input string. The caller is responsible for filtering out any such characters or sequences before calling the function.

CDRMSCI - Multiple-Step Convert Initialize

Function Description

This function is part of the triplet of functions used in a multiple step conversion. It is the initializing function, CDRMSCP is the actual conversion function, and CDRMSCC is the cleanup function.
CDRMSCI performs initialization in preparation for subsequent calls to CDRMSCP. Initialization-related steps performed by the function include:

- Locate the appropriate method and conversion table(s)
- Allocate workspace
- Retrieve and move the method or tables into the execution workspace
- Resolve all the default values that are needed to perform the conversion, applying any specified overrides for them
- Initialize all pointers, state flags, or other controlling information associated with the conversion method.

For the conversion to be performed, an entry must exist for the specified set of:
From CCSID, To CCSID, From ST, To ST, and GCCASN
in the Graphic Character Conversion Selection Table (GCCST), along with the conversion method and conversion table(s). This function returns a token, which can be used to convert graphic character strings in subsequent calls to CDRMSCP (perform conversion). The token corresponds to a control block (or other equivalent mechanism) in the conversion service. The control block and all the allocated resources and pointers are made available to the invoker for performing conversion without incurring any initialization overhead.

Resources Used

- CCSID Resource (see chapter 7 for a description of the resource)
- Graphic Character Conversion Table (GCCT) repository (see chapter 7 for a description of the resource)
- Graphic Character Conversion Selection Table (GCCST) (see chapter 7 for a description of the resource)

Function Syntax

CDRMSCI (CCSID1, ST1, CCSID2, ST2, GCCASN, TOKEN, FB)

Input

CCSID1: this variable contains the CCSID value for the input graphic character data string being converted; field-type: 32-bit two's complement binary; a positive value in

the range 1 (X'00000001') to 65,279 (X'0000FEFF').

ST1: this variable contains the type of input string (types of strings are defined in chapter 6); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255. The following types are defined:

Type	Explanation
0	A Graphic Character String, as semantically defined by CCSID1.
1	A Graphic Character String, as semantically defined by CCSID1, and null-terminated. Null-terminated strings are defined in chapter 6.
2	A Graphic Character String, as semantically defined by CCSID2, and SPACE-padded. Padded strings are defined in chapter 6.
3	Special Newline Nextline Handling, a complete description is found in chapter 6 under Types of Strings.
4-15	String types for bi-directional languages. Details of the characteristics of these string types are found in chapter 6 under Types of Strings.
16-255	Reserved for future use by CDRA

CCSID2: this variable contains the CCSID value for the converted graphic character data string; field-type: 32-bit two's complement binary; a positive value in the range 1 (X'00000001') to 65,279 (X'0000FEFF').

ST2: this variable contains the type of output string (sting types are defined in chapter 6); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255. The following types are defined:

Type	Explanation
0	A Graphic Character String, as semantically defined by CCSID1.
1	A Graphic Character String, as semantically defined by CCSID1, and null-terminated. Null-terminated strings are defined in chapter 6.
2	A Graphic Character String, as semantically defined by CCSID2, and SPACE-padded. Padded strings are defined in chapter 6.
3	Special Newline Nextline Handling, a complete description is found in chapter 6 under Types of Strings.
4-15	String types for bi-directional languages. Details of the characteristics of these string types are found in chapter 6 under Types of Strings.
16-255	Reserved for future use by CDRA

GCCASN: this variable contains a number that identifies which conversion alternative is to be selected to convert graphic character data from (CCSID1, ST1) to (CCSID2,

ST2); field-type: 32-bit two's complement binary; a non-negative number in the range 0 to 255.

Value	Nature of the Conversion Alternative Selected
0	is used to select the designated "installation default" conversion method and table(s) (see chapter 7 for a model).
1	is used to select the CDRA-defined default method and associated conversion table(s). The difference management criterion used in the creation of the selected tables is based on country requirements to serve the majority of applications using the selected CCSID pairs.
2-9	are reserved for future allocation by CDRA.
10-55	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the round-trip mismatch management criterion.
56-101	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the enforced subset mismatch management criterion.
102-147	are reserved to select other CDRA-defined alternatives. These alternatives may include conversions where: • the mismatch management criterion used in creating any of the selected tables is other than round trip or enforced subset • more than one conversion table is selected and unequal criteria have been used when creating the different tables.
148-255	are reserved for selecting customer-defined alternatives. A customer organization may establish and control ranges of GCCASN to distinguish between different mismatch management criteria, similar to the IBM-defined ones described above.

Output

TOKEN: a 256-bit (eight 32-bit two's complement binary numbers) array to contain the value of a token returned by the initialize function. This token must be passed unchanged as an input parameter to subsequent CDRMSCP (Multiple-Step Convert Perform) calls and a closing CDRMSCC (Multiple-Step Convert Clean Up) call.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0001	0001	requested conversion is not supported (there is no entry in GCCST for the specified CCSID1, ST1, CCSID2, ST2, GCCASN combination)
0001	0005	the requested conversion algorithm specified by GCCASN does not support the specified (CCSID1, ST1) to (CCSID2, ST2) combination
0001	0006	the GCCASN value is 0; but an "installation default" was not found in the GCCST, for the pair (CCSID1, ST1) to (CCSID2, ST2)
0002	0001	CCSID1 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The invoker must resolve the default before invoking this function.
0002	0002	CCSID2 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The invoker must resolve the default before invoking this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0003	0002	CCSID2 has one of the special-purpose CCSID values in the range 65,280 (X'0000FF00') to 65,535 (X'0000FFFF')
0005	0007	a SPACE-padded output string was specified using ST2=2; however, the definition for SPACE character could not be obtained -- the CCSID resource definition did not have an entry for SPACE character definition, or the CCSID resource definition could not be found.
0006	0001	the selection table (GCCST) was not found
0006	0002	a CDRA resource is currently unavailable
0006	0003	the conversion method identified in the GCCST for the specified selection is currently unavailable
0006	0004	a conversion table identified in the GCCST for the specified selection could not be found.
0006	0007	unable to generate TOKEN as requested
0007	0001	the system GCCST resource accessed by the function is found to be invalid in structure
0007	0002	the system GCCT resource accessed by the function is found to be invalid in structure
0007	0003	the table type of GCCT does not match the method selected from GCCST.
0008	0001	the CCSID1 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0002	CCSID2 value is not in the range 0 (X'00000000') to 65,535 (X'0000FFFF')
0008	0003	ST1 value is not in the range 0 to 255
0008	0004	ST2 value is not in the range 0 to 255

Status	Reason	Meaning
0008	0007	GCCASN is not in the range 0 to 255

Usage Notes

See Usage Notes under CDRCVRT.

CDRMSCP - Multiple-Step Convert Perform

Function Description

This function is part of the triplet of functions used in a multiple-step conversion. CDRMSCI is the initializing function, this is the actual conversion function, and CDRMSCC is the cleanup function.
CDRMSCP converts a graphic character data string using the previously allocated control block (along with all the associated resources needed to perform the conversion). The token received from a previous invocation of CDRMSCI (Multiple-Step Convert Initialize) is the mechanism to access the allocated control block.

Resources Used

Graphic Character Conversion Table (GCCT) (see the description of this resource in chapter 7).

Function Syntax

CDRMSCP (TOKEN, S1, L1, L2, S2, L3, L4, FB)

Input

TOKEN: a 256-bit (eight 32-bit binary) array that contains the value of a token obtained by invoking the initialize function CDRMSCI (Multiple-Step Convert Initialize).

S1: this variable contains the starting address of the area in the invoker's address space containing the graphic character data to be converted

L1: this variable contains the length (in number of bytes) of:
• the string to be converted when ST1=0 or
• the input buffer when ST1=1

contained in the area starting at S1; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific

L2: this variable contains the byte-length of the allocated area starting at S2 to contain the converted graphic character data; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific.

Output

S2: the converted graphic character data is placed in the area (in the invoker's address space), whose starting address is specified by the invoker in S2; the area's allocated length is given in L2.
Under certain error conditions the output may contain the results of converting only a part of the input string.

L3: this variable contains the byte-length of the converted string returned in S2; field-type: 32-bit two's complement binary; a positive number whose maximum value is implementation-specific.
The byte-length includes any null termination or padding characters necessary to retain the semantics of CCSID2 and ST2.

L4: this variable contains a byte-number in the input string; field-type: 32-bit two's complement binary; a non-negative number whose maximum value is implementation-specific. The value of L4 is dependent upon how the convert function terminates. The values that may be returned are as follows:
• When the function detects an output buffer overflow condition, L4 is set to the first byte of the code point representing the next character to be converted in the input string S1.
• If the function detects an error in the input string, L4 contains the byte number in the input string S1 that is being processed when the error is detected.
• When the conversion is error-free, a value of zero is returned in L4.

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully

Status	Reason	Meaning
0004	0001	the length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to the first byte of the code point representing the next character to be converted in the input string S1.
0004	0002	the encoding scheme of CCSID1 is X'1301' (mixed Host SB/DB encoding). The length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to the first byte of a double-byte character (between SO and SI brackets) that would have been converted next in the input buffer.
0005	0001	a pure double-byte CCSID1 was specified and either • ST1=0 and L1 is odd, or • ST1=1 and an orphan byte was found
0005	0004	ES of CCSID1 is X'1301', and a malformed string -- an odd number of bytes between SO, SI bracket -- was encountered
0005	0005	a null-terminated input string was specified using ST1=1; however, there was no null-termination character in S1 within the length L1 specified.
0005	0006	a null-terminated output string was specified using ST2=1; however, the output string contains one or more characters matching the null-termination character, resulting from using the selected conversion tables and methods.
0005	0008	a pure double-byte CCSID2 with ST2=1 was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns only an even number of bytes (maximum L2-1 bytes), including the null-termination character in S2.
0005	0009	a pure double-byte CCSID2 with ST2=2 (SPACE-padded string) was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns L2-1 bytes, including the SPACE-padding characters, in S2.
0005	000C	ES of CCSID1 is X'1301', and a trailing SI bracket is missing.
0005	000D	ES of CCSID1 is X'1301', and a trailing SI code point was encountered without first encountering its corresponding leading SO code point (the number of intervening code points may have been odd or even; the code points would have been treated as single-byte code points because the leading SO was missing)
0006	0006	the token is invalid in structure.

Status	Reason	Meaning
0008	0005	L1 is outside the range permitted by this implementation.
0008	0006	L2 is outside the range permitted by this implementation.
0100	0001	one or more input graphic characters were replaced with a "SUB" character specified for the output string.
0100	0002	the conversion specified have resulted in character mismatches.

Usage Notes

None

CDRMSCC - Multiple-Step Convert Clean Up

Function Description

This function is part of the triplet of functions used in a multiple-step conversion. CDRMSCI is the initializing function, CDRMSCP is the actual c conversion function, and this function is the cleanup function.
CDRMSCC releases the control block and all the allocated resources associated with TOKEN, on behalf of the invoker. All intermediate state or other control information for the conversion methods are also released once the control block (or equivalent internal structures in the common service implementation) is released.

Resources Used

Graphic Character Conversion Table (GCCT) (see "Graphic Character Conversion Table (GCCT) Resource" in chapter 7).

Function Syntax

CDRMSCC (TOKEN, FB)

Input/Output

TOKEN: a 256-bit (eight 32-bit binary) array that contains the value of a token that was generated by an earlier invocation of the initialize function CDRMSCI (Multiple-Step Convert Initialize). The token is filled with zeros and returned from a successful cleanup.

Output

FB: the function returns in this feedback array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully
0006	0006	the token is invalid in structure

Usage Notes

An invalid token condition can be caused by:

- The caller did not call CDRMSCI to initialize conversion
- The TOKEN value got garbled in the invoker's address space between the CDRMSCI call and the CDRMSCC call
- Some system service such as an idle or hung TOKEN cleanup routine may have removed that token value from a list of active tokens.

Function for Identifying Exception Conditions

CDRXSRF - Extract Status and Reason Codes from Feedback Code

Function Description

This function extracts the values of status and reason codes contained in the 96-bit structure of an input Feedback code. Its purpose is to assist the caller of a function dealing with the status and reason codes as individual values during error handling. An error condition is indicated by a nonzero value in the first 32 bits (the first array element) of the feedback code.

Resources Used

None

Function Syntax

CDRXSRF (INFB, STATUS, REASON, FB)

Input

INFB: this variable is the start of the area containing the input Feedback Code; it is an array of three 32-bit two's complement binary values (96 bits) (see Figure 15).

Output

STATUS: this variable contains the status code (value in the first 16 bits of INFB); field type: 32-bit two's complement binary.

REASON: this variable contains the reason code (value in the second 16 bits of INFB); field type: 32-bit two's complement binary.

FB: the function returns in this array the processing status (and any associated reason) for this function; field type: array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in Hex) contained in the first 32 bits of FB:

Status	Reason	Meaning
0000	0000	the function completed successfully

Usage Notes

None

Chapter 6. Difference Management

One of the key challenges in heterogeneous environments is to be able to deal with different coded graphic character sets and code pages in a consistent manner. Differences exist for reasons such as the origins of operating systems, the provision of national language support in different countries, or an application's requirements.

Migration to interoperable character sets and code pages for different countries and groups of countries will minimize, but not eliminate, the differences to be dealt with. Applications will continue to face this challenge, but now with assistance from CDRA.

Four functions, CDRCVRT, CDRMSCI, CDRMSCP, and CDRMSCC, were defined earlier (in "Chapter 5. CDRA Interface Definitions") in support of difference management. The present chapter describes the concepts behind difference management, the principles and criteria in designing the contents of conversion tables, and some aspects of managing the selection of the required conversion methods and tables.

Concepts

Difference management is a process by which differences in coded graphic character representation of data are recognized and dealt with. Difference-detection mechanisms must be placed in appropriate locations within a system to determine if such differences do exist.

Differences in the data representation and the processing capabilities of an application are used to trigger a difference management action. The choices may be to convert the graphic character data, to leave it as it is, or to terminate the current function altogether. CDRA provides assistance when a choice to convert the data has been made. Conversion is viewed as a tool in difference management. The results of difference management within CDRA will be dependent upon the conversion tables and the conversion methods chosen.

A general view of the difference management process is shown in Figure 17.

The query function (see Querying Tag Values in chapter 4) assists in finding the relevant tag values to decide if a difference exists.

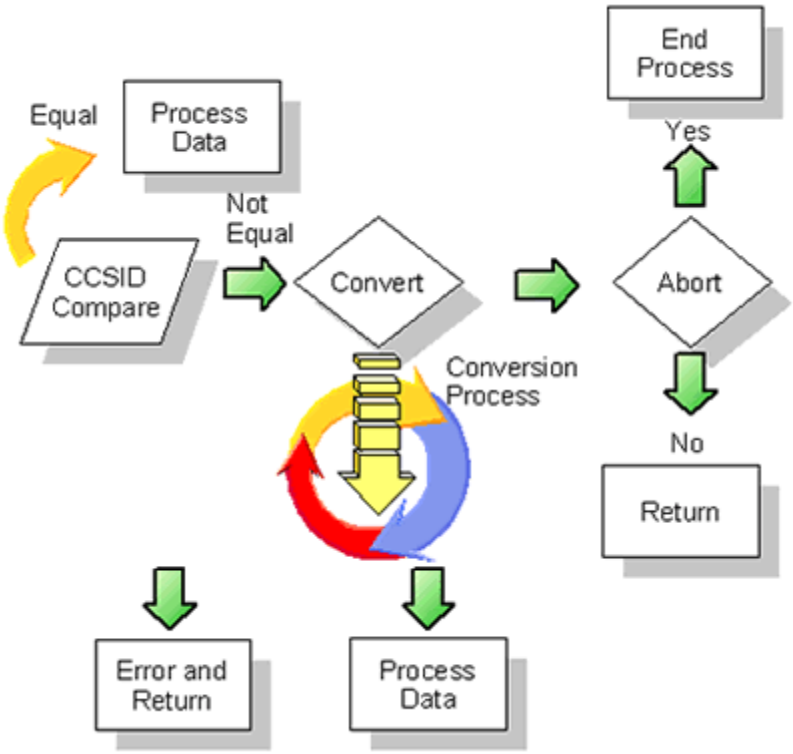


Figure 17. Difference management process flow

Do Not Convert

Potential reasons for not converting the data are as follows:

- The data can be processed as-is. Processing logic or a resource that can process the data correctly is available and can be selected.
- The data is required to remain in its original form (retaining its associated tag value). For example, when the context of use is simply store and retrieve, no processing is to be done in the storage environment; thus, conversion is not necessary. Note that this may not always be feasible, in that a receiving component may have restrictions on the CCSID(s) for the data handled by that component.
- The required conversion cannot be performed, as a necessary resource is not available. Here, the "do not convert" decision is made after the conversion function is called.
- The conversion results will be unsatisfactory. For example, the data loss or integrity loss could be unacceptable.

Convert

The simplest form of conversion is the case where the input and output character sets are equivalent, but the code point assignments for the characters are different. Here, all the matched characters will only need to have their code points mapped.

The more general form of conversion must deal with input and output character sets within which only a subset of the characters is equivalent.

During a conversion, only the common set of coded graphic characters can be preserved. Management of the remaining unmatched characters depends on the nature and context of the data. Conversion with mismatches can generate converted data that may not have an assigned graphic character meaning in the output. Such results of conversion may not be acceptable to an application.

CDRA has defined criteria for dealing with mismatches during the conversion process. The specific criterion to be used is reflected in the content of the conversion tables (and the logic that uses them) that are used in the conversion process. A set of default conversion tables has been defined to map between specific pairs of CCSIDs according to the most appropriate criterion, and are defined with consistency as the goal. The use of such tables enhances the consistency among implementing products when performing coded graphic character data conversion.

One of CDRA's goals is to minimize the loss of coded graphic character data during conversion. The interoperable character sets and associated CDRA-defined conversion tables help to address this goal by maximizing graphic character integrity within a character-set group or subgroup.

Generic data conversion process

Once the decision to convert has been made, a generic data conversion process can be used.

A generic data conversion process contains many elements, one of which is the graphic character data conversion process. Figure 18 shows the elements of a typical data conversion procedure.

The different elements and their functions are:

- A parser, which is selected based on the architecture of the input data stream or on the description of the caller's view (input view) of the data. The parser is responsible for separating the input data into different classifications or substrings, such as graphic character data, control characters/functions,

floating-point numbers etc. These substrings, along with their characteristics, are passed on to the converter.

- The converter is responsible for sending each substring to the appropriate mapping module. Once mapped, each module returns the converted substring back to the converter. These substrings are then passed on the output generator.
- The mapping modules, accept substrings from the converter, perform the appropriate mappings and return the new substrings to the converter.
- The output generator receives the converted data substrings from the converter and puts them together as an output data stream made up of substrings of different classes.

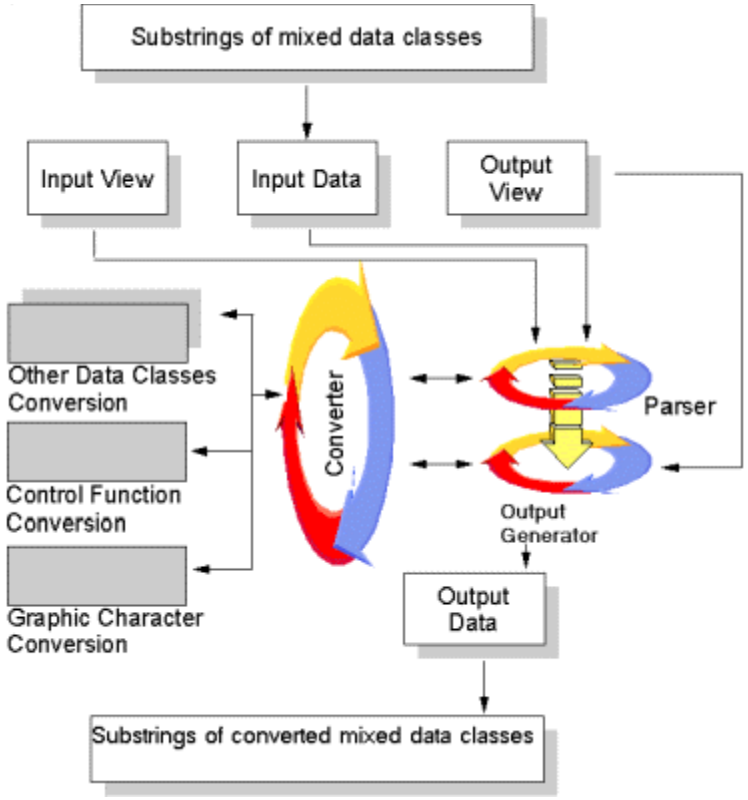


Figure 18. Generic data conversion process

CDRA defines only the graphic character data conversion part of the overall data conversion process. A limited number of control characters are addressed as part of handling different string types (see Types of Strings) and as part of control character mappings (see Pairings of Code Points). Other control characters are treated as bytes, and are dealt with according to mismatch management criteria.

Separation of Graphic Characters

For correct results, the caller of the CDRA conversion function should ensure that the input string does not contain characters other than graphic character data.

Each one of these conversion modules may permit direct access by an application. Here, the application assumes the responsibility for the functions of parsing and output generation. For example, when an application creates a sequential file in the PC, only it knows where the string of bytes is broken into logical substrings and which of these substrings represent graphic character data. Conventions such as CR, LF to show an end of record for organizing a file, must be known and handled by the parsing logic and the output generator. Handling of the data organization for output is not performed by the graphic character conversion function.

Misinterpretation of Data

If the separation of graphic character data from other classes of data is not done, the graphic character conversion function can find byte strings that may or may not have graphic character meanings. The criterion selected for mismatch management specifies how to convert such byte strings if they appear in the input string. However, the problem of possible misinterpretation cannot be entirely dealt with using the CDRA conversion criteria alone.

For example, if the data byte was representing a counter value equal to 74, which is the same bit configuration as the code point X'4A' for a left square bracket in a System/370* CCSID 00500, it will get converted to another code point (X'5B') representing the left square bracket in a PC using CCSID 00850. If this value is interpreted as a count on the PC, the value is now 91. Neither the CDRA identifiers nor the graphic character conversion process can deal with this kind of misinterpretation.

Types of Strings

A graphic character string may have several characteristics or properties associated with it. Some of these characteristics or properties are inherited from the encoding scheme such as the number of bytes per character. Others, such as how a string is terminated, the orientation of the string or whether the characters are shaped or unshaped cannot be determined by the CCSID tag or encoding scheme alone. The following String Types are defined for use within the CDRA architecture.

String Type 0: CDRA Default

If there is no string type specified in a CCSID definition or as a parameter on an API call then the string type is zero. A string type of 0 means that the character data string is semantically defined by the CCSID. All the characteristics of the string can be determined from the CCSID definition alone. No additional information is needed.

String Type 1: Null-terminated string

A variable-length graphic character string, which is terminated by a character whose code point has a binary value of zero. The number of bits in the code point used to represent the terminating character (the null terminator) is the smallest number of bits allowed for code points in the encoding scheme used.

The above definition is used in the following examples to determine the null-termination character:

- If the ES associated with CCSID1 indicates a "pure single-byte" or "mixed single-byte and double-byte" encoding, X'00' is used to terminate the string. In a mixed-byte string a null-termination can occur only in the single-byte segment of the string.
- In a string encoded using ES X'1301' (host mixed), a null-termination can occur only outside the SO and SI that surround the double-byte coded substring. Thus, any double-byte code point that begins with X'00' (such as in 327x data stream, where some EBCDIC control characters are represented in the data stream with X'00' preceding their corresponding single-byte code points) or ends with X'00' must not be interpreted to be a null-termination character.
- In a string using ES X'3300', X'2300' or X'2305', a double-byte code point can never begin with or terminate with X'00'. This also implies that no data code point in the string can be X'00'.
- If the encoding scheme indicates "pure double-byte" encoding, the null-termination character is X'0000'. This implies that none of the data code points in the string can be X'0000'.

The above definitions reflect the current usage and definitions of a null-terminated string in the C programming language. A length value may additionally be provided for the string; however, the null terminator takes precedence over the length value.

A null-terminated string is given a string type identifier of 1 in CDRA function calls and in the Graphic Character Conversion Selection Table (GCCST).

String Type 2: Padded string

A graphic character string that is padded with one or more space characters. Padding is done only when there is unused storage space available in an area containing the unpadded string, and when it can be done without violating the semantics of the encoding scheme of the CCSID of the string. The resultant space padded string will be a well-formed string following the semantics of its encoding scheme.

Caution: When space padding is done as part of graphic character conversion, it is not possible to distinguish (in the resultant output buffer) the space pad characters that are generated as a result of conversion maps from those generated by the padding process. If a subsequent string operation removes the space characters, there can be a potential loss of the converted pad characters.

- If the encoding scheme of the string is either "pure single-byte" or "mixed single-byte and double-byte", when the string occupies less than the area allocated for it, the string is padded to fill the remaining area with SPACE characters. The definition of SPACE is to be taken from the CCSID resource definition. In a mixed string the padding must be done only in the single-byte segment of the string.
- If the encoding scheme of the string is "pure double-byte", the SPACE character will have a double-byte code point specified in the string's CCSID resource definition.

String Type 3: "Special Newline Nextline Handling"

String type 3 has special meaning in certain IBM products. If a character data string is defined as a string type 3 than it is semantically defined by the CCSID with the additional property that any newline control characters in the string should be treated as linefeed control characters and likewise, any linefeed control characters should be treated as newline control characters.

String Types 4 - 15: String Types for Bidirectional Languages

In the case of bidirectional languages, the string type is used to describe characteristics that are not implied by the CCSID or Encoding Scheme. The string characteristics which are defined for the bidirectional string types are:

- Text Type
- Numeric Shaping
- Orientation
- Text Shaping
- Symmetrical Swapping.

Following is a brief description of each of these characteristics and their possible values.

Text Type
The text type characteristic states what kind of algorithm is to be used when transforming the text layout. The text type can be visual (reading sequence), implicit (typing sequence), or explicit (includes directional control characters in the text segments explicitly). A visual algorithm copies entire lines of text as they appear without bothering about existing embedded directional segments. An implicit algorithm recognizes directional segments based on the natural directionality of the characters (i.e., right to left for Arabic characters and left to right for English characters) and performs segment inversions accordingly. An explicit algorithm recognizes directional segments and performs inversions based on special, explicit, directional controls embedded in the text.

Example:
Visual, shaped text:

Implicit, unshaped text: Arabic implicit string

ة ي ب ر ع ل ا ة غ ل ل ا

Numeric Shaping
The numeric shaping characteristic states whether the numbers embedded in a text string will have the shapes that are used in English (called Arabic digits), or the national numerical shapes. Possible values for this characteristic are Arabic, Hindi or passthrough. When passthrough is specified numeric digits are left as they appear in the data string (no numeric shaping occurs).

Orientation
The orientation of a data string together with the text type, indicates the storage or display sequence of the Arabic and English characters. The possible values for this characteristic are left to right (LTR), right to left (RTL), Contextual LTR and Contextual RTL. The term contextual is used to indicate that the orientation should be taken from the context of the data. The data may contain "strong" characters that are either

orientation left or orientation right. The term following contextual (LTR or RTL) specifies what should be the default orientation when the data is orientation-neutral (i.e. there are no strong characters).

Text Shaping
The text shaping characteristic of a bidirectional string type indicates whether text shaping is performed. This is relevant for the scripts of Arabic languages (including Farsi and Urdu), where characters assume different shapes (initial, medial, final, or isolated) according to their position in a word and the connectivity traits of the character and its surroundings.

Symmetrical Swapping
The symmetric swapping characteristic states whether, in a right-to-left text phrase some directional pairs of characters (such as left and right parentheses, greater than and lesser than signs, left and right brackets, left and right braces) will be interchanged to preserve the logical meaning of the inverted text.

Each CCSID that is defined in support of a bidirectional language may have a default string type associated with it. If a string is tagged with a CCSID for a bidirectional language and no string type is explicitly specified than the default string type is to be used. If no default string type has been specified then the string type is defined to be 0.

The following table shows the specific characteristics of each bidirectional string type that have been defined to date.

String Type	Text Type	Numeric Shaping	Orientation	Text Shaping	Symmetrical Swapping
4	Visual	Passthrough	LTR	Shaped	Off
5	Implicit	Arabic	LTR	Unshaped	On
6	Implicit	Arabic	RTL	Unshaped	On
7	Visual	Passthrough	Contextual	Unshaped-Lig	Off
8	Visual	Passthrough	RTL	Shaped	Off
9	Visual	Passthrough	RTL	Shaped	On
10	Implicit	Arabic	Contextual LTR	Unshaped	On
11	Implicit	Arabic	Contextual RTL	Unshaped	On
12	Implicit	Arabic	RTL	Shaped	Off
13	Visual	Hindi Arabic-Indic	LTR	Shaped	Off
14	Visual	Hindi Arabic-Indic	RTL	Shaped	Off

15	Visual	Hindi Arabic-Indic	RTL	Shaped	On
16	Visual	Contextual	LTR	Shaped	Off
17	Visual	Contextual	RTL	Shaped	Off

Such strings are often interchanged in heterogeneous (or distributed) environments between applications that can support these string types. If the data conversion methods used for graphic character mapping are enhanced to deal with the parsing and assembly aspects of converting between specific types of strings, a degree of efficiency in performance can be attained. With this in view, provisions are made in the graphic conversion functions of CDRA to allow string-type specifications to select conversion methods that can deal with various string types besides converting the graphic characters.

Generic Graphic Character Conversion

A generic graphic character conversion function (see Conversion functions) converts an input graphic character string represented in a CCSID (the input CCSID) to an output string according to the CCSID specified for the output (the output CCSID). The interpretation of the input character string and the generation of the code points of the output character string adhere to the definitions of CCSIDs (see “Tagging in CDRA” in chapter 3).

The results of the conversion process will be the following:

- The meaning of all the graphic characters that are common (same GCGIDs) between the input CS and output CS will be preserved
- All other input graphic and non-graphic characters will be converted to output code points following the mismatch management criterion used. Their meaning cannot be preserved in the output CCSID, but they may be retrievable by mapping back to the input CCSID using an appropriate conversion table.

Conversion of strings between some CCSIDs cannot maintain the same byte-length between the input and output strings. For example, the coded representation of a string containing a mixture of Katakana characters (single-byte code points) and Japanese ideographic characters (double-byte code points):

- Will have embedded shift-in and shift-out control characters between the Katakana characters and the ideographic characters in a Japanese EBCDIC-based system
- Will not have any embedded shift-in and shift-out control characters in a Japanese PC-based system.

A function that converts the data between the two coding methods in this example will find a byte-length difference of at least two bytes. Provisions must be made to accommodate differences in byte lengths when developing and using conversion functions.

The designer of the conversion program can reference the CCSID elements and their definitions from CDRA documents. The logical steps in performing the conversion are:

1. Select an appropriate conversion method (see “Appendix B. Conversion methods”) based on the encoding schemes associated with the input and the output.
2. Select one or more conversion tables based on the CS and CP elements of the input and output CCSIDs. The following section describes the criteria that can be used for defining the contents of the conversion tables.

The various steps involved in selecting the conversion methods and the associated tables for different conversion criteria are described in section “Graphic character conversion selection table (GCCST) resource” in chapter 7.

Defining the contents of conversion tables

The input and output CCSIDs identify the CS, CP pairs. The content of a conversion table is determined by the input and output CS, CP pairs to be mapped. When there is more than one set of CS, CP pairs in the input to be matched with more than one set in the output, the principles described in Pairings of Code Points are used to determine the mapping.

If the input CS, CP pair has some common graphic characters that are split between two output CSs, then the corresponding support in the conversion method and tables of the appropriate type (see Appendix B. Conversion methods) are needed.

After the characters and their code point assignments are examined, they are categorized, and decisions are made about pairing the input and output code points.

A code point can be placed into one of the following categories:

1. SPACE: the code point is assigned to the SPACE character GCGID SP010000
2. Valid Graphic: the code point is assignable to a graphic character in the encoding structure, and is assigned a graphic character in the identified character set

3. Code Extension: the code point is assignable to a control character, and its assigned value is a valid code extension control character or the first character of a multiple-character code extension control as determined by the encoding scheme identified
4. Invalid Graphic: the code point is assignable to a graphic character in the encoding structure, but either it is not assigned any graphic character or it is assigned one that is not in the character set identified
5. Single Control: the code point is assignable to a control character, and is assigned a permitted control character for the application
6. Start of Control: the code point is assignable to a control character, and is assigned a permitted start of control sequence for the application
7. Invalid Control: the code point is assignable to a control character but is not assigned any control character, or it is assigned a character that is valid neither for the application nor as a code extension control defined in the encoding scheme.

Pairings of Code Points

The following general principles are used in pairing the input and output code points:

- Matched Graphic Characters and SPACE
  The graphic characters in the Valid Graphic category and the SPACE character from the input are compared with the Valid Graphic and the SPACE character in the output. For each Valid Graphic and SPACE character that is found in both sets, the code point in the input code page is mapped to its corresponding code point in the output code page. This set of characters is known as the "common character set". Graphic characters are defined to be matching if they have the same Graphic Character Global Identifier (GCGID).
- Code Extension Controls
  Some input or output control character code points may be used for code extension purposes. It is the responsibility of the conversion functions to handle these code points correctly. An example of this can be found in Method3 for EBCDIC Mixed to PC Mixed.
- Matched Control Mnemonics
  Some of the nongraphic characters can be commonly used control characters, such as Horizontal Tab (HT) or Carriage Return (CR). If these are found in graphic-character conversion, they will be handled on a "best-can-do" basis. The mnemonic names associated with them will be used as a guide to pair input and output code points in the control areas. The mnemonics may or may not have identical functional meanings in the input and the output environments.

- Remainder
  The remaining code points are treated as bytes. Some of these bytes may be graphic character code points outside the common character set, control mnemonics that have no matching control in the output, or non-allocated input code points. The character set mismatch management criterion is used to specify how these remaining characters are mapped.

Criteria for character set mismatch management

Character set mismatch management is necessarily context- or application-sensitive: what is best for one application may not be appropriate for another. Sometimes arbitrary decisions must be made, depending on the specific set of mismatched characters. Some general criteria for mismatch management are:

- Round trip integrity, where each byte value is preserved when data is returned from the target to the source
- Character replacement, or irreversible substitution of characters with appearance or meaning retention
- Enforced subset match, or irreversible substitution with SUB (substitute) control or an equivalent loss indicator character.

The application of these criteria results in different pairings of input and output code points for mismatched characters in conversion tables.

The above criteria are discussed in the following sections.

Round Trip Integrity

The objective of this criterion is to send data from one system to another one that has different representations of character data, and retrieve it without loss. Often the "do not convert" choice is not available. For example, data stored in a System/370 database is configured to have all its graphic character data in one CCSID. If it acts as a remote repository for data from a PC application, or from an application in another System/370 using a different CCSID, the data must be converted to the configured CCSID. The data is intended to be retrieved by the same application without loss when it is converted back for use in its original CCSID.

Interpretation of Converted Data in the Output CCSID

The tag associated with the converted data will be the CCSID of the output. The data will be interpreted -- possibly misinterpreted -- in the output environment. In the

absence of any validation or filtering services, data that has been converted using the round-trip criterion cannot be distinguished from data that has been created locally in the system, or that has been converted from another CCSID using the round-trip criterion. Data conversion is only one of the possible generators of code points that have no graphic meaning in a data object tagged with a CCSID. An application that generates hexadecimal constants and stores them along with other textual data is another possible generator.

Feasibility of Round-trip

Round trip mapping is always feasible for a common set of graphic characters or for a set of control characters with the same mnemonics, assuming there are no control sequences involved. The common sets of graphic and control characters within the initial input and output CCSIDs can be preserved irrespective of how many intermediate CCSIDs may be involved, provided that all the intermediate CCSIDs contain the same common sets.

The round trip of all remaining code points from a specific input to an output and back is feasible only under the following conditions:

- There are equal numbers of unmatched code points between the input and output.
- The bytes are mapped one-for-one from the input CCSID to the output CCSID.
- The same one-for-one relationship is used in the return path.
- If there are duplicate graphic characters in an input code (for example, CP 850) and if the output has a matching graphic character in it, the conversion preserves the byte value and not the graphic character meaning.
- If there are unequal numbers of input and output code points (such as between the PC DBCS and host DBCS), round trip conversion is only possible from the smaller of the two sets to the larger set and back. There is insufficient coding space for all of the code points in the reverse direction. This situation also exists between ISO-7 encodings (without code extensions) and any 8-bit SBCS code.

When round trip mapping is not feasible or not desirable for a specific application, other criteria must be used.

Pairing of Code Points Using Round-trip

In addition to the general principles described in Pairings of Code Points, the following principles are used when the round-trip integrity criterion is chosen:

- An input graphic code point outside the common set is mapped to an output graphic code point outside the common set
- An input control code point is mapped to an output control code point outside the mnemonic-based common set
- If the graphic encoding space of the source is larger than the graphic encoding space of the target, some graphic code points will be mapped to control code points, and vice versa.

Character Replacements

When round trip integrity is not feasible or desired, an alternative is to permanently replace each mismatched character in the input character set with its nearest equivalent in the output character set. The criterion for determining the nearest equivalent depends on the context within which the converted data will be used. For display and printing purposes, the nearest visual representation may be chosen; for processing purposes, a character with the nearest meaning may be selected. If neither criterion applies, an arbitrary character may be chosen from the output character set.

Pairing of Code Points Using Character Replacements

In addition to the general principles described in Pairings of Code Points, the following additional principles are used when the character replacement criterion is chosen:

- An input graphic code point is mapped to an output graphic code point outside the common set with the nearest shape or meaning. Any remaining input graphic code points -- those with no nearest equivalent based on the criterion being used -- are mapped arbitrarily.
- An input control code point is mapped to an output control code point outside the mnemonic-based common set. Any remaining input control code points are arbitrarily mapped (folded) to other output control code points.
- Any round tripping achieved is incidental.

Enforced Subset Match

The enforced subset match criterion guarantees the preservation of the subset of characters that are common to both the input and output character sets. Any character not in this common subset will be replaced with a unique character that indicates that a substitution has occurred.

Wherever possible, CDRA recommends that the standardized control character SUB (substitute) be used for this purpose. Alternatives for "substitution character" may be

declared as part of the CCSID resource definitions. The default SUB definition for each CCSID is included as part of the CCSID definition found in Appendix C. CCSID Repository.

In environments using the PC-Data or PC-Display encoding structures, X'7F' is recommended as the default SUB. In single-byte EBCDIC environments, the defined SUB is X'3F', and in ISO-7 and ISO-8 environments it is X'1A'.

Visualization of SUB Character

The SUB character should be visually represented by a uniquely distinguishable character on presentation media. A warning flag should be returned to the caller of the mapping service to show that a substitution has occurred.

Default SUB-Visualization Character

Some presentation devices and data streams specify a unique character to be presented when a SUB code point is encountered in the presentation data. For example: the 3270 Data Stream defines a "filled circle" as default; the PC displays it as an "empty house symbol"; some printers print it as a "filled square".

When a presentation medium or a component interfacing to the presentation medium is not capable of replacing the SUB character with a unique non-SPACE visual character, the application sending data to be presented needs to convert the SUB character to an appropriate graphic character. For consistency among different implementations that do such a conversion, the Uppercase X (LX020000) (or its equivalent) is defined as the CDRA-recommended default.

Products that perform such SUB character replacement should also provide a means by which customers can select another graphic character of their choice as an alternative.

Pairing of Code Points Using Enforced Subset Match

In addition to the general principles described in Pairings of Code Points, the following additional principle is used when the enforced subset criterion is chosen:

All unmatched input graphic code points and mnemonically unmatched input control code points are converted to the "substitution character" code point prescribed for the output CCSID.

Conversion tables for CDRA level 2

Default conversion tables to be used for specific pairs of CCSIDs in different groups are available. For information on how to obtain these tables see Appendix J. CDRA Conversion Resources The pairs of CCSIDs are those that are required within each character set group, and include both interoperable and coexistence and migration sets.

Each table has its own difference management criterion. Where possible, the round-trip integrity criterion has been used; in other instances, enforcement and character replacement have been used.

Exceptions

The following exceptions to the basic mapping principles exist in some of the tables:

- In the SBCS-PC to EBCDIC tables for Group 2 countries, SO and SI Code Extension controls are substituted with substitution code point (X'3F') to avoid the potential risk of generating invalid SO-SI pairs.
- Some PC code pages such as 00850 and 00863 assign two code points for the symbols GCGID SM240000 and SM250000, in both graphic and control code range (PC-Data Encoding Scheme). Both symbols are included in Group 1 Interoperable Character Set 00697 and in the associated EBCDIC code pages. The other PC code pages derived from 00437 (for example, CP 00437, 00860) contain the symbols in control code range without duplication.
- The following rules are applied to the default tables in Group-1 and Group-1A Co-existance and Migration sets for SM240000 and SM250000:
  For Code Pages such as 00850 and 00863:
  The code points assigned to the symbols in the graphic code range (PC-Data Encoding Scheme) are treated as graphics. The code points in the control code range are treated as control code points, and are mapped based on the mnemonic names.
  For other PC code pages containing SM240000 and SM250000 only in control code range:
  The code points are treated as valid graphics, and are mapped based on GCGID when the symbols are included in the common graphic set between the CCSID pair. Otherwise, they are mapped based on the control mnemonic names.

Alternatives to conversion defaults

The default tables defined in CDRA are based on specified criteria for mismatch management. These tables may not suit all application requirements; IBM products have used different tables for data conversion based on the criteria most suited to

their customer. It may be necessary for the products to continue to support such tables.

Customers may have the need to continue using existing conversion tables or methods. Such methods or tables may produce conversion results that are different from those obtained using the default conversion tables.

Based on individual product and customer requirements, the ability to select alternative conversion methods or tables for a pair of CCSIDs may be supported by products as an option. If a product supports custom modifications, its documentation should describe the procedure for selecting the alternative method or table. Guidelines to prevent undesirable effects caused by such modifications should also be documented by the individual products.

Conversion functions

All the concepts described above can be incorporated into a collection of conversion methods and related conversion tables. The management aspects can also be embodied along with this collection. A single-step convert function and a three-part multiple-step conversion are defined in Chapter 5. CDRA interface definitions.

Chapter 7. CDRA Resources and Their Management

A CDRA resource is a collection of information that is needed by a CDRA function or used by several modules within a system in the correct processing of graphic character data in the system. CDRA resources can be machine representations of CCSID definitions, tables defining relationships between different CCSIDs, and the various tables associated with graphic-character-data conversions. A collection of resources of the same type is called a "resource repository".

The data structures of these resources are implementation-specific. This chapter defines the different elements (and their semantics) that these resources must contain. It includes definitions of the following resources:

- CCSID resource
- Graphic Character Conversion Table (GCCT) resource
- Graphic Character Conversion Selection Table (GCCST) resource
- Normalization Support CCSID Table (NSCT) resource
- Related Default CCSID Table (RDCT) resource

This chapter also describes the resource management considerations that must be given by implementations that support the CDRA resources.

Common Conventions

The elements of each resource are grouped into three categories:

Semantic Elements

Those elements that are required to complete the semantic definition of the identifier are listed and defined. Some of these elements must always be present, while others are conditional on contents of some other element. (For example, an ACRI-PCMB element is present only when the value of the ESID element of a CCSID resource is X'2300', X'2305' or X'3300'.)

Graphic Character String Elements

These elements consist of strings of graphic characters. They are mostly non-semantic in the sense that they are not required for completing the semantic definition of the identifier.

To provide consistency among different implementations, CDRA defines the format and maximum length values for these string elements. The two formats of this element are: short and long.

The short format is restricted to a maximum of 256 bytes consisting of:

Bytes	Meaning
1 - 16	reserved for implementation-specific use
17 - 18	the length of the string
19 - 20	CCSID of the string
21 - 256	character string

The long format is restricted to a maximum of 1024 bytes consisting of:

Bytes	Meaning
1 - 16	reserved for implementation-specific use
17 - 18	the length of the string
19 - 20	CCSID of the string
21 - 1024	character string

Other Elements

All other information associated with the identifier are lumped together under this category. They are mostly non-semantic in the sense that they are not required for completing the semantic definition of the identifier. Some of them may be required for some functions that depend on them for their success.

CCSID Resource

A CCSID resource is a machine representation of the elements associated with a particular CCSID value. A collection of CCSID resources is a CCSID resource repository. Some elements of a CCSID resource are accessed using call interfaces. The CCSID resource elements are summarized in Figure 19.

The following CCSID resource elements are defined in this section:

- Semantic Elements
- The following elements are required to define a CCSID resource:
    - CCSID
    - ESID
    - CS, CP pair
    - ACRI-List
        - ACRI-PCMB
        - ACRI-EUC
        - ACRI-TCP
- Other Elements
    - The following elements convey specific values associated with a CCSID that are of interest in different kinds of processing:
    - Control Function Definitions
        - SPACE Definition
        - SUB Definition
        - NEW LINE Definition
        - LINE FEED Definition
        - CARRIAGE RETURN Definition
        - END OF FILE Definition
    - F/M/S Indication

Semantic Elements of CCSID Resource

The CCSID resource elements described below are all required to completely define a CCSID and its associated long-form.

CCSID Element of CCSID Resource:

This element contains the value of the CCSID that this resource definition pertains to. It is used as the unique identifier of this resource. The CCSID value is used as the key to access this resource in many functions that get the individual elements of a CCSID. It is a number from 1 to 65,279 (X'0001' to X'FEFF'). All other values in the range 65,280 to 65535 (X'FF00' to X'FFFF') are reserved as special values and unlikely to appear in a CCSID resource. See the sections in chapter 3 on “Coded character set identifier”, “CCSID values”, and Figure 11 for more detailed information on these values and how they are used.

ESID Element of CCSID Resource:

This element contains the ESID value associated with the CCSID of this resource. Assigned values of ESID (in the range X'1100' to X'FFFE') are detailed in Figure 9 in chapter 3.

CS, CP Pair Element of CCSID Resource:

Depending on the ESID value, a CCSID is associated with one or more CS, CP pairs. This element contains the number of pairs, and the values of each CS, CP pair associated with the CCSID of this resource.

Most CCSIDs registered to date have a maximum of four CS, CP pairs with the exception of CCSIDs in support of Unicode which each have 18 CS, CP pairs (one defined for each of planes 0 – 16 plus one for the PUA area of the BMP). Each CP value is a number in the range 1 to 65,534 (X'0001' to X'FFFE'). Each CS value is a number in the range 1 to 65,535 (X'0001' to X'FFFF').

ACRI-List Element of CCSID Resource:

Depending on the ESID value, a CCSID has associated additional coding-related required information (ACRI) to make the definition of CCSID complete. The ACRI-List element identifies the number and types of ACRI needed and their definitions (see the section on “Additional coding-related required information” in chapter 3).

ACRI-PCMB Element:

ACRI-PCMB is required for CCSIDs having ESID values X'2300', X'2305' or X'3300' (PC mixed single-byte and double-byte encodings).

ACRI-EUC Element:

ACRI-EUC is required for CCSIDs having an ESID value X'4403'.

ACRI-TCP Element:

ACRI-TCP is required for CCSIDs having an ESID value X'5404'.

Element	Type and Value Range	Description	Used By
CCSID	A number in the range 1 to 65,279; see figure 11 in chapter 3.	Value of the CCSID that this resource definition pertains to. Unique identifier of this CCSID resource.	Most functions

The following are semantic elements associated with a CCSID. A CCSID definition is not unique without all these elements, where defined. The number and type of elements in this set are prescribed by the semantics associated with the ES id (see figure 9 in chapter 3).			
ESID	A number in the range 4352 to 65,534; see figure 9 in chapter 3.	The ESID element associated with this CCSID.	CDRGESP, CDRSCSP, and conversion methods
CS, CP	Pairs of numbers; CS can be in the range 1 to 65,535; CP can be in the range 1 to 65,534.	The number and values of CS, CP pairs associated with this CCSID. A CS value of 65,535 indicates that the maximal character set of the code page is defined within the installation's code page resource definition.	CDRGESP, and CDRSCSP
ACRI	Variable Lists	The type of ACRI, and values associated with that ACRI. The ACRI types defined in CDRA are identified here. Future ESIDs may have other types.	—
ACRI-PCMB	Number of ranges, pairs of From and To first-byte ranges; maximum 64 ranges; each first byte value is in the range 128 to 255.	ACRI-PCMB is valid only with CCSIDs using ESID X'2300', X'2305' or X'3300' (see section “ACRI PC mixed byte (ACRI-PCMB)” in chapter 3).	—
ACRI-EUC	Number of coded graphic character sets, width of each set. Maximum 5 values; first value is the number, subsequent values are corresponding widths.	ACRI-EUC is valid only with CCSIDs using ESID X'4403' (see section “ACRI Type EUC (ACRI-EUC)” in chapter 3).	—
ACRI-TCP	Number of coded graphic character sets, and a triplet for each set made up of the width of the set, the length of the designation escape sequence, and the actual escape sequence.	ACRI-TCP is valid only with CCSIDs using ESID X'5404' (see section “ACRI type TCP (ACRI-TCP)” in chapter 3).	
The following elements are "default" values associated with the CCSID. They are for use by different functions such as conversion methods that need to know the appropriate code points for "substitution", "space padding" or others. Informative elements such as F/M/S also belong in this group.			

SUB Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element	CDRGCTL, and some conversion methods
SPACE Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element	CDRGCTL, and some conversion methods
NL Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element. Meaningful only when the Encoding Scheme defines this control character NEW LINE.	CDRGCTL
LF Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element. Meaningful only when the Encoding Scheme defines this control character LINE FEED.	CDRGCTL
CR Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element. Meaningful only when the Encoding Scheme defines this control character CARRIAGE RETURN.	CDRGCTL
EOF Defn	Triplets of numbers: Code Point, Width of Code Point, and State Number	One triplet entry for each state (corresponding to each CS, CP pair) that appears in the CS, CP element. Meaningful only when the Encoding Scheme defines this control character END OF FILE.	CDRGCTL
F/M/S	0, 1 or 2	Value indicating if the CS associated with the CP in the CS/CP is a Full (2), Maximal (1), or Subset (0).	CDRSMXC

Figure 19. CCSID resource elements

Other Elements of CCSID Resource

In addition to the semantic elements described above, other information assigned to a CCSID may be queried. For example, a SPACE code point may be required for "SPACE-padding". When more than one code page is involved, there can be more than one SPACE code point associated with a CCSID. The default code points to be used for specific purposes, such as SPACE in this example, can be kept in the CCSID resource, to be accessed by functions as required.

The following elements are defined in this set:

SPACE Definition Element of CCSID Resource

The SPACE (GCGID SP010000) code point is used in conversion services; for example, to pad output strings for SPACE-padded string types. The code point to be used for a SPACE is usually reserved in an encoding scheme definition. However, there are cases where it is code-page-specific; for example, a wide space or double-byte SPACE (SP010080) in the case of CP 300.

When the ES associated with a CCSID specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definitions of SPACE code points are also defined by the encoding scheme semantics. The definition can be one of the following:

1. A SPACE code point value is defined only for some code pages in the list of CS, CP pairs associated with the CCSID. At least one of the code pages will have a SPACE code point defined. In this instance a state change may be required to access the SPACE code point. Where defined, it has the same code point width as its corresponding code page.

Figure 20 shows several examples of the SPACE definitions.

Current State	Value of SPACE code point (Hex)	Width of SPACE code point (number of bytes)	State in which SPACE code point is used	CP used for graphics (example)	Encoding Scheme
1	40	1	1	00500	EBCDIC Single Byte
1	20	1	1	00850	PC-Data Single Byte
1	20	1	1	00819	ISO-8 Single Byte
1	40	1	1	00290	Host Mixed (Japan)
2	4040	2	2	00300	Host Mixed (Japan)
1	20	1	1	01041	PC-Data Mixed (Japan)
2	8140	2	2	00301	PC-Data Mixed (Japan)

Figure 20. Example of SPACE definitions in CCSID resource

The SPACE Definition element of a CCSID resource (shown in Figure 21) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 20, ordered (similar to the CS, CP pairs associated with the CCSID) in ascending order by state. The State Number values start at 1 (the starting state). A zero in the State in which SPACE code point is used indicates that there is no SPACE definition entered in the CCSID resource for the current state.

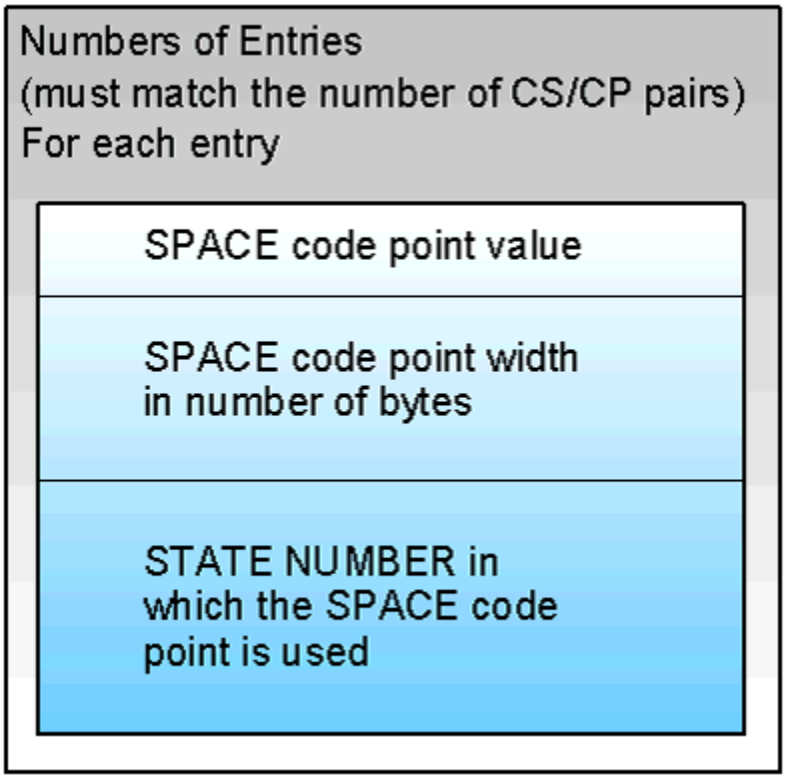


Figure 21. Space definition element

SUB Definition Element of CCSID Resource

The SUB code point is used in conversion services, when the chosen mismatch management criterion replaces all mismatched and invalid code points with a single character, SUB. The code point to be used for a SUB is usually reserved in an encoding scheme definition. However, there are many cases (such as PC-Display codes, where a graphic character code point has been chosen for a SUB indication) where it is code-page-specific.

When the ES associated with a CCSID specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definitions of SUB code points are also defined by the encoding scheme. The definition can be one of the following:

1. A SUB code point value is defined for each code page, having the same number of bytes as the code points in the code page. Whenever the switching mechanism (such as the SO and SI used in ES X'1301') used to switch between the different CS, CP pairs selects the code points from a particular code page, the SUB code point associated with that code page is to be used without any

change in the "state". During the creation of conversion tables, the SUB code point associated with a target code page is used, along with the state associated with that code page in a given CCSID. For example, the PC Mixed and Host Mixed encoding schemes each define one SUB code point for the single-byte code page and one for the double-byte code page. These SUB code point values can differ from one code page to another.

2. A SUB code point value is defined only for some code pages in the list of CS, CP pairs associated with the CCSID. At least one of the code pages will have a SUB code point defined. In this instance a state change may be required to access the SUB code point. Where defined, it has the same code point width as its corresponding code page.

Figure 22 shows several examples of the SUB definitions.

Current State	Value of SUB code point (Hex)	Width of SUB code point (number of bytes)	State in which SUB code point is used	CP used for graphics (example)	Encoding Scheme
1	3F	1	1	00500	EBCDIC Single Byte
1	7F	1	1	00850	PC-Data Single Byte
1	1A	1	1	00819	ISO-8 Single Byte
1	3F	1	1	00290	Host Mixed (Japan)
2	FEFE	2	2	00300	Host Mixed (Japan)
1	7F	1	1	01041	PC-Data Mixed (Japan)
2	FCFC	2	2	00301	PC-Data Mixed (Japan)

Figure 22. Example of SUB definitions in CCSID resource

The SUB Definition element of a CCSID resource (shown in Figure 23) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 22, ordered (similar to the CS, CP pair associated with the CCSID) in ascending order by state. The State Number values start at 1 (the starting state). A zero in the State in which SUB code point is used indicates that there is no SUB definition entered in the CCSID resource for the current state.

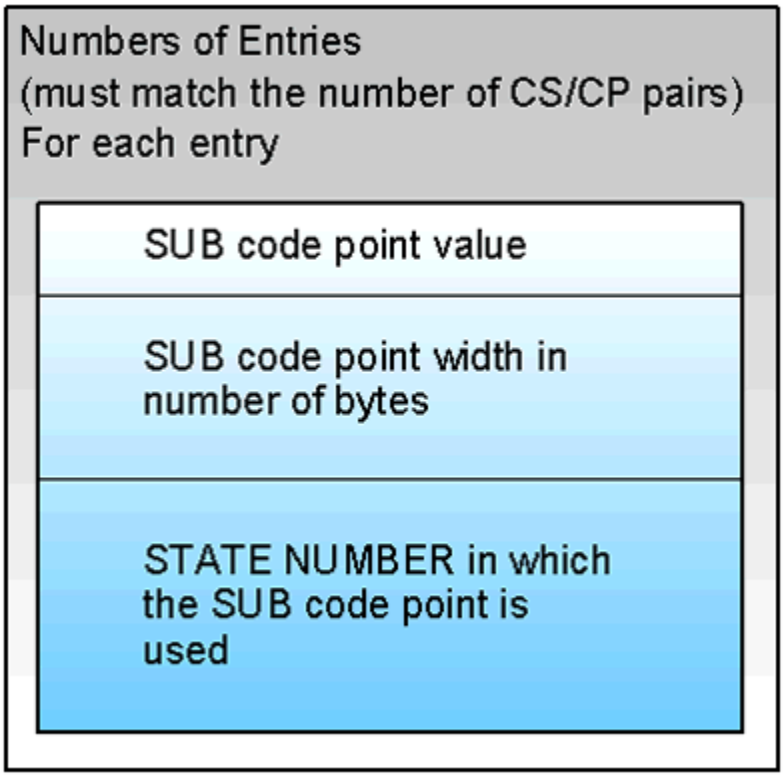


Figure 23. Sub definition element

NEW LINE Definition Element of CCSID Resource

The NEW LINE code point is used by parsing services to identify substrings of a file. The code point to be used for a NEW LINE is not defined for every CCSID, as it is encoding-scheme-dependent. When the ES associated with a CCSID specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definition is the following:

- Whenever the switching mechanism (such as the SO and SI used in ES X'1301') used to switch between the different CS, CP pairs is used, the NEW LINE code point associated with the single-byte code page is to be used. A state change and return may be required when using the NEW LINE definition.

Figure 24 shows several examples of the NEW LINE definitions.

Current State	Value of NEW LINE code point (Hex)	Width of NEW LINE code point (number of bytes)	State in which NEW LINE code point is used	CP used for graphics (example)	Encoding Scheme
1	15	1	1	00500	EBCDIC Single Byte
1	0D0A	2	1	00850	PC-Data Single Byte
1	—	1	1	00819	ISO-8 Single Byte
1	15	1	1	00290	Host Mixed (Japan)
2	15	1	1	00300	Host Mixed (Japan)
1	0D0A	2	1	01041	PC-Data Mixed (Japan)
2	0D0A	2	1	00301	PC-Data Mixed (Japan)

Figure 24. Example of NEW LINE definitions in CCSID resource

The NEW LINE Definition element of a CCSID resource (shown in Figure 25) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 24, ordered (similar to the CS, CP pairs associated with the CCSID) in ascending order by state. The State Number values start at 1 (the starting state). A zero in the State in which NEW LINE code point is used indicates that there is no NEW LINE definition entered in the CCSID resource for the current state.

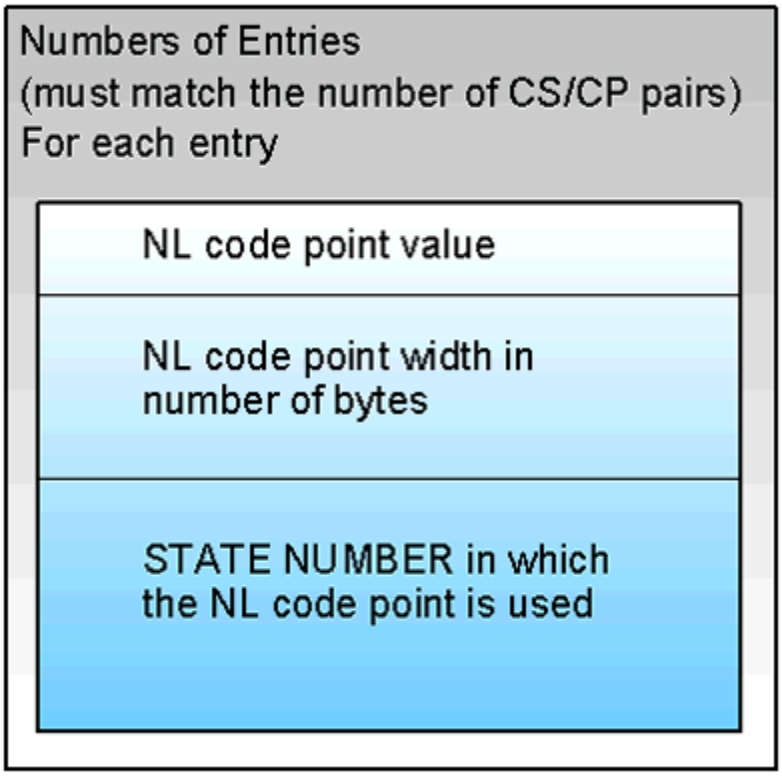


Figure 25. New Line definition element

LINE FEED Definition Element of CCSID Resource

The LINE FEED code point is used by parsing services to identify substrings of a file. The code point to be used for a LINE FEED is not defined for every CCSID, as it is encoding-scheme-dependent. When the ES associated with a CCSID specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definition is the following:

- Whenever the switching mechanism (such as the SO and SI used in ES X'1301') to switch between the different CS, CP pairs is used, the LINE FEED code point associated with the single-byte code page is to be used. A state change and return may be required when using the LINE FEED definition.

Figure 26 shows the LINE FEED definitions using several examples.

Current State	Value of LINE FEED code point (Hex)	Width of LINE FEED code point (number of bytes)	State in which LINE FEED is used	CP used for graphics (example)	Encoding Scheme
1	25	1	1	00500	EBCDIC Single Byte
1	0A	1	1	00850	PC-Data Single Byte
1	0A	1	1	00819	ISO-8 Single Byte
1	25	1	1	00290	Host Mixed (Japan)
2	25	1	1	00300	Host Mixed (Japan)
1	0A	1	1	01041	PC-Data Mixed (Japan)
2	0A	1	1	00301	PC-Data Mixed (Japan)

Figure 26. Example of LINE FEED definitions in CCSID resource

The LINE FEED Definition element of a CCSID resource (shown in Figure 27) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 26, ordered (similar to the CS, CP pairs associated with the CCSID) in ascending order by state. The State Number values start at 1 (the starting state). A zero in the State in which LINE FEED code point is used indicates that there is no LINE FEED definition entered in the CCSID resource for the current state.

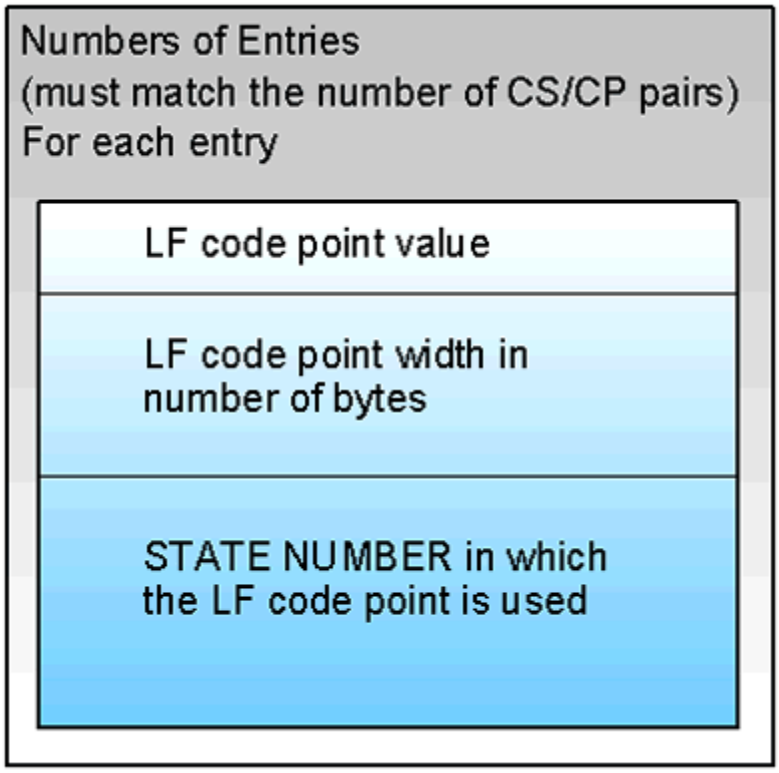


Figure 27. Line Feed definition element

CARRIAGE RETURN Definition Element of CCSID Resource

The CARRIAGE RETURN code point is used by parsing services to identify substrings of a file. The code point to be used for a CARRIAGE RETURN is not defined for every CCSID, as it is encoding-scheme-dependent. When the ES associated with a CCSID, specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definition is the following:

- Whenever the switching mechanism (such as the SO and SI used in ES X'1301') to switch between the different CS, CP pairs is used, the CARRIAGE RETURN code point associated with the single-byte code page is to be used. A state change and return may be required when using the CARRIAGE RETURN definition.

Figure 28 shows several examples of the CARRIAGE RETURN definitions.

Current State	Value of CARRIAGE RETURN code point (Hex)	Width of CARRIAGE RETURN code point (number of bytes)	State in which CARRIAGE RETURN code point is used	CP used for graphics (example)	Encoding Scheme
1	0D	1	1	00500	EBCDIC Single Byte
1	0D	1	1	00850	PC-Data Single Byte
1	0D	1	1	00819	ISO-8 Single Byte
1	0D	1	1	00290	Host Mixed (Japan)
2	0D	1	1	00300	Host Mixed (Japan)
1	0D	1	1	01041	PC-Data Mixed (Japan)
2	0D	1	1	00301	PC-Data Mixed (Japan)

Figure 28. Example of CARRIAGE RETURN definitions in CCSID resource

The CARRIAGE RETURN Definition element of a CCSID resource (shown in Figure 29) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 28, ordered (similar to the CS, CP pairs associated with the CCSID) in ascending order by state. The State Number values start at 1 (the starting state). A zero in the State Number in which CARRIAGE RETURN code point is used indicates that there is no CARRIAGE RETURN definition entered in the CCSID resource for the current state.

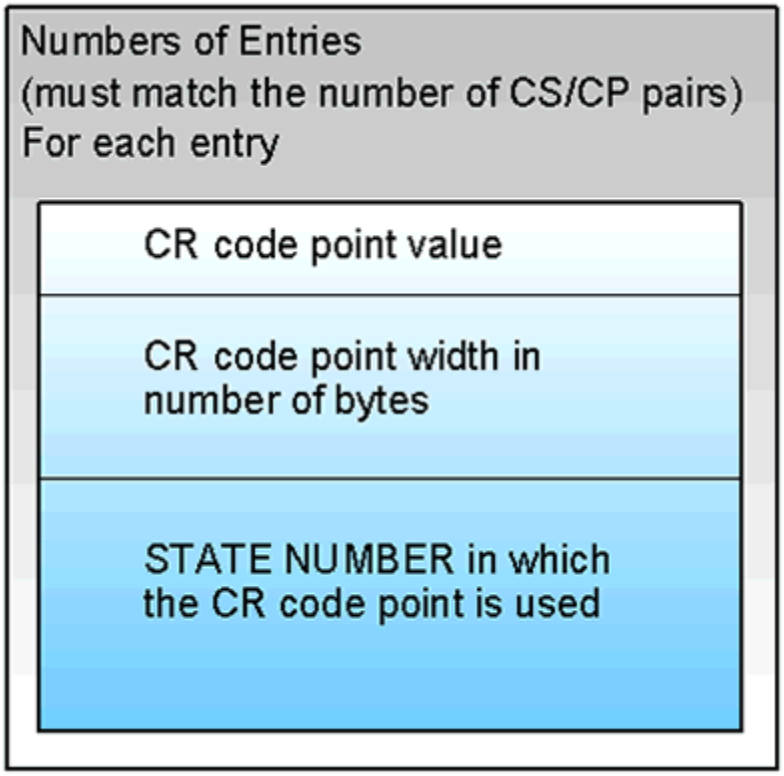


Figure 29. Carriage Return definition element

END OF FILE Definition Element of CCSID Resource

The END OF FILE code point is used by parsing services to identify substrings of a file. The code point to be used for a END OF FILE is not defined for every CCSID, as it is encoding-scheme-dependent. When the ES associated with a CCSID specifies more than one CS, CP pair (multiple states using explicit or implicit code extension techniques), the definition is the following:

- Whenever the switching mechanism (such as the SO and SI used in ES X'1301') to switch between the different CS, CP pairs is used, the END OF FILE code point associated with the single-byte code page is to be used. A state change and return may be required when using the END OF FILE definition.

Figure 30 shows several examples of the END OF FILE definitions.

Current State	Value of END OF FILE code point (Hex)	Width of END OF FILE code point (number of bytes)	State in which END OF FILE code point is used	CP used for graphics (example)	Encoding Scheme
1	1C	1	1	00500	EBCDIC Single Byte
1	1A	1	1	00850	PC-Data Single Byte
1	1A	1	1	00819	ISO-8 Single Byte
1	1C	1	1	00290	Host Mixed (Japan)
2	1C	1	1	00300	Host Mixed (Japan)
1	1A	1	1	01041	PC-Data Mixed (Japan)
2	1A	1	1	00301	PC-Data Mixed (Japan)

Figure 30. END OF FILE definitions in CCSID resource

The END OF FILE Definition element of a CCSID resource (shown in Figure 31) contains the number of entries, and the information contained in the second, third, and fourth columns of Figure 30, ordered (similar to the CS, CP pairs associated with the CCSID) in ascending order by state. A zero in the State in which END OF FILE code point is used indicates that there is no END OF FILE definition entered in the CCSID resource for the current state.

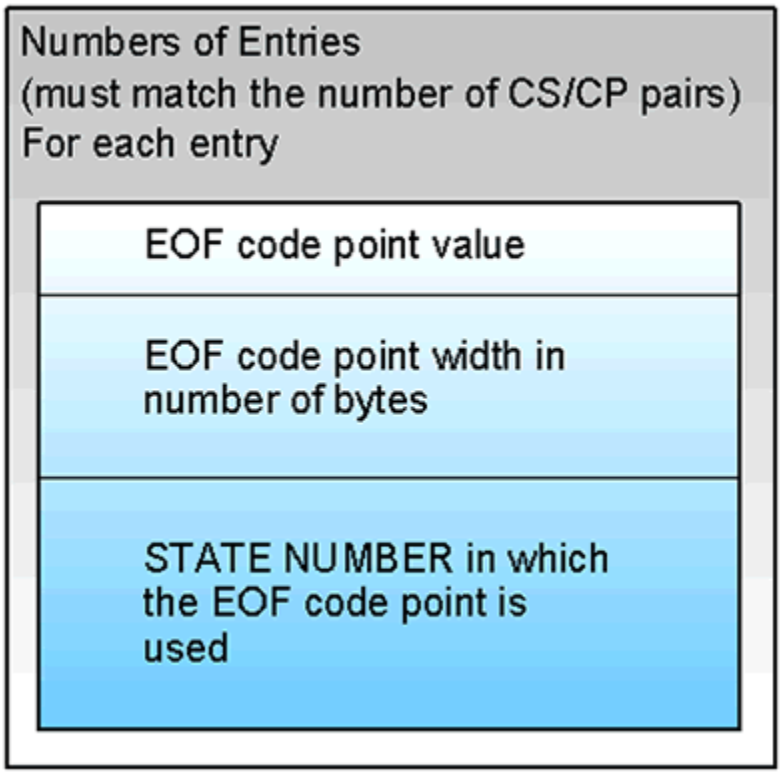


Figure 31. END OF FILE definition element

Graphic Character Conversion Table (GCCT) Resource

In the CDRA model, this resource is assumed to be a repository of the supported conversion tables. It is used by the following CDRA-defined services:

- CDRCVRT
- CDRMSCI

Appendix J, CDRA Conversion Resources, describes how users can obtain the conversion tables defined in support of this architecture. The convert functions defined in "Functions Related to Difference Management" use these tables to perform the conversion. The conversion tables are represented in the machine in a suitable format for the conversion methods implemented in each system. These machine representations are "conversion table resources", and a collection of these is a "conversion table resource repository".

In the CDRA model, this resource is assumed to be a repository of the supported conversion tables. It is used by the following CDRA-defined services:

The various elements of a GCCT resource are shown below:

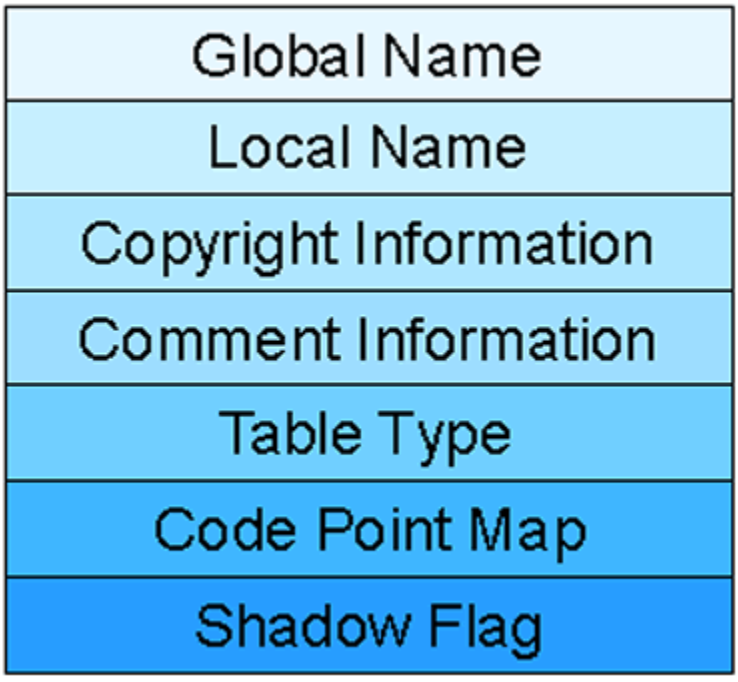


Figure 32. Graphic character conversion table (GCCT) resource

These elements can be divided into two groups -- semantic elements and graphic character string elements.

Semantic Elements of GCCT Resource

The semantic elements of the GCCT resource are:

GCCT Table Type Element:

The CDRA conversion table registry, located on the CD included with this document, details a variety of conversion tables including single-byte to single-byte, double-byte to double-byte, single-byte to double-byte and others. Each of the various table types are described in Appendix B, "Conversion Methods", as are the conversion methods which use them.

The table type element is used to identify the following:

- The organization and interpretation of the code point map element.
- The organization and interpretation of the shadow flag resource element.

GCCT Code Point Map Element:

The contents of the conversion tables from the CDRA registry are entered in this element in a format suitable for the implementing system.

The structure and contents of this element can vary from being an array of 256 single-bytes (for single- to single-byte map of type 1), to a collection of controlling sub tables, a sub pool of single- to single-byte maps, and a sub pool of single- to double-byte maps, to support Method 4 (mixed single-byte and double-byte conversions) (see Appendix B. "Conversion Methods").

The structure of this element is implementation-specific.

GCCT Shadow Flag Element:

The CDRA Registry uses a shadow flag technique to indicate that a graphic character substitution (with another graphic character or with a SUB) has been made in the conversion tables. When the conversion function supports issuing a feedback code when such substitutions are detected, the shadow flag element must be available to provide this information. It captures the indications such as: a character has been replaced, substituted, or dropped, for each code point pairing, to supplement any algorithmic method used to check for such conditions in the associated conversion

method. Figure 71 in Appendix B shows an example of how the shadow flag element is used with a conversion method.

The structure and complexity of this element corresponds to that of the code point map element described above.

Graphic Character String Elements of a GCCT Resource

Products or service functions may require access to GCCT descriptive information such as names or copyright information. This information is in graphic character string elements, encoded in an identified CCSID. Except for the Global Name, the contents of these elements are implementation-specific.

The following graphic character string elements are defined for a GCCT resource:

Global Name Element of a GCCT Resource:

The global name element of a GCCT resource is a CDRA-defined string associated with each GCCT. This element consists of the length, the CCSID used for encoding, and the string representing the global name. The string will be encoded using one of the global use CCSIDs, using the syntactic character set of CS 00640. If a system cannot support the lowercase a through z, these characters will be mapped to the corresponding uppercase A through Z.

The global name is used wherever there is a need to display a GCCT as a globally readable and understood string of graphic characters.

This element is a short format Graphic Character String Element as defined above in common conventions.

Local Name Element of a GCCT Resource:

The local name element is a string of graphic characters representing the local name assigned to the GCCT resource defined within a system installation. It will be encoded using one of the CCSIDs supported in the system.

This element is a short format Graphic Character String Element as defined above in common conventions.

Copyright Information Element of a GCCT Resource:

The copyright information element of a GCCT resource is a string of graphic characters that detail any copyright on the associated GCCT contents. If this string exists it should be presented to the end user whenever information about this GCCT is presented, according to the current information asset protection practices. The string may be in whatever national language is most suited to that installation, and encoded in an appropriate CCSID.

This element is a short format Graphic Character String Element as defined above in common conventions.

Comments Information Element of a GCCT Resource:

The comments information element of a GCCT resource is a string of graphic characters that conveys any descriptive information that will be useful to the end user and is associated with the conversion table in the GCCT resource.

This element is a long format Graphic Character String Element as defined above in common conventions.

Graphic Character Conversion Selection Table (GCCST) Resource

This resource is used to access the correct conversion method and conversion tables corresponding to the parameters that are associated with the input string to be converted and with the output string to be created. This resource is used by the following common service functions:

- CDRCVRT

In addition to the selection table contents corresponding to Figure 35, the GCCST resource has a Local Name and Comments Information elements.

Local Name Element of a GCCST Resource:

The local name element is a string of graphic characters representing the local name assigned to the GCCST resource defined within a system installation. It will be encoded using one of the CCSIDs supported by the installation.

This element is a short format Graphic Character String Element as defined above in common conventions.

Comments Information Element of a GCCST Resource:

The comments information element of a GCCST resource is a string of graphic characters that conveys any descriptive information associated with the GCCST resource.

This element is a long format Graphic Character String Element as defined above in common conventions.

The contents of a GCCST resource depend on the conversion services supported, the set of methods, and associated tables, in an installation. They must be alterable to reflect the support in each installation.

Figure 33 illustrates a model called the Graphic Character Conversion Selection Table (GCCST). In this model there is an entry for every supported conversion alternative between each pair of From-CCSID and To-CCSID. The parameters needed to uniquely identify an entry are the From-CCSID, the From-ST, the To-CCSID, the To-ST, and the Graphic Character Conversion Alternative Selection Number (GCCASN).

A conversion function will use the input From-CCSID and To-CCSID values, the From-ST and To-ST values, and the GCCASN to select the conversion method and the associated conversion tables needed. These will be used to perform the conversion, selecting the installation default alternative when necessary. The conversion function may contain a method, or may access a method provided elsewhere (via a known call interface). The columns in this table are defined as follows:

- The From-CCSID column contains the value of the "From" CCSID within which the input string to be converted is represented
- The From-ST column contains the value of the string type of the input string. See the section, “Types of Strings” in chapter 6 for a complete list of defined string types.

ST	Type of String
0	A Graphic Character String, as semantically defined by CCSID.
1	A Graphic Character String, as semantically defined by CCSID, and null-terminated.

- # The To-CCSID column contains the value of the "To" CCSID within which the converted string is represented
- The To-ST column contains the value of the string type desired for the converted string. See "Types of Strings".

ST	Type of String
0	A Graphic Character String, as semantically defined by CCSID.
1	A Graphic Character String, as semantically defined by CCSID, and null-terminated.
2	A Graphic Character String, as semantically defined by CCSID, and SPACE-padded.

- The GCCASN column contains the GCCASN assigned to this entry. This number is used to differentiate between alternatives of conversion for a given (From-CCSID, ST) - (To-CCSID, ST) combination. The alternatives differ from one another if:
    - The conversion method is different and/or
    - At least one of the conversion tables used is different.

The following alternative numbers are defined for the model:

Value	Nature of the Conversion Alternative selected
0	not valid as an entry in this column
1	is used to select the CDRA-defined default method and associated conversion table(s). The difference management criterion used in the creation of the selected tables is based on country requirements to serve the majority of applications using the selected CCSID pairs.
2 to 9	are reserved for future allocation by CDRA
10 to 55	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the round-trip mismatch management criterion.
56 to 101	are reserved to select other CDRA-defined alternatives; each conversion table selected is created using the enforced subset mismatch management criterion.
102 to 147	are reserved to select other CDRA-defined alternatives. These alternatives may include conversions where: • the mismatch management criterion used in creating any of the selected tables is other than round trip or enforced subset • more than one conversion table is selected and unequal criteria have been used when creating the different tables.
148 to 255	are reserved for selecting customer-defined alternatives. A customer organization may establish and control ranges of GCCASN to distinguish between different mismatch

Value	Nature of the Conversion Alternative selected
	management criteria, similar to the IBM-defined ones described above.

Note: The value of 0 for GCCASN can only be used as a parameter in a function call to a convert function. If a value of 0 is received, the conversion selection logic (that uses this model) will scan the Def column instead of the GCCASN column, and select the alternative that is marked as installation default (a 1 in the Def column). For all other values of GCCASN, the selection is made by comparing the non-zero input GCCASN value with the entries in the GCCASN column of the GCCST.

Note: The value of 0 for GCCASN can only be used as a parameter in a function call to a convert function. If a value of 0 is received, the conversion selection logic (that uses this model) will scan the Def column instead of the GCCASN column, and select the alternative that is marked as installation default (a 1 in the Def column). For all other values of GCCASN, the selection is made by comparing the non-zero input GCCASN value with the entries in the GCCASN column of the GCCST.

- The Def column contains an entry "0" or "1". A "1" is used to show that this alternative is the system or installation default for the given From-CCSID To-CCSID pairing. This alternative can be selected either by specifying the GCCASN associated with it, or by specifying a GCCASN value of 0 indicating that the installation default alternative should be used when the conversion function is called. When there is more than one alternative for a given pair of CCSIDs, only one of these alternatives will have a "1" in this column. Entries for the remaining alternatives will have a "0" in this column.
- The Method column shows the conversion method required (see Appendix B. “Conversion Methods”). It must be compatible with the CCSIDs and the string types shown in the appropriate column. Two or more conversion methods may be used for some (CCSID, ST) pairs, depending on the criterion used for difference management and assumptions made about the input strings.
- In the assumed model, the string types From-ST and To-ST are passed as parameters to the method selected. These parameters are in turn used by the method for parsing the input string and assembly of the output string during conversion.
- The Number of Tables column shows the number of conversion tables that are required by the method indicated in the Method column.
- The Local Table Name column shows the local names of the required conversion tables to be used with the method for this alternative. The local names -- the structure and any naming conventions or constraints -- are

implementation-dependent. If the method selected requires more than one table, it is shown as having more than one table name in this column. The table name must be unique within the sphere of control of the graphic character conversion management process. The contents of the table follow the model specified by the table type value, to match the capability of the conversion method selected.

- The Table Type column is for information only; it shows the type of table (see Appendix B. “Conversion Methods”) that must be used with the method selected. If more than one table is needed, the type of each table is identified. (Note: An implementation may choose to use this information to cross-check if the table selected is of the appropriate type, and issue a warning to the caller.)
- The Remarks column contains some comments and references for this architecture document, such as:
    - How the same conversion table can be re-used
    - How an installation default is marked
    - Where the GCCASN is used.

From CCSID	Fr ST	To CCSID	To ST	GCCASN	Def	Method	Number of Tables	Local Table Name	Tab Type	Remarks
Note: This table is just a model used to describe the management related aspects of accessing and selecting the appropriate conversion method and associated conversion table(s) for graphic character data conversion.										
Note: Appendix B. Conversion Methods describes models for different conversion methods and associated conversion tables of different types,										
00500	0	00037	0	1	1	Method 1	1	TT00001	1	See Method 1 for SBCS and figure 69 in Appendix B
00500	0	00037	0	2	1	Method x	1	TT00005	1	This entry indicates that Method x and the table with the local name TT00005 should be used to convert as selected,

Figure 33. Sample Graphic Character Conversion Selection Table (GCCST)

Normalization Support CCSID Table (NSCT) Resource

This resource is used by the following CDRA-defined function:

- CDRGCCN

The various elements of an NSCT resource are shown below.

Figure 34. Normalization Support CCSID Table (NSCT) Resource

Local Name Element of an NSCT Resource:

The local name element of an NSCT resource is a string of graphic characters representing the local name assigned to the NSCT resource. It will be encoded using one of the CCSIDs supported in the system.

This element is a short format Graphic Character String Element as defined above in common conventions.

Copyright Information Element of a NSCT Resource:

The copyright information element of a NSCT resource is a string of graphic characters that represents any copyright on the associated NSCT contents. If this string exists it should be presented to the end user whenever information about this NSCT is presented, according to the current information asset protection practices. The string may be in whatever national language is most suited to the installation, and encoded in an appropriate CCSID.

This element is a short format Graphic Character String Element as defined above in common conventions.

Comments Information Element of an NSCT Resource:

The comments information element of an NSCT resource is a string of graphic characters that conveys any descriptive information associated with the NSCT resource.

This element is a long format Graphic Character String Element as defined above in common conventions.

In addition to the Character String Elements, the NSCT resource contains a number of rows containing four values:

- CCSID1 (first CCSID of an input pair to be matched)
- CCSID2 (second CCSID of an input pair to be matched)
- CCSID for normalization
- Hint (CCSID relationship information)

Related Default CCSID Table (RDCT) Resource

This resource is used to provide a predetermined CCSID for an expected input CCSID. This resource is used by the following CDRA-defined service:

- CDRGRDC

The various elements of an RDCT resource are shown below.

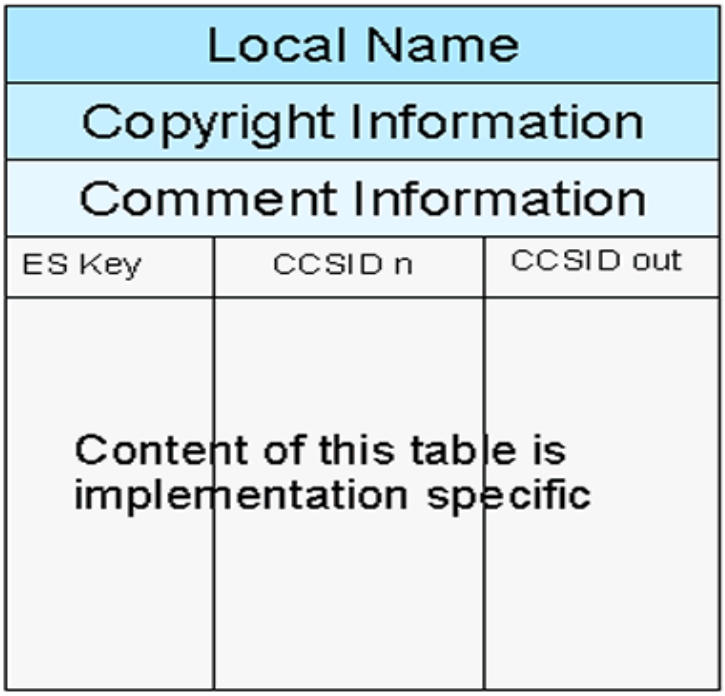


Figure 35. Related Default CCSID Table (RDCT) Resource

Local Name Element of an RDCT Resource:

The local name element of an RDCT resource is a string of graphic characters that is defined within a system installation, referring to the RDCT. It will be encoded using one of the CCSIDs supported in the system.

The local name element consists of the length, the CCSID used for encoding, and the string of characters representing the local name assigned to the RDCT resource.

This element is a short format Character String Element as defined above in common conventions.

Copyright Information Element of an RDCT Resource:

The copyright information element of an RDCT resource is a string of graphic characters that represents any copyright on the associated RDCT contents. The string may be in whatever national language is most suited to that installation, and encoded in an appropriate CCSID.

The copyright information element consists of the length, the CCSID used for encoding, and the string of characters representing the copyright information assigned to the conversion table contents of the resource.

This element is a short format Graphic Character String Element as defined above in common conventions.

Comments Information Element of an RDCT Resource:

The comments information element of an RDCT resource is a string of graphic characters that conveys any descriptive information associated with the RDCT resource.

The comments information element consists of the length, the CCSID used for encoding, and the string of characters representing the comment information that is related to the RDCT resource.

This element is a long format Character String Element as defined above in common conventions.

A model of a Related Default CCSID Table is shown in Figure 36. RDCT is the primary resource supporting the function CDRGRDC (see "CDRGRDC - Get Related Default CCSID" for more details on how RDCT is used). The columns in the table are described below:

Key ES (hex): this value is the encoding scheme (hex) that the user requires for the returned CCSID

ES of CCSID-in (hex): this is the hex value of the encoding scheme of the input CCSID. (This value is not required in an RDCT but is included to assist in understanding the sample data.)

CCSID-in: this is the input CCSID. It is the value for which a related default is being requested.

CCSID-out: this is the output CCSID. It is the CCSID determined by the implementation to be the most appropriate CCSID with an encoding scheme of ES.

The model consists of pairs of CCSID values, organized with the ES of the output CCSID as the primary key. The CCSID-in is used as the secondary key to determine the CCSID-out.

The entries in this table are sample data only. The contents of an actual table on a system are implementation specific.

Key ES (hex)	ES of CCSID-in (hex)	CCSID-in	CCSID-out	Comments
1100	1100	00500	00500	Here a user is looking for a CCSID with an encoding scheme of 1100 which is a related default for CCSID 500. The table lookup returns CCSID 500, indicating that for this implementation CCSID 500 should be used.
1100	1100	01027	00290	Here a user is looking for a CCSID with an encoding scheme of 1100 which is a related default for CCSID 1027. The table lookup returns CCSID 290, indicating that for this implementation CCSID 290 should be used.
1100	2100	00850	00500	In this case the user is looking for an EBCDIC CCSID (ES 1100) which is a related default for the PC CCSID 850 (ES 2100). The table lookup returns CCSID 500. Thus 500 is the EBCDIC CCSID identified by this implementation as 'best related' to CCSID 850.
1100	2100	00874	00838	The table shows that in this implementation the EBCDIC CCSID (ES 1100) best related to PC CCSID 874 (ES 2100) is CCSID 838.
1100	4100	00819	00500	The table shows that in this implementation the EBCDIC CCSID (ES 1100) best related to the ISO-8 CCSID 819 (ES 4100) is CCSID 500.

Figure 36. Model of a Related Default CCSID Table

CDRA Resource Management

Any implementation of CDRA resources will also require services to maintain and manage them. The implementation and management of these resources are implementation specific.

Appendix A. Encoding Schemes

This appendix contains descriptions of the encoding structures defined by CDRA encoding schemes. The information is a summary of the definitions taken from relevant standards or system documentation.

EBCDIC Single-byte Structures

IBM Extended Binary Coded Decimal Interchange Code (EBCDIC) is based on an 8-bit-per-byte structure. The basic EBCDIC structure is shown in Figure 37.

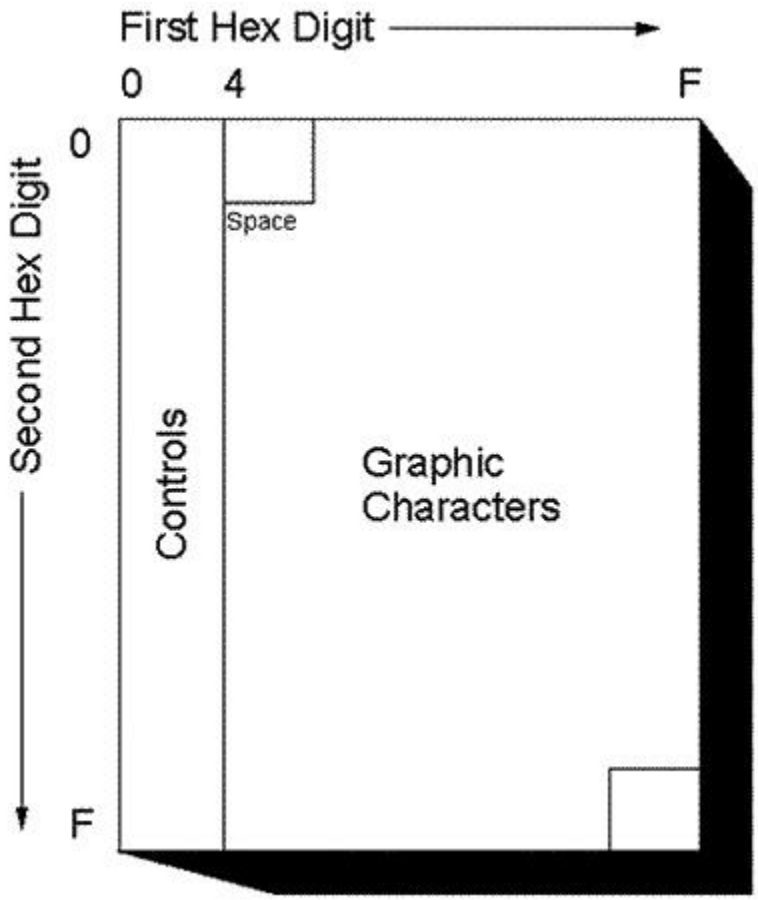


Figure 37. Basic Structure of EBCDIC Code (ES=X'1yzz')

EBCDIC structure implies the following:

- The valid code point range is from X'00' to X'FF'.
- Code points in the range X'00' to X'3F' are reserved for control characters.

- A control function can be a single control character, or a sequence of characters beginning with one of the appropriate control characters. The first character of a control sequence must be a control character. However, the parameters of the control sequence can take values from the entire coding space.
- X'FF' is always assigned the character EIGHT ONES (EO).
- X'40' is reserved for the SPACE character.
- Code points in the range X'41' to X'FE' are reserved for representing graphic characters.
- EBCDIC definition also permits use of more than one byte per code point to represent graphic characters (see Figure *43*).In a single-byte (coded) character set (SBCS) EBCDIC code, a maximum coding space of 190 octets (X'41' to X'FE') is available for assigning to graphic characters.
- The EBCDIC standard also prescribes the invariant code points allocated to the syntactic character set CS 640, when the character set is coded in EBCDIC SBCS codes (see Figure 43). However, it should be noted that there are some EBCDIC-coded character sets (notably the Katakana and Extended Katakana codes used in Japan) that do not follow this property of invariance.

The EBCDIC Presentation Structure

IBM Extended Binary Coded Decimal Interchange Code (EBCDIC) for presentation is based on an 8-bit-per-byte structure. The basic EBCDIC presentation structure is shown in Figure 38.

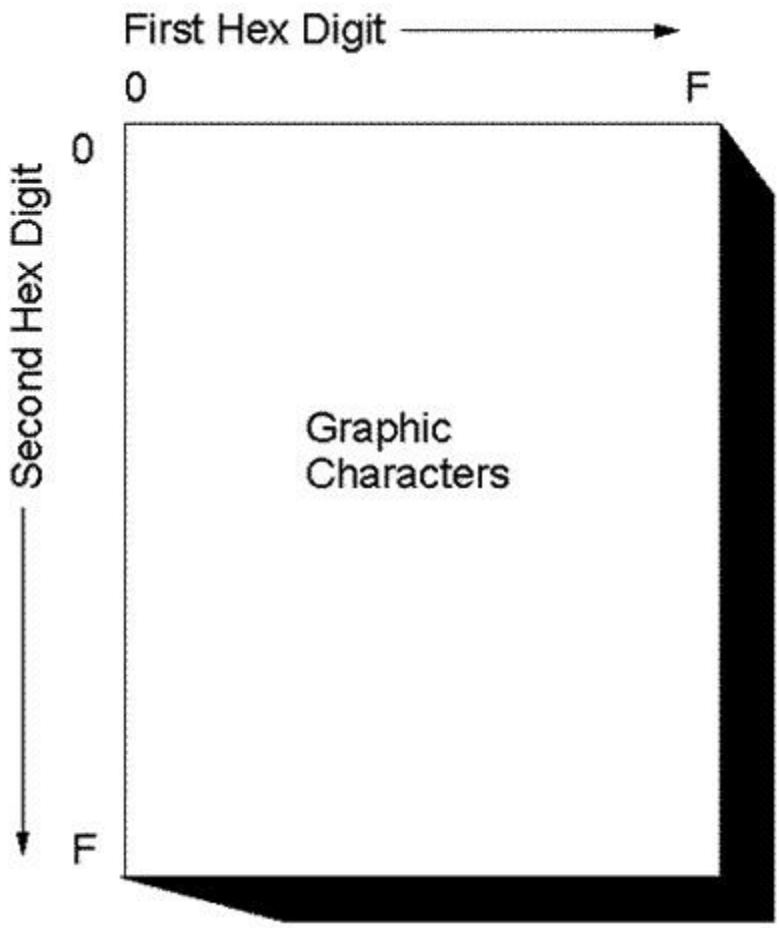


Figure 38. Basic Structure of EBCDIC Presentation Code (ES=X'6100)

Some products have modified the EBCDIC structure for presentation purposes. The following describes the semantics of this encoding structure.

All the code points in the range X'00' to X'FF' are assignable to graphic characters with the following considerations:

- If a character from CS 640 is included in the character set being encoded, it will be assigned a code point respecting the invariance properties of CS 640 in the EBCDIC encoding structure shown in Figure 37.
- The SPACE (SP010000) character can be assigned to a position other than the traditional EBCDIC location of X'40'.

IBM PC Single-byte Structures

IBM-PC structure is an extension of the ISO 646 (ANSI version) 7-bit code structure to an 8-bit structure. Unlike the EBCDIC and ISO structures, this structure is ill-defined,

especially in distinguishing control character codes and graphic character codes in a context-independent manner.

The valid hexadecimal codes are in the range X'00' to X'FF'. When the codes are used to represent graphic characters on displays, all the code points are allocated for graphic characters. The range X'00' to X'1F' is reserved for control characters, following the ISO 646 scheme, except for the code points X'14' and X'15', which are used for graphic characters in some PC codes. Two basic structures, called IBM-PC Data Code and IBM-PC Display Code, are described below.

More than one byte per code point can also be used with the IBM-PC structures (see IBM-PC Data Code Structure).

IBM-PC Data Code Structure

The IBM-PC Data code is shown in Figure 39.

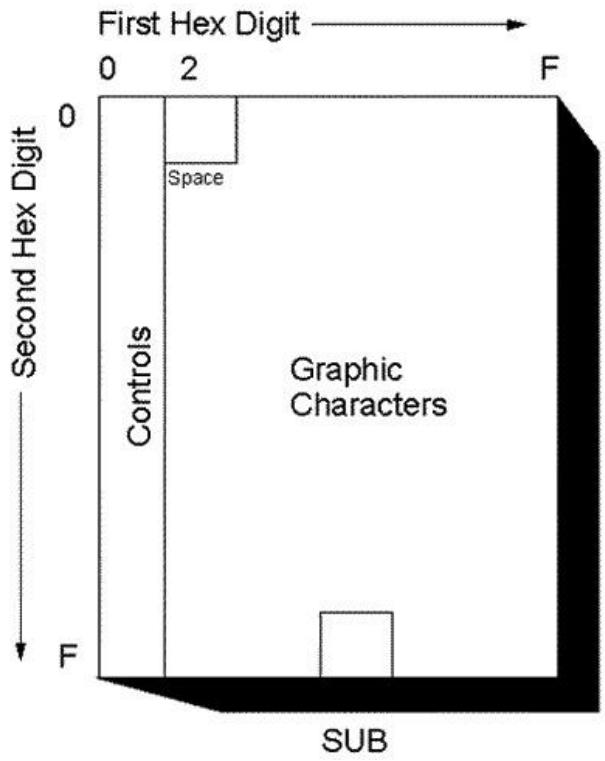


Figure 39. IBM-PC Data Code Structure (ES=X'2yzz')

It has the following characteristics:

- The range X'00' to X'1F' is reserved for control characters. Control functions can be single characters or sequences. A control sequence must begin with a control character. The parameters in the sequence can be any hexadecimal value. Of these, X'1A' is allocated to the "END OF FILE" marker character.
- X'7F' is allocated the SUBSTITUTE (SUB) control character used to show or map invalid or mismatched characters.
- X'20' is reserved for the SPACE character.
- X'21' to X'7E' and X'80' to X'FF' are assignable to graphic characters. A total of 222 code points are available to represent graphic characters in a single-byte PC Data code.
- The PC Data codes assign invariant code points allocated to the syntactic character set CS 640 (like ISO 646), when the character set is coded in PC Data SBCS codes (see Figure 43). In PC and ISO 646 there is one more character, the exclamation point, which is also assigned an invariant code point.

IBM-PC Display Code Structure

The IBM-PC Display Code, shown in Figure 40, has the following characteristics:

- X'01' to X'1F' and X'21'X'FF' are assignable to graphic characters. A total of 254 graphic code points are available in a single-byte IBM-PC Display code.
- X'00' is reserved for the control character NUL.
- X'20' is reserved for the SPACE character.
- Similar to ISO 646, the PC Display codes assign invariant code points allocated to the syntactic character set CS 640, when the character set is part of the character set coded as PC Display single-byte code (see Figure 43). In PC and ISO 646 there is one more character, the exclamation point, which is also assigned an invariant code point.

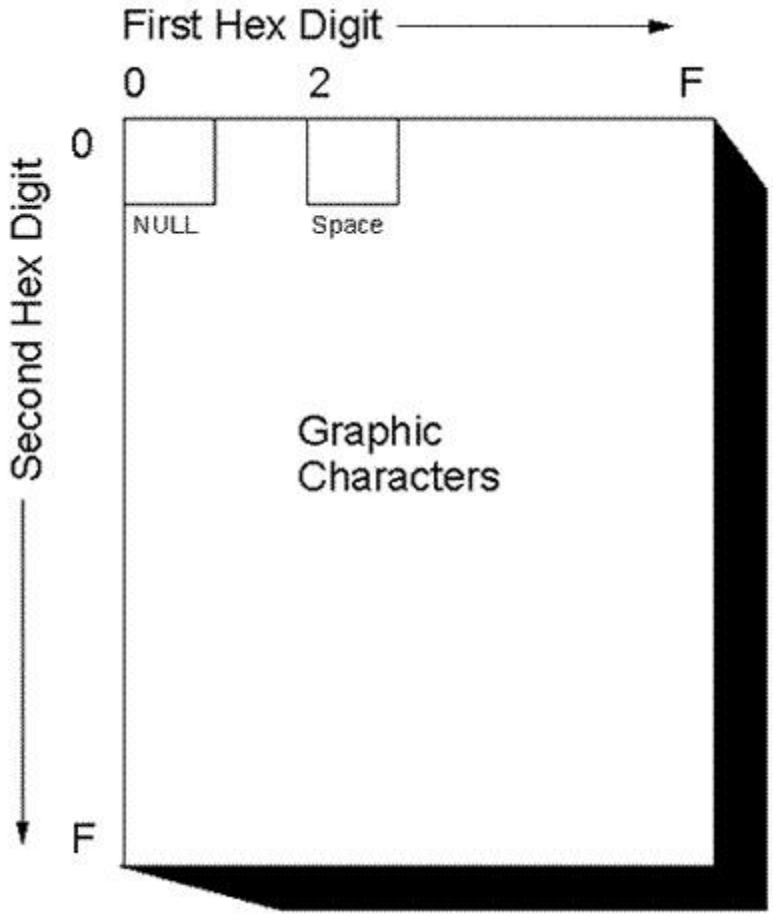


Figure 40. IBM-PC Display Code Structure (ES=X'3yzz')

ISO Single-byte Structures

The international standard ISO 2022, Information Processing - ISO 7-bit and 8-bit Coded Character Sets - Code Extension Techniques specifies the general structures and code extension schemes in the ISO 2022 environments. Other ISO standards, such as ISO 646, ISO 4873, ISO 6429, ISO 6937, and ISO 8859, define further specific use of subsets of the environments prescribed by ISO 2022. CCITT recommendations on Telematics, such as T.61 and T.100, also use ISO 2022 techniques. (American Standard Code for Information Interchange, ASCII, is the US national version of ISO 646 code; it is defined in the ANSI X3.4 standard.)

There are other encoding schemes outside ISO 2022, such as in the International Telegraphic Alphabet Number 2 (ITA2), a 5-bit code with an Alpha-shift and a Numeric-shift, which is used in international Telex services. Picture coding is another example. ISO 2022 has defined a scheme to switch to such non-ISO 2022 codes.

ISO 7-bit Structure

The ISO 7-bit structure (see Figure 41) is characterized by:

- Septet (7-bit byte) values can range from X'00' to X'7F'.
- X'00' to X'1F' are reserved for control characters. The set of control characters assigned to this range of code points is called a C0 set.
- X'21' to X'7E' are assignable to graphic characters. The set of graphic characters assigned to this range is called a G0 set.
- X'20' is reserved for the *SPACE* character if it is part of the last invoked character set.
- X'7F' is reserved for the *DELETE* character if it is part of the last invoked character set.
- The registered ISO-7 G sets that are based on ISO 646 assign invariant code points allocated to the syntactic character set CS 640 when the character set is part of the character set in the G set (see Figure 43). There is one more character, the exclamation point, which is also assigned an invariant code point.
- ESID X'5150' is used to describe ISO 7-bit Presentation code where graphic characters are found in the C0 control space (X'00' to X'1F").

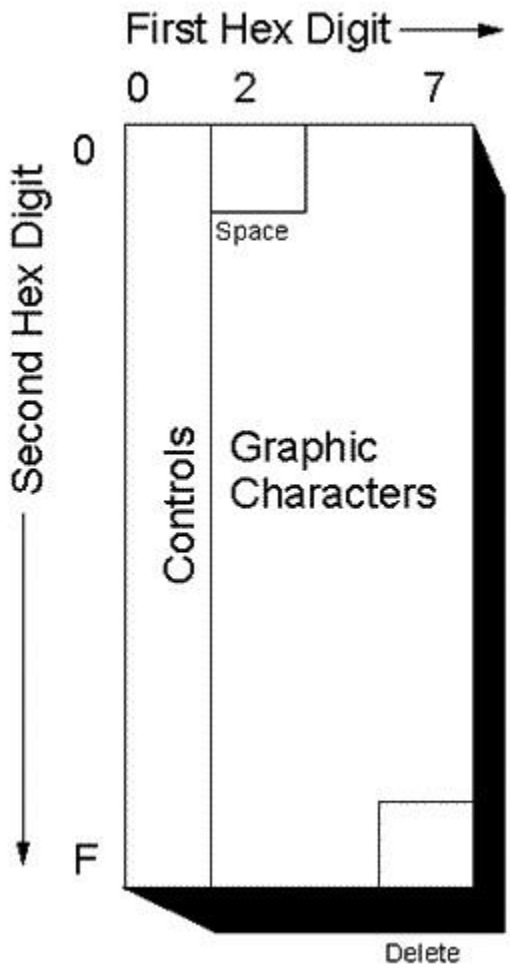


Figure 41. ISO 7-Bit Code Structure (ES = X'5yzz')

ISO 8-bit Structure

The ISO 8-bit structure is shown in Figure 42.

It has the following characteristics:

- Octet (8-bit byte) values can range from X'00' to X'FF'.
- X'00' to X'1F' (called the C0 control set) and X'80' to X'9F' (called the C1 control set) are reserved for assigning to control characters.
- X'21' to X'7E' (the G0 graphic set), and X'A0' to X'FF' (the G1 graphic set) are reserved for assigning to graphic characters.
- X'20' is reserved for the *SPACE* character if it is part of the last invoked character set.

- X'7F' is reserved for the *DELETE* character if it is part of the last invoked character set.
- X'A0' may be assigned to the SPACE character, and X'FF' may be assigned to the DELETE character.
- The registered ISO G sets that are based on ISO 4873 assign invariant code points allocated to the syntactic character set CS 640 when the character set is part of the character set in the G set (see Figure 43). There is one more character, the exclamation point, which is also assigned an invariant code point.
- ESID X'4105' is used to describe an ISO 8-bit encoding where graphic characters may be present in the C1 control space (X'80' to X'9F').
- ESID X'4155' is used to describe ISO 8-bit Presentation code where graphic characters are found in both the C0 and C1 control spaces.

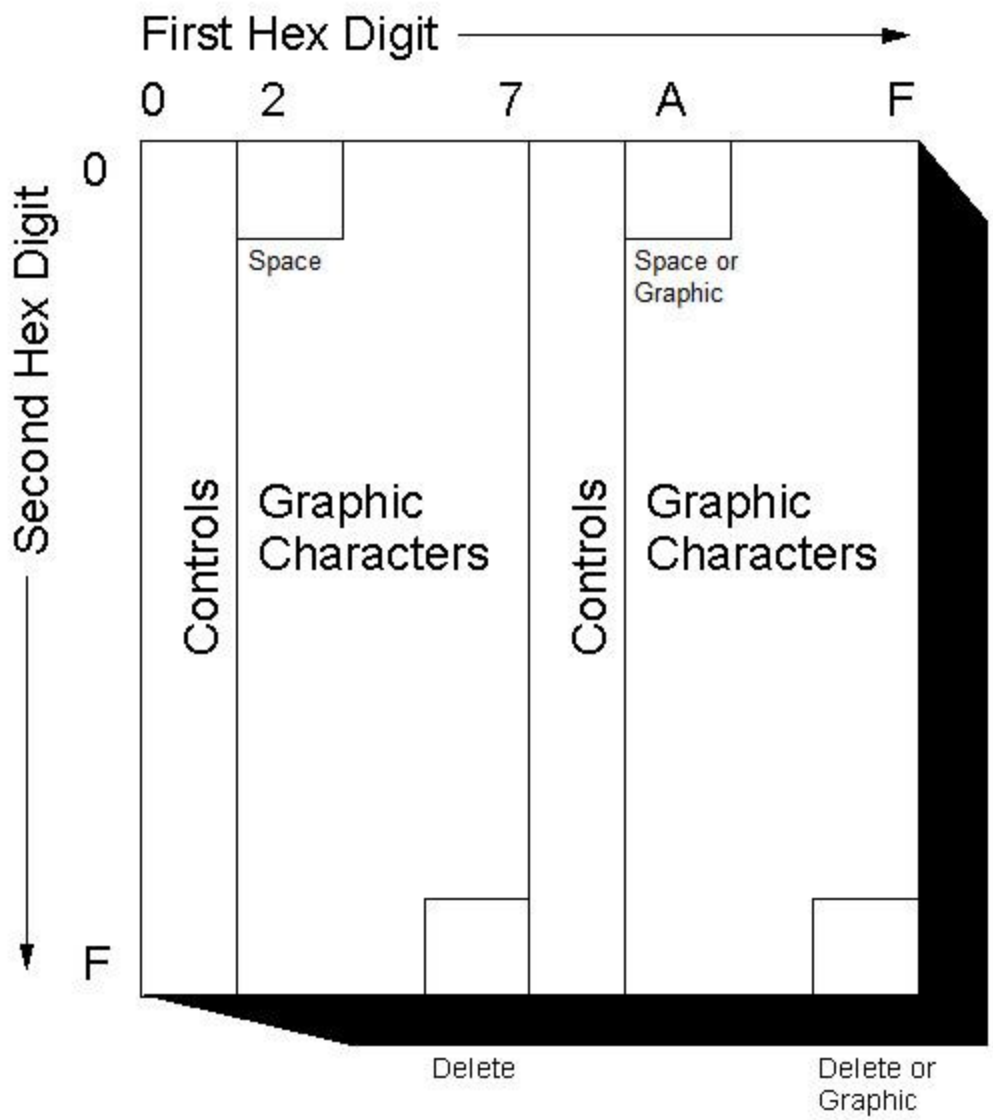


Figure 42. ISO 8-Bit Code Structure (ES = '4yzz')

The range of code positions X'20' to X'7F' are often referred to as the Left Half (GL) and X'A0' to X'FF' as the Right Half (GR) of an ISO-8 code.

Figure 43 shows the invariance of the syntactic character set found in the basic single byte (SBCS) encoding structures.

Character	GCGID	PC, ISO-7, ISO-8	EBCDIC
CAUTION: There are some coded character sets in use in which the invariant property is not guaranteed. Among the ISO-7 and derived codes, one more character, the exclamation mark (SP020000) is allocated the invariant code point X'21'. It is not included in this table, since it is not in the syntactic character set (CS 640).			
" (double quote)	SP040000	22	7F

% (percent)	SM020000	25	6C
& (ampersand)	SM030000	26	50
' (apostrophe)	SP050000	27	7D
('left parenthesis'	SP060000	28	4D
) 'right parenthesis'	SP070000	29	5D
* (asterisk)	SM040000	2A	5C
+ (plus)	SA010000	2B	4E
, (comma)	SP080000	2C	6B
- (hyphen)	SP100000	2D	60
. (period)	SP110000	2E	4B
/ (slash)	SP120000	2F	61
0	ND100000	30	F0
1	ND010000	31	F1
2	ND020000	32	F2
3	ND030000	33	F3
4	ND040000	34	F4
5	ND050000	35	F5
6	ND060000	36	F6
7	ND070000	37	F7
8	ND080000	38	F8
9	ND090000	39	F9
: (colon)	SP130000	3A	7A
; (semi-colon)	SP140000	3B	5E
< (less than)	SA030000	3C	4C
= (equal)	SA040000	3D	7E
> (greater than)	SA050000	3E	6E
? (question mark)	SP150000	3F	6F
A	LA020000	41	C1
B	LB020000	42	C2
C	LC020000	43	C3
D	LD020000	44	C4
E	LE020000	45	C5
F	LF020000	46	C6
G	LG020000	47	C7
H	LH020000	48	C8
I	LI020000	49	C9
J	LJ020000	4A	D1
K	LK020000	4B	D2
L	LL020000	4C	D3
M	LM020000	4D	D4
N	LN020000	4E	D5
O	LO020000	4F	D6

P	LP020000	50	D7
Q	LQ020000	51	D8
R	LR020000	52	D9
S	LS020000	53	E2
T	LT020000	54	E3
U	LU020000	55	E4
V	LV020000	56	E5
W	LW020000	57	E6
X	LX020000	58	E7
Y	LY020000	59	E8
Z	LZ020000	5A	E9
_ (underscore)	SP090000	5F	6D
a	LA010000	61	81
b	LB010000	62	82
c	LC010000	63	83
d	LD010000	64	84
e	LE010000	65	85
f	LF010000	66	86
g	LG010000	67	87
h	LH010000	68	88
i	LI010000	69	89
j	LJ010000	6A	91
k	LK010000	6B	92
l	LL010000	6C	93
m	LM010000	6D	94
n	LN010000	6E	95
o	LO010000	6F	96
p	LP010000	70	97
q	LQ010000	71	98
r	LR010000	72	99
s	LS010000	73	A2
t	LT010000	74	A3
u	LU010000	75	A4
v	LV010000	76	A5
w	LW010000	77	A6
x	LX010000	78	A7
y	LY010000	79	A8
z	LZ010000	7A	A9

Figure 43. Invariance of Syntactic Character Set in Basic SBCS Encoding Structures

EBCDIC Double and Mixed-byte Structures

The structure of IBM double-byte coded character sets is specified in IBM standards.

The double-byte EBCDIC code is called DBCS-HOST code. The basic EBCDIC structure has allocated coding space for control characters and graphic characters separately. The following describes the graphic character range of hexadecimal codes in the DBCS-HOST structure. Figure 44 illustrates the DBCS-HOST graphic character coding space. There are no 16-bit codes for control characters in the EBCDIC structure definition. A DBCS-HOST graphic character code has the following characteristics:

- The first byte is in the range X'41' to X'FE'
- The second byte is also in the range X'41' to X'FE', for all currently defined code pages
- X'4040' represents DBCS-HOST SPACE
- All other undefined 16-bit patterns are invalid as graphic characters.

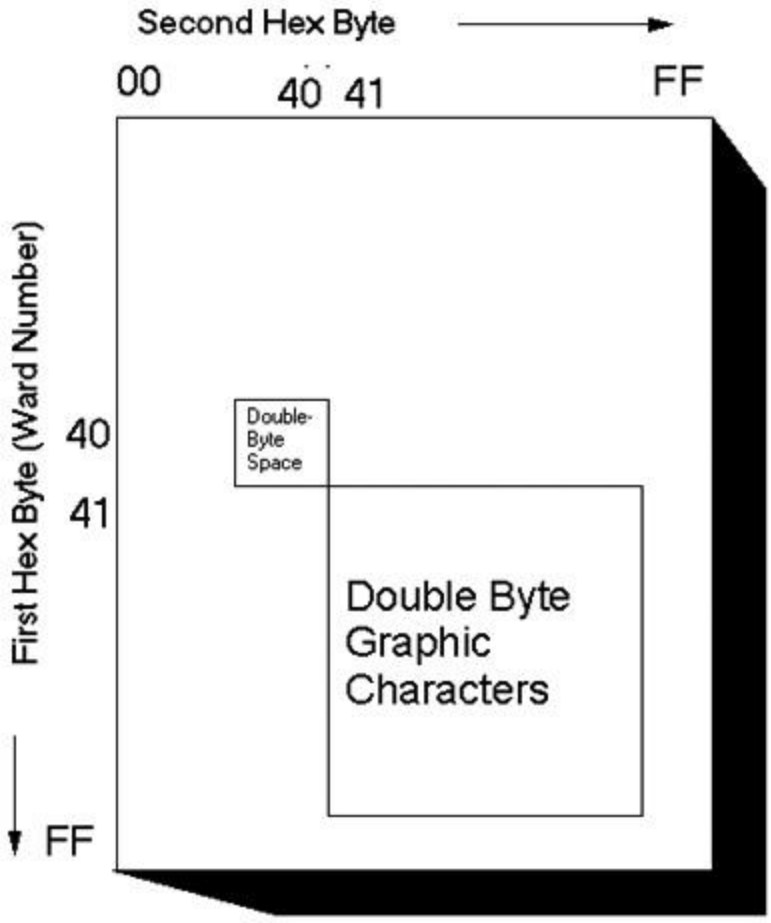


Figure 44. EBCDIC DBCS Graphic Character Coding Space (ES = X'12zz' or X'13zz')

A DBCS Ward

A section of a DBCS where the first bytes of all the code points belonging to it are the same is called a ward. A set of wards can be registered with a unique character set identifier, GCSGID, and the associated CPGID of the DBCS. This unique identifier, CGCSGID, defines the valid graphic character code points belonging to that set of wards.

EBCDIC Single/Double-Byte Mixed Encoding Structure

The coding space for EBCDIC Single/Double-byte mixed graphic characters is shown in Figure 45. The encoding scheme is a hybrid of the two EBCDIC schemes: EBCDIC SBCS, described earlier in Figure 37, and EBCDIC DBCS, described in and Figure 44. This encoding scheme is a stateful encoding and uses a code extension technique to

change between SBCS mode and DBCS mode. The control codes used to identify this change of state are X'0E' (shift out of SBCS) and X'0F' (shift into SBCS mode). The default starting state for a string encoded using this encoding scheme is single-byte. For a mixed string to begin in DBCS mode the first double-byte character must be preceded by a X'0E' in order to 'shift out' of SBCS mode. A well-formed mixed host string must have matching shift out, shift in (SO, SI) pairs. All well-formed mixed host strings will end in single-byte mode. When in either mode, the behavior of this encoding is as prescribed by the respective encoding scheme. All the semantics of the two individual encoding schemes apply in this case as well.

The following are examples of well-formed mixed EBCDIC strings. In these examples SO - represents a shift-out control, SI - represents a shift-in control, s - represents a single-byte character and dd - represents a double-byte character.

ssssSOddddddddSIssssss - in this example the string begins in single-byte mode, shifts to double-byte mode for 4 characters and then returns to single-byte mode.

SOddddddddSIssssss - in this example the string begins with double-byte characters, thus the first character of the string must be the shift-out, following the double-byte characters there is a shift-in to change to the single-byte state for the last 5 characters in the string.

ssssSOddddddddSI - in this example the string begins in single-byte mode, shifts to double-byte mode and even though the string ends in double-byte mode the Shift-in control is required to create a well-formed string.

SOSIssssSOddddddddSIssssss - in this example the SOSI at the beginning of the string is treated as a no-op. This is true for a SOSI pair found anywhere in a mixed EBCDIC string.

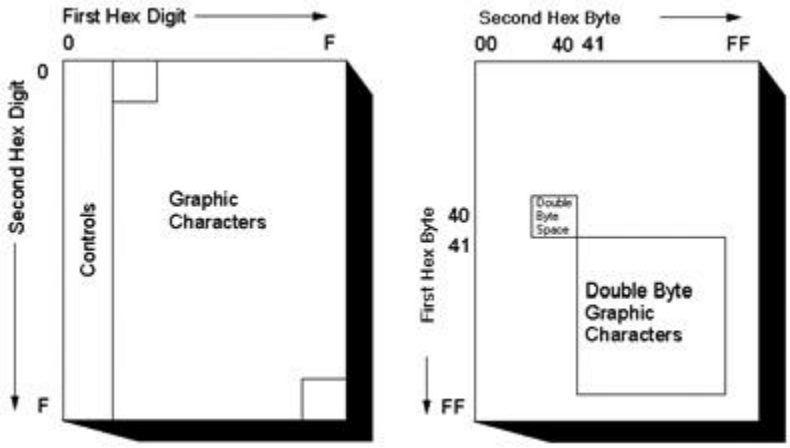


Figure 45. EBCDIC Mixed Single/Double-Byte Code Structure (ES = X'1301')

IBM PC Double and Mixed-byte Structures

The coding space for DBCS-PC graphic characters is shown in Figure 46.
The DBCS-PC graphic character code has the following characteristics:

- The first byte is in the range X'81' to X'FE'
- The second byte is in the ranges X'40' to X'7E' or X'80' to X'FE', for all currently defined code pages

  Note: Application developers are cautioned to not rely on the absolute code point range values as they may change in the future. The begin and end values may be CCSID dependent.

- DBCS-PC SPACE is variant and CCSID-dependent
- All other undefined 16-bit patterns are invalid as graphic code points.

Note: It is not advised to rely on the specific values above X'40' (second byte value) to denote the presence or absence of DBCS characters. These values will be encoding scheme specific and can change over time.

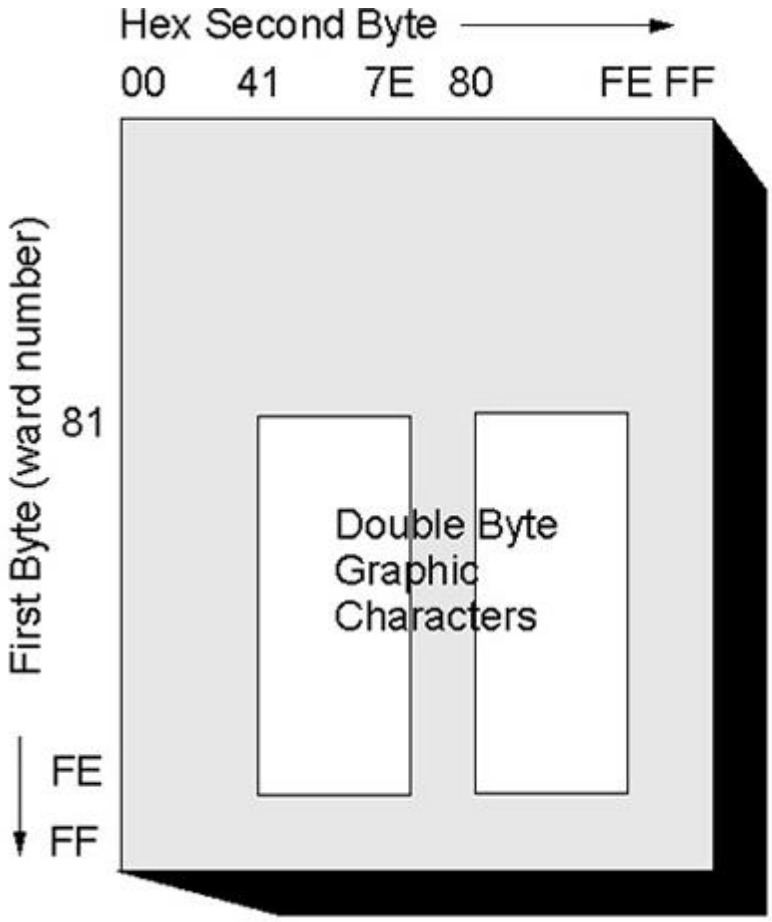


Figure 46. IBM-PC DBCS Graphic Character Coding Space (ES = X'22zz', X'32zz')

Note: In practice, the graphic characters of DBCS-PC are used with a single-byte PC coded character set. The specific values in the allocated range to be used as the first byte of a double-byte are detailed when the coded character set is registered. Other values from this range may be defined to be used as single-byte code points, and when so defined are not available for use as the first byte of a double-byte. Similarly, when a code point is declared to be the first byte of a double-byte code point, it cannot be used as a single-byte code point.

The control characters are all single-byte codes, as defined earlier for the IBM-PC Display and IBM-PC Data code structure. The definition of a ward given above also applies to DBCS-PC.

IBM-PC Mixed Single- and Double-Byte Structure

In the PC-Mixed scheme, both single-byte and double-byte code points may exist in the same data stream, without any explicit demarcation points between them.

Each specific use of a PC-Mixed scheme (ES=X'23zz' or X'33zz') must have an associated declaration of the specific single-byte code points to be used as the first byte of the double-byte code point. This set of code points is equivalent to a set of specific single-shift control code points in ISO (for example, Single-shift-2 (X'8E') as defined in ISO 6429). Each single-shift control causes the meaning of the following single-byte code point to be taken from a specific ward. The value of the first byte, besides being a single-shift control, is equal to the ward number. Figure 47 illustrates this definition.

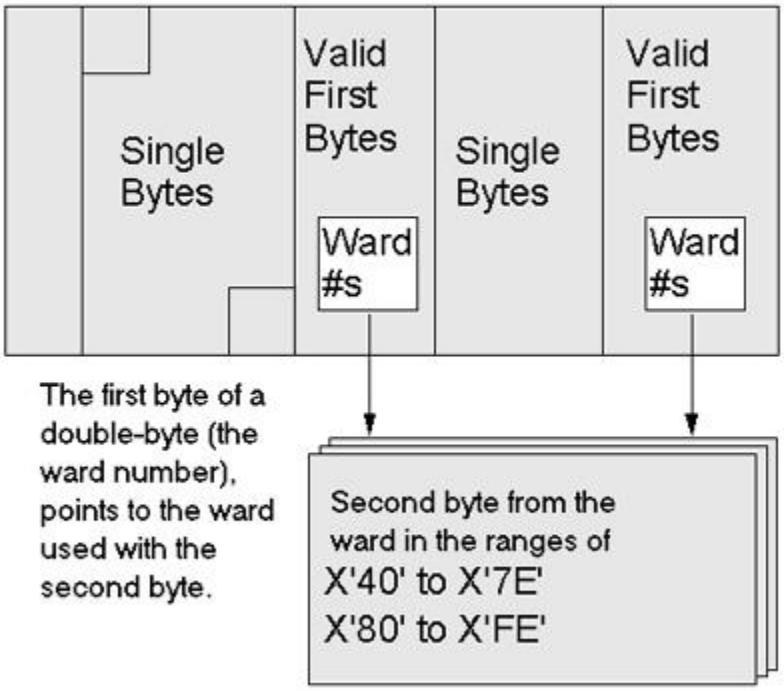


Figure 47. IBM-PC Mixed Single/Double-Byte Graphic Character Coding Space (ES = X'2300', X'3300')

Note: Application developers are cautioned to not rely on the absolute code point range values as they may change in the future. The begin and end values may be CCSID dependent.

The double-byte codes starting with a valid first byte follows the definition for IBM-PC Double-byte code structure. All the bytes that are not in the valid list of first bytes will have their single-byte code points assigned per IBM PC Single-Byte Data or Display structure definition. In comparison, in a pure PC-DBCS scheme the single-byte graphic code points of the base PC Encoding structure that are not used as the first byte of a double-byte code point cannot be assigned a graphic character.

Note: The size of the maximal character set of the double-byte code page determines the size of the double-byte coding space needed. This in turn governs the number of wards needed, and the corresponding number of code points to be reserved for use as the first byte of a double-byte code point. The character set of the single-byte code page also influences the maximum number of single-byte code points needed, by trading off with the maximum number of wards possible. The net result is that when a specific single-byte code page and a specific double-byte code page are used with the mixed encoding structure of the PC, the list of valid first bytes also gets fixed.

IBM Extended Unix Code (EUC)

IBM's adaptation of Extended Unix* Code (EUC) is called IBM EUC. It is also known (in IBM AIX documentation) as Multiple Byte Character Set (MBCS). The structure of IBM EUC coded character sets is specified in IBM Corporate Standard, Double-Byte Character Set (DBCS), Terminology and Coding Scheme, C-S 3-3220-102, 1992-07. The encoding scheme used in IBM EUC is shown in Figure 48.

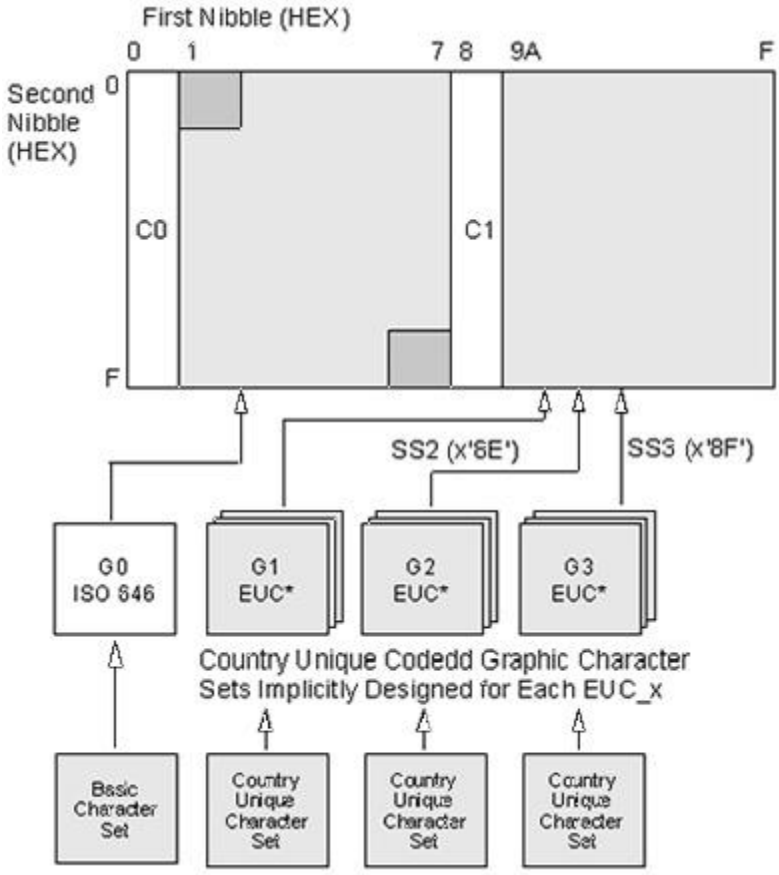


Figure 48. Designation and Invocation of IBM-EUC (ES = X'4403')

IBM EUC is an adaptation of one of the several code extension techniques defined in ISO 2022. It uses the 8-bit coding environment. The coded graphic character sets used are a national version of ISO 646 designated as the G0 set, and at most three additional G sets (G1, G2, and G3). The graphic character sets used correspond to the national standards of the different countries in the Far East.

The 8-bit environment of ISO 2022 implicitly designates the G0 set into the left half and the G1 set into the right half of the ISO-8 encoding structure (see section ISO 8-bit Structure). Encoding scheme X''8100' has been defined to describe a G1 set in the right-hand side of the ISO 8-bit encoding space when it is being used as a standalone portion of an EUC encoding. The single-shift controls, Single-shift 2 (SS2) and Single-shift 3 (SS3), are used for invoking the G2 and G3 sets into the right half of the 8-bit code. IBM EUC omits all the announcer, invocation, and designation sequences of ISO 2022.

The resultant complete coded graphic character sets are often called EUC_J (for use in Japan), EUC_K (for use in Korea), EUC_T (for use with Traditional Chinese), or EUC_S (for use with Simplified Chinese).

The EUC scheme combines up to four coded graphic character sets. The collection includes a basic character set (the G0 set of a national version of ISO 646), and one or more of the following coded graphic character sets:

- ISO 7/8 bit -- SBCS-EUC
- double-byte -- DBCS-EUC
- triple-byte -- TBCS-EUC.

The valid ranges of graphic character code points for each one of these sets when used in IBM EUC are given below:

- Basic Character Set is the G0 set of a national version of ISO 646, and is implicitly designated and invoked into the code point range X'21' to X'7E' for graphic characters.
- SBCS-EUC is a single-byte code page used with the code extension technique of IBM EUC. Each graphic character code point can be in the range X'A0' to X'FF' (called a 96-character G set in ISO 2022).
- DBCS-EUC is a double-byte coded graphic character set, which is used with the code extension technique of IBM EUC. The valid set of graphic character code points of DBCS-EUC is shown in Figure 49. Any graphic character code point of DBCS-EUC meets the following criteria:
    - Both bytes are in the range X'A0' to X'FF'
    - Any two-byte pattern that includes a byte value outside the above range is invalid.

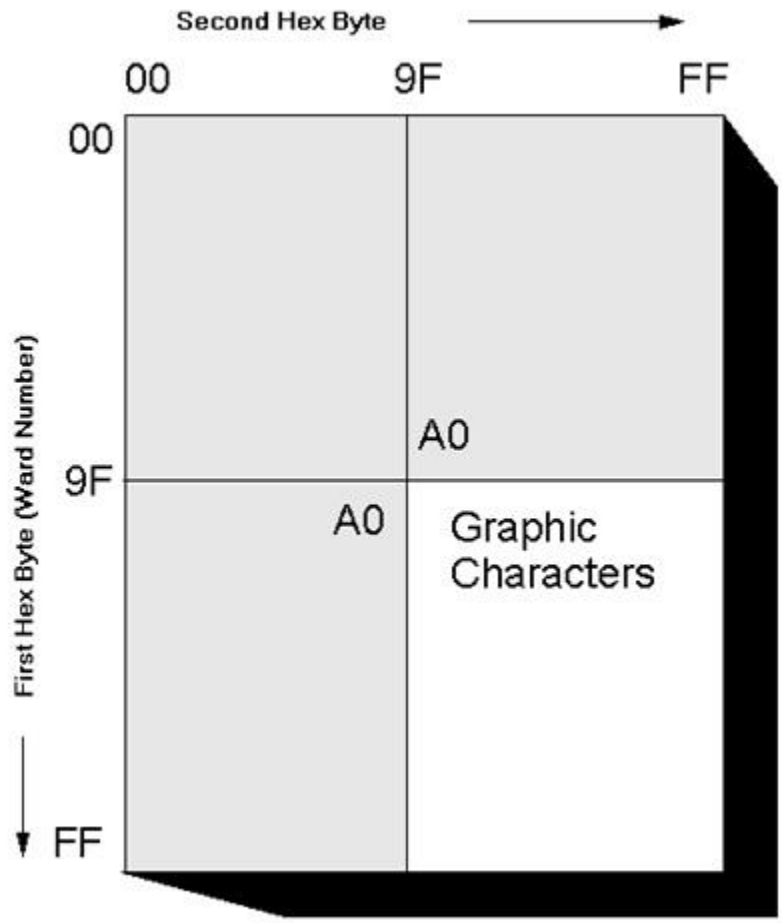


Figure 49. IBM-EUC Double-Byte Code Structure

- TBCS-EUC is a triple-byte coded graphic character set, which is used with the code extension technique of IBM EUC. The valid set of graphic character code points of TBCS-EUC is shown in Figure 50. Any graphic character code point of TBCS-EUC meets the following criteria:
  - All three bytes are in the range X'A0' to X'FF'
  - Any three-byte pattern that includes a byte value outside the above range is invalid.

  Any one of the SBCS-EUC, DBCS-EUC, or TBCS-EUC can be used as any of G1, G2, or G3 sets.

- Each code point invoked from G2 is preceded by an SS2 control that has been assigned X'8E' in C1.
- Each code point invoked from G3 is preceded by an SS3 control that has been assigned X'8F' in C1.

The remaining code points in the space X'00' to X'1F', and X'7F' to X'9F', follow the rules for an ISO 8-bit code (see ISO 8-bit Structure).

- The default SPACE (X'20'), DELETE (X'7F'), and control code point assignments for C0 and C1 sets as defined in ISO 6429.
- The SS2 and SS3 controls are from the C1 set (X'8E' and X'8F').
- The default Substitute (SUB) code point is from the C0 set (X'1A').
- Additional SPACE and SUB control code points may also be specified to be used with G1, G2, or G3 sets.

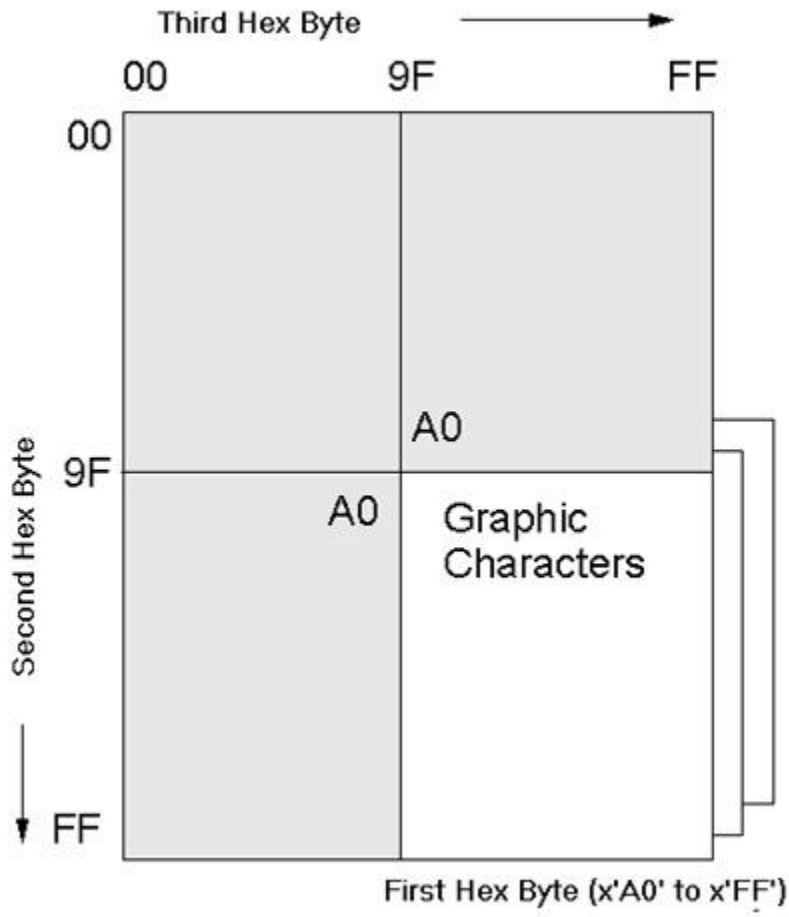


Figure 50. IBM-EUC Triple Byte Code Structure (ES = X'5700' standalone or as part of ES = X'4403')

Notes:
1. EUC does not specify what happens when a control set that is designated and

invoked as a C1 set has SS2 and SS3 controls assigned to code points other than X'8E' and X'8F' -- for example, the control sets of CCITT T.61 for Telematic Services.

2. IBM EUC specifies that the right half of the 8-bit coding space (GR) is the single-shift area. The following announcer sequences of ISO 2022 correspond to the EUC adaptation:

- ESC 20 43 announces an ISO-8 environment, with G0 on the left side and G1 on the right side of the 8-bit code
- ESC 20 5A announces an additional G2 invoked using SS2
- ESC 20 5B announces an additional G3 invoked using SS3.
- ESC 20 5C announces the single-shift to be GR.

Unicode

Unicode is a universal character encoding scheme that has been developed by a consortium made up of members of the worldwide IT community. The consortium is committed to maintaining synchronization between Unicode and ISO/IEC 10646, Information technology - Universal Coded Character Set (UCS). The encoding structure defined here for use in CDRA is applicable to both Unicode and ISO/IEC 10646.

Unicode provides a means of encoding all of the characters used for the written languages of the world. It has the capability to encode up to 216 x 17 characters. Unicode has been accepted by many as the strategic direction towards multilingual computing.

The basic encoding structure of Unicode is shown in Figure 51. Unicode is made up of 17 planes of 256 rows and 256 columns. Plane 0 is the Basic Multilingual Plane (BMP). It contains most of the currently encoded characters. Plane 0 includes an area reserved for Private Use Characters (PUA) and an area used for surrogate characters. Plane 1 is the Supplementary Multilingual Plane. Its purpose is to encode characters from archaic or obsolete writing systems. Plane 2 is the Supplementary Ideographic Plane and is used for encoding rare and unusual Han characters (Chinese, Japanese, Korean and Vietnamese unified Ideographs). Planes 3 through 13 are currently (and expected to remain) unassigned. Plane 14 is reserved for special purpose characters and is thus called the Supplementary Special-Purpose Plane. The final two planes, 15 and 16, are Private Use Planes to be used as an extension of the private use area found in the BMP. Encoding scheme X'7209' has been defined to represent an individual plane within the Unicode structure. This encoding scheme is used when

referencing a plane. Additional information about the Unicode encoding structure can be found on the Unicode web site.

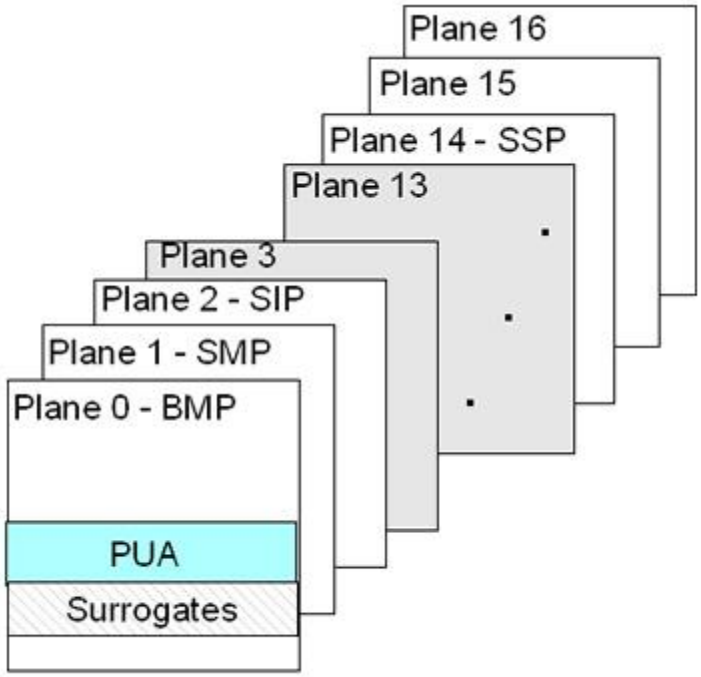


Figure 51. Unicode Basic Code Structure

Unicode Encoding Formats

Unicode is unique and different from most character encodings in that there are several formats defined for the encoding. The various encoding formats are identified and described in chapter 2 of the Unicode standard. While the encoding space is well structured and clearly defined, the Unicode Standard allows a number of different encoding formats. Characters may be encoded in one, two or four-byte formats. Each of these formats is briefly described below. For more detailed information refer to the Unicode Standard V4.0 documentation or the Unicode web site. The Unicode encoding structure can easily be defined using the standard CDRA Encoding Scheme Identifiers (ESIDs). Several ESIDs have been defined in order to accurately define the various encoding formats. To accurately interpret Unicode encoded data it is essential that the encoding be known and clearly defined.

UTF-8 (ES = X'7807')

Unicode Transformation Format 8 (UTF-8) is a way of transforming all Unicode characters into a variable length encoding of bytes. It has the advantages that the Unicode characters corresponding to the standard 8-bit ASCII set have the same values as ASCII, and that Unicode characters transformed into UTF-8 can often be used with existing software without extensive software rewrites. The main disadvantage of this encoding form is the overhead required to perform the

transformation from one of the other encoding formats into UTF-8. UTF-8 is commonly used for file storage and as the default by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) protocols. The CDRA defined encoding scheme identifier for UTF-8 is 7807.

UTF-16 (ES = X'7200', X'720B', X'720F', X'8200)

Unicode Transformation Format 16 (UTF-16) is a reasonably compact encoding and all the heavily used characters fit into a single, 16-bit code units in byte serialized form. All other characters are available via pairs of 16-bit codes (surrogates). UTF-16 is the most commonly used encoding form for internal processing. When using UTF-16 the order of the bytes of the character can be either most-significant-byte-first (big-endian, BE order) or least-significant-byte-first (little-endian, LE order). CDRA defines four ESIDs for UTF-16. The first is 7200. 7200 indicates UTF-16 with BE order. The second is 720B which indicates UTF-16 LE. The third ESID defined is 720F. 720F indicates UTF-16 where the endian order is determined by a byte order mark (BOM). If presents, a byte order mark will be found as the first two bytes of a data string. The value of the BOM indicating BE order is x'FEFF' and indicating LE order is x'FFFE'. If no BOM is found, the data is assumed to be big endian. The final encoding scheme defined for UTF-16 is 8200. This encoding scheme is called 'Unicode Presentation'. It is defined to be BE order in the absence of a BOM and is used exclusively by IBM printing systems. 8200 is a derivation of Unicode. It defines the C0 and C1 space of Unicode to be used for graphic characters.

UTF-32 (ES = X'7500', X'750B', X'750F')

Unicode Transformation Format 32 (UTF-32) provides fixed width, single code unit access to all of the characters. Each Unicode character is encoded in a single 32-bit code unit when using UTF-32. As is the case with UTF-16, UTF-32 can also be byte-serialized in either big-endian (BE) or little-endian (LE) order. CDRA defines three encoding schemes for UTF-32 format. 7500 is defined for UTF-32 BE. Encoding scheme 750B explicitly defines the data to be UTF-32 LE. The third ESID is 750F. 750F indicates UTF-32 where the endian order is determined by a byte order mark (BOM). The byte order mark for UTF-32 is X'0000FEFF' for indicating BE order and X'FFFE0000' for indicating LE order. If no BOM is found, the data is assumed to be in BE order.

UTF-EBCDIC (ES = X'1808')

Unicode Transformation Format EBCDIC (UTF-EBCDIC) provides an EBCDIC friendly way of encoding Unicode. UTF-EBCDIC is defined in Unicode Technical Report 16. UTF-EBCDIC defines a means of transforming Unicode characters to a form that is

safe for EBCDIC systems for the control characters and invariant characters. CDRA defines encoding scheme 1808 for UTF-EBCDIC. UTF-EBCDIC is intended to be used inside EBCDIC systems or in closed networks where there is a dependency on EBCDIC hard-coding assumptions. UTF-EBCDIC is unsuitable for use over the Internet or for data interchange.

Standard Compression Scheme for Unicode (SCSU) (ES = X'7B0C')

The Unicode Standard defines a compression scheme for storing and transmitting Unicode data. The details of this encoding form can be found in Unicode Technical Standard 6. The CDRA defined ESID for Unicode SCSU is 7B0C.

Binary Ordered Compression for Unicode (BOCU-1) (ES = X'7B0E)

The Unicode Standard defines this MIME compatible compression for Unicode. The details of this encoding form can be found in Unicode Technical Note #6. The CDRA defined ESID for Unicode BOCU-1 is 7B0E.

Compatibility Encoding Scheme for UTF-16: 8-Bit (ES = X'780D')

Unicode Technical Report 26 specifies an 8-bit Compatibility Encoding Scheme for UTF-16 (CESU) that is intended for internal use within systems processing Unicode in order to provide an ASCII-compatible 8-bit encoding that is similar to UTF-8 but preserves UTF-16 binary collation. It is not intended nor recommended as an encoding used for open information exchange. The CDRA defined ESID for Unicode CESU-8 is 780D.

Chinese Standard GB18030

GB18030 is a Chinese Standard which was defined as a super set of previously defined standards including GB 2312-80. It was defined to give customers the capability of using and processing a greater number of Chinese characters which are necessary for many applications used in organizations such as banks, insurance companies and by the postal service. It currently contains all characters defined in Unicode 3.0 including more than 27,000 Chinese characters. This standard provides solutions for the urgent needs of Chinese characters used in names and addresses.

GB 18030 uses a combination of one-byte, two-byte and four-byte codes and has a capacity of over 1.5 million code positions. The determination of character width (one, two or four-byte) is handled implicitly using code point ranges as shown in Figure 52.

Number of Bytes	Valid Byte Ranges				Number of Codes
One-byte	X'00'-X'80'				129 codes
Two-byte	First byte X'81' ~ X'FE'		Second byte X'40'~X'7E' X'80'~X'FE'		23,940 codes
Four-byte	First byte X'81'~X'FE'	Second byte X'30'~X'39'	Third byte X'81'~X'FE'	Fourth byte X'30'~X'39'	1,587,600 codes

Figure 52. GB18030 Structure (ES = X'2A00')

Lotus Multi-Byte Character Set (LMBCS)

LMBCS encoding is used exclusively by Lotus. It is defined as a multi-byte encoding made up of one, two and three-byte values. The first byte is the Group Byte. The Group Byte is a value between X'00' and X'1F' with meaning as described in Figure 53. Following the group byte will be either one or two bytes identifying the character. For optimization purposes, the group byte is omitted in Notes for single-byte values between X'20' and X'FF'. For example, LMBCS is always optimized to group 0x01, which means that any character where the first byte is greater than 0x1F, has an implicit group byte of 0x01.

Group byte	Character size (bytes)	Description
0x00		Reserved for future use
0x01	2	Byte 2 = Codepage 850, i.e. Multilingual DOS
0x02	2	Byte 2 = CP 851 (Greek DOS)
0x03	2	Byte 2 = CP 1255 (Hebrew Windows)
0x04	2	Byte 2 = CP 1256 (Arabic Windows)
0x05	2	Byte 2 = CP 1251 (Cyrillic Windows)
0x06	2	Byte 2 = CP 852 (Latin-2 DOS)
0x07	1	BEL
0x08	2	Byte 2 = CP 1254 (Turkish Windows)
0x09	1	TAB
0x0A	1	NL
0x0B		Reserved for future use
0x0C		Reserved for future use
0x0D	1	CR
0x0E		Reserved for future use
0x0F		Reserved for future use
0x10	3	Bytes 2 & 3 = CP 932

Group byte	Character size (bytes)	Description
0x11	3	Bytes 2 & 3 = CP 949
0x12	3	Bytes 2 & 3 = CP 950
0x13	3	Bytes 2 & 3 = CP 936
0x14	3	Bytes 2 & 3 = UTF-16 bytes
0x15 - 0x1F		Reserved for future use

Figure 53. LMBCS Structure (ES = X'9300')

Appendix B. Conversion Methods

Conversion tables alone do not ensure the transfer or sharing of data objects between different computing environments: the proper selection and use of these tables is essential. Conversion methods, as described in the following sections, are used with the tables referenced in Appendix J. As with the selection of a table, the conversion method that is best for one application may not be appropriate for another.

It is the responsibility of the person designing the conversion method to respect the characteristics and requirements of the input and output data. An appropriate method can be selected based on the encoding schemes (ESs) and string types (STs) of the input and output data. The conversion method models described in the following sections are specifically for coded graphic character strings whose semantics follow the respective ES definitions for the character encodings. Necessary enhancements needed to deal with the following string types are also briefly described:

- Input null-terminated.
- Output null-terminated.
- Output SPACE-padded.

Conversion methods are not supplied by CDRA, but are described here in the context of use with the conversion tables created and supplied by CDRA.

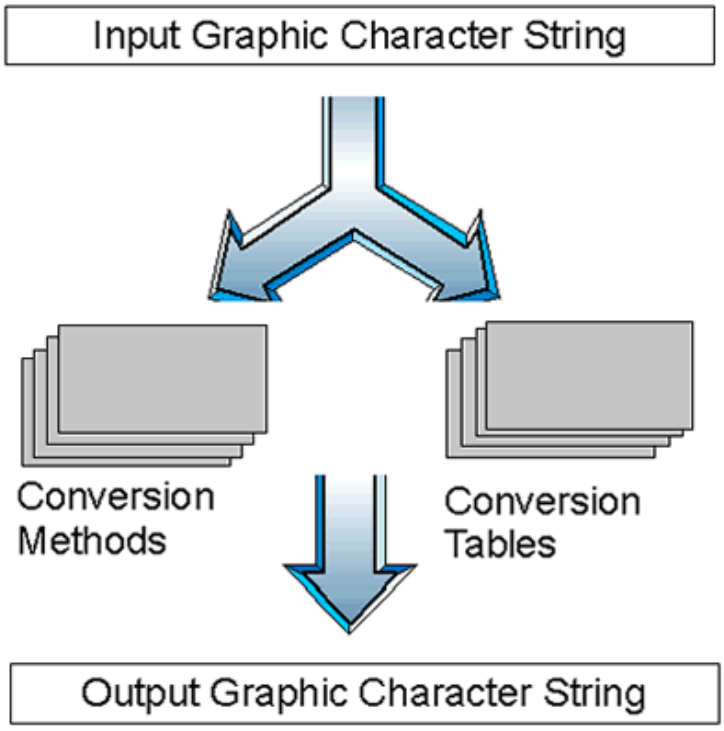


Figure 54. Use of conversion methods

Figure 54 shows the use of the conversion methods and tables within the overall conversion process. The conversion method first parses the input data string, and if necessary performs any required substring operation. A substring operation may be required if the input data string contains embedded code-extension controls, such as SO/SI controls in EBCDIC-mixed SBCS/DBCS data. The rules for parsing the specified string type should also be followed. The resulting substrings should contain code points that possess similar characteristics -- they are all from the same CS, CP pair identified. Each substring is converted from input code points to output code points using the appropriate conversion table. This table selection is based on the characteristics of the input data and the desired characteristics of the output data including the CS, CP pairs and ESs. Finally, the conversion method assembles the resulting output substrings into the final output string. This process should include the insertion of any code extension control characters that are required by the output ES. Rules for assembling the specified output string type (ST) should also be followed.

Each of the CDRA conversion methods is explained in detail in the following pages.

- Method 1 for SBCS
- Method 2 for Pure DBCS
- Method 3 for EBCDIC Mixed to PC Mixed
- Method 4 for PC Mixed to EBCDIC Mixed
- Method 5 for Single-byte to Double-byte
- Method 6 for Double-byte to Single-byte
- Method 7 for Mixed Single/Double-byte to Double-byte
- Method 8 for Double-byte to Mixed Single/Double-byte
- EUC and 2022 TCP/IP Conversion Tables
- Method 9 for PC to EUC Conversions
- Method 10 for EUC to PC Conversions
- Method 11 for Host to EUC Conversions
- Method 12 for EUC to Host Conversions
- Method 13 for PC to TCP Conversions
- Method 14 for TCP to PC Conversions
- Method 15 for Host to TCP Conversions
- Method 16 for TCP to Host Conversions
- Conversions Methods in Support of GB18030
- Use of Shadow Flags - An Example
- Enhancements to Support String Types

Method 1 for SBCS

This method has the following characteristics:

- It is used for conversions between two pure single-byte CCSIDs.
- The valid encoding schemes for the input and output data are X'1100', X'2100', X'3100', X'4100', X'4105', X'4155', X'6100' and X'8100'; this method can also be used with ES X'5100' and X'5150' (single-byte 7-bit code) with considerations for the 7-bit limit.
- The conversion table selected by this method will be a single-byte code point to single-byte code point table from the input CS, CP pair to the output CS, CP pair (known as a TYPE 1 table). Figure 55 shows a model for a TYPE 1 conversion table.
- The contents of the table will reflect the criterion used for mismatch management.
- All control characters are treated as pure single-byte controls, and are mapped according to the mismatch management criterion.
- Handling of control function sequences is beyond the scope of this method.

The machine-readable format of the single-byte to single-byte conversion table is a file containing a single 256-byte record. This allows for 256 single-byte output values. Each character in the table corresponds to one input code point, X'00' through X'FF'. The byte value of the character that is found in the location corresponding to the input code point value is the output code point.

In the example shown in Figure 55, to find the output code point for the input code point X'53', we look at offset X’53’ in the record. (The first position is offset X’00’ and the final position is offset X’FF’.) The value that we find at offset X’53’ is the corresponding output code point. In this example it is X'67'.

These tables are the standard distribution and storage format of CDRA.

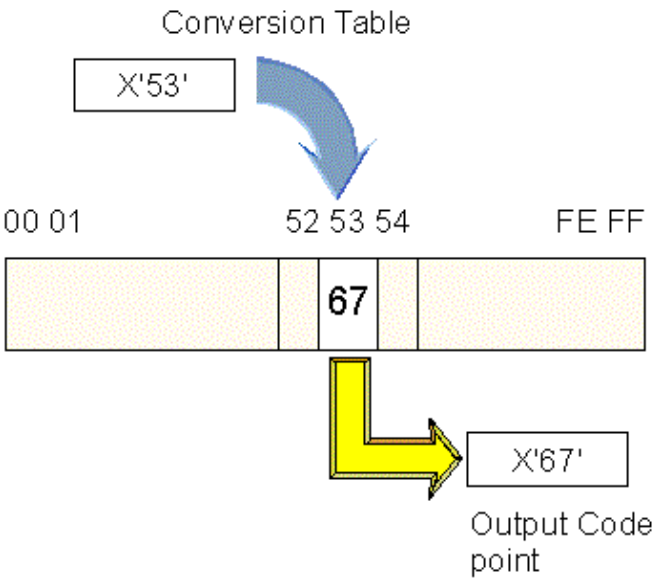


Figure 55. Single-byte to Single-byte Conversion

Method 2 for pure DBCS

This method has the following characteristics:

- It is used for conversions between two pure double-byte CCSIDs.
- The valid encoding schemes for the input and output data are X'1200', X'2200', X'3200', X'5200', X'6200', X'7200', X'8200' and X'9200'.
- The conversion table selected by this method will be a double-byte code point to double-byte code point table from the input CS, CP pair to the output CS, CP pair (known as a TYPE 2 table). Figure 56 shows a model for TYPE 2 conversion table.
- Most double-byte codes do not use all the available first bytes as valid ward numbers. Thus, the tables are organized as several sub tables, each containing 256 double-byte code point entries.
- The contents of the table will reflect the criterion used for mismatch management.
- Any fragmented double-bytes or first-bytes that do not have an entry in the conversion table are treated as errors in the data.

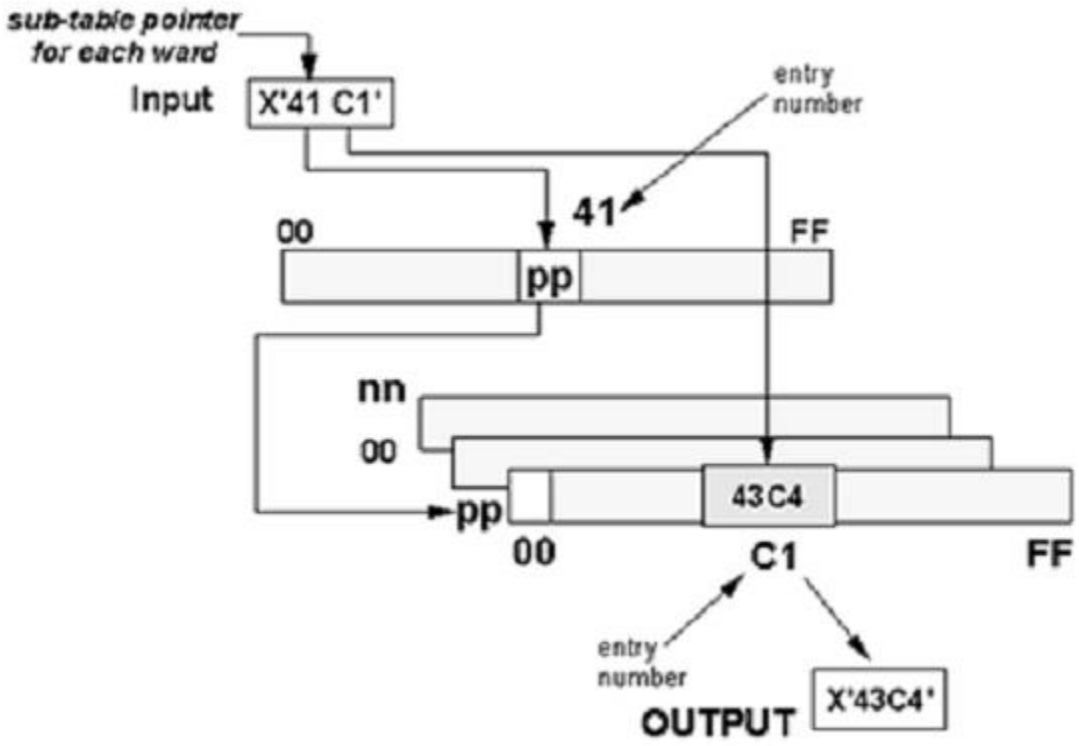


Figure 56. Method2: Double-byte to double-byte conversion

To understand the machine-readable format of the double-byte tables, you must first understand the concept of a "ward". A ward is a section of a double-byte coded character set, where the first byte of all code points contained in that section have the same value. A ward is populated if there are any characters in the double-byte coded character set whose first byte is the ward value. Conversely, a ward is not populated if there are no characters in the double-byte coded character set that have that ward value as the first byte. Each of the Far East double-byte coded character sets contains many unpopulated wards.

The CDRA machine-readable format of the double-byte conversion tables is a structure composed of many 512-byte records: a sub table pointer record, a substitution character record, and one record for each populated ward. The sub table record is the first record in the structure, and it is used as the index into the other character records of the structure.

The first 256 bytes of the sub table record contains information, whereas the second 256 bytes contains zeros. Each of the 256 assigned bytes corresponds to the first byte (the ward number) of an input code point. The byte values found in these locations are pointers to subsequent records in the structure that contain the output code points. Each of the subsequent records in the structure contain information in all 512 bytes in the form of 256 double-byte code point values. The appropriate subsequent record is selected from the structure using the value of the first byte of the input code point as an index into the sub table pointer record. The value obtained from the sub table pointer record points to the subsequent record required. The second byte of the input code point is then multiplied by two to calculate the correct offset into the selected record. Each output code point is two bytes in length, beginning at offset n into the record, where n is 2 times the input code point value.

In Figure 56 the conversion process may be described as follows:

1. Input code value X'41C1'
2. Use X'41' as an index into the sub table pointer record
3. Retrieve record pointer "pp"
4. Use the second input byte, X'C1', as an index into record "pp" to retrieve the output value X'43C4'.

Thus, the input code point X'41C1' maps to the output code point of X'43C4'.

The third record type in the structure is the substitution record. It is a 512-byte record constructed from 256 two-byte values, all of which are the code point value for the defined Substitute character for the target coded character set. The value retrieved

from the sub table pointer record for each unpopulated ward will point to this substitution record.

Method 3 for EBCDIC mixed to PC mixed

This method has the following characteristics:

- It is used for conversion between an input EBCDIC mixed CCSID (with SO-SI code extension controls) and a PC mixed CCSID.
- The valid encoding scheme for the input data is X'1301', and for the output data the encoding schemes are X'2300', X'2305' and X'3300'.
- The input parser separates the double-byte strings contained within the SO-SI pairs from the single-byte substrings. The SO-SI pair is discarded from the input string. The single-byte and the double-byte substrings are converted separately (shown in Figure 57).
- The input single-byte substrings are converted to corresponding output single-byte substrings using the appropriate Type 1 table (shown in Figure 55).
- The input double-byte substrings are converted to corresponding output double-byte substrings using the appropriate Type 2 table (shown in Figure 56).
- The output generator concatenates the converted substrings in the same order as their corresponding input substrings.
- The contents of the conversion tables used govern the accuracy of the output data.
- Handling of the single-byte controls within the input double-byte substrings is beyond the scope of this method.
- The removal of the SO-SI code extension controls generally results in an output string that is shorter in length than the corresponding input string.

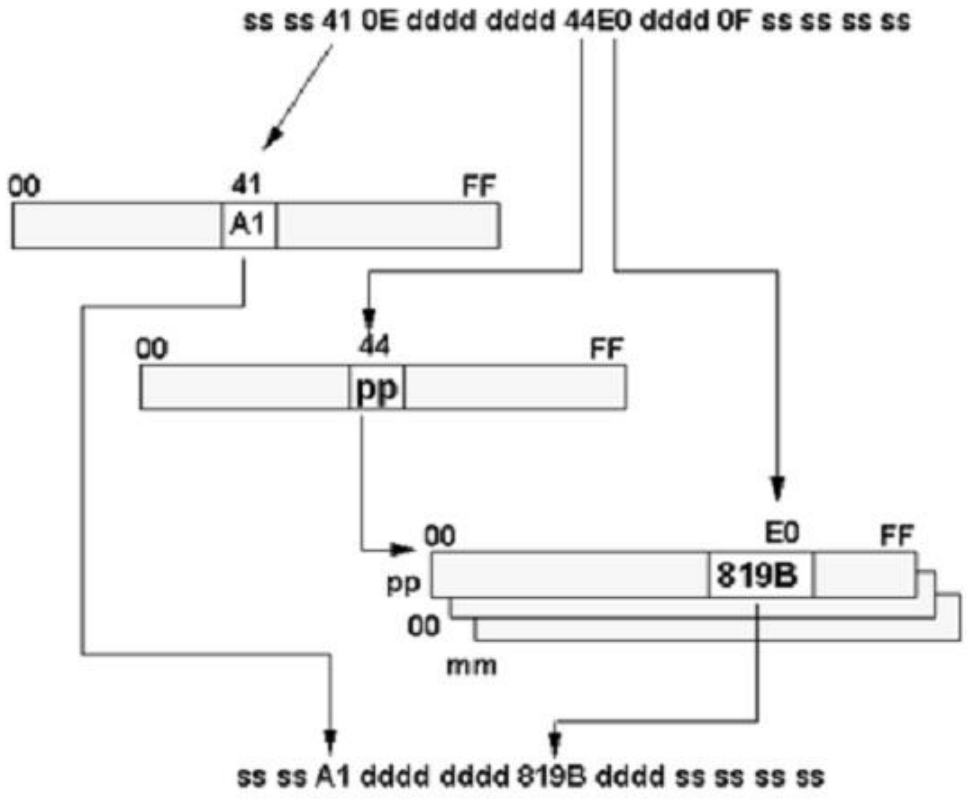


Figure 57. Method 3: Host Mixed Single/Double-byte to PC Mixed Single/Double-byte

Method 4 for PC mixed to EBCDIC mixed

This method has the following characteristics:

- It is used for conversion between an input PC mixed CCSID and an EBCDIC mixed CCSID (with SO-SI code extension controls).
- The valid encoding schemes for the input data are X'2300', X'2305' and X'3300', and for the output data is X'1301'.
- It uses a TYPE 3 conversion table (see Figure 58) consisting of:
  - A control sub table
  - A pool of single-byte to single-byte sub tables (Type S)
  - A pool of single-byte to double-byte sub tables (Type D)
  - There is an entry in the control sub table for each possible byte in the input string
  - The two-byte entry in the control sub table shows how many bytes are to be taken from the input string (single- or double-byte), the pool from which the conversion table is to be selected, and which conversion table to select.

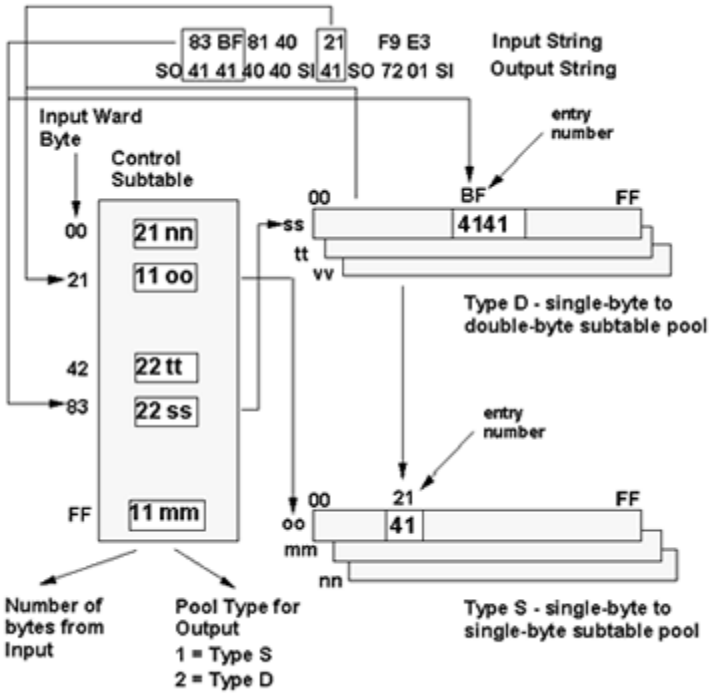


Figure 58. Method 4: PC Mixed Single/Double-byte to Host Mixed Single/Double-byte

- The list of valid first bytes associated with the input CCSID is used for separating the double-byte code points from the single-byte code points. This

function is built into the control sub table of a TYPE 3 table when the table contents are defined.

- This method can handle situations where the characters in an input CS are split among multiple output CSs.
- The pure double-byte substrings are mapped using the tables indicated in the control sub table. The first byte of each character is used as the index into the control sub table. The corresponding control table entry identifies the appropriate single-byte to double-byte table. The second byte is then used as the index into the single- to double-byte table. The contents of this cell will be a double-byte code point.
- The output double-byte substrings are bracketed within the SO-SI pair.
- The pure single-byte substrings are mapped using the appropriate single-byte to single-byte sub table, as identified in the control sub table of a TYPE 3 table.
- The output generator concatenates the converted substrings in the same order as their corresponding input substrings (the SO-SI pairs are part of the output double-byte substrings).
- The contents of the conversion tables used governs the accuracy of the output data.
- The addition of the SO-SI code extension controls generally results in an output string that is longer in length than the corresponding input string.
- The conversion method will generate single-byte control code points only outside the SO-SI pairs.
- All double-byte control code points will appear inside SO-SI pairs.
- The resultant output mixed string must be well formed. That is, all SO control codes must be paired with a matching SI control code even if the SI is the last character in the resultant string.

Method 5 for single-byte to double-byte

This method has the following characteristics:

- It is used for conversion between an input single-byte and an output double-byte CCSID.
- The valid encoding schemes for the input data are X'1100', X'2100', X'3100', X'4100', x'4105', X'4155'and X'6100', and for the output data are X'1200', X'2200', X'3200', X'5200', X'6200', X'7200', X'8200' and X'9200' (21).
- It uses a TYPE 4 conversion table (see Figure 59) consisting of one 512-byte record (this allows for 256 double-byte output values).
- The possible input, single-byte code point values are in the range X'00' through X'FF'.
- The input code point is used as an offset into the conversion table. The 2-byte entry beginning at this location is the actual output double-byte code point.
- The following steps are taken in order to convert a single-byte X'40' to a UCS-2 (double-byte) value, as shown in Figure 59:
  - The input value of X'40', is used as an index into the conversion table.
  - The X'40'th entry is found (remembering that each entry in the conversion table is two bytes long).
  - The two bytes found at this location comprise the output double-byte code point value.
- The resultant output string will be twice the length of the input string (each single-byte is converted to a double-byte).
- The content of the conversion table used governs the accuracy of the output data.

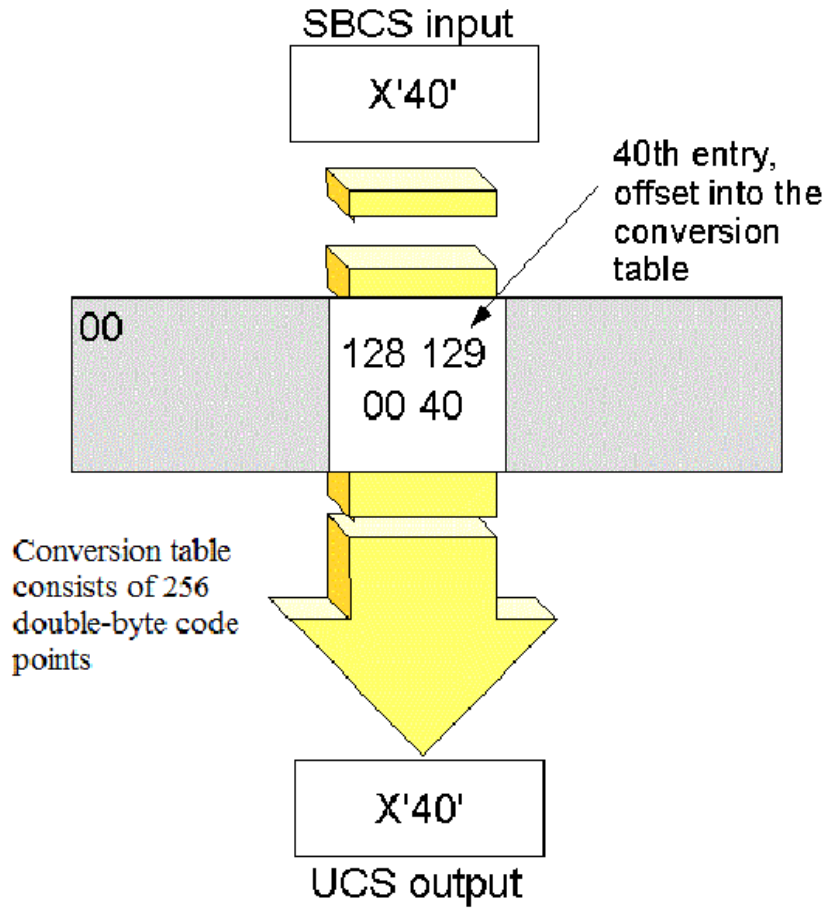


Figure 59. Method 5: SBCS to UCS Conversion Table

Method 6 for double-byte to single-byte

This method has the following characteristics:

- It is used for conversion between an input double-byte and an output single-byte CCSID.
- The valid encoding schemes for the input data are X'1200', X'2200', X'3200', X'5200', X'6200', X'7200', X'8200' and X'9200' and for the output data are X'1100', X'2100', X'3100', X'4100', X'4105', X'4155' and X'6100'.
- It uses a TYPE 5 conversion table (see Figure 59) consisting of:
- A 256-byte sub table pointer record.
- A pool of 256-byte sub tables.
- The method takes each input double-byte code point and separates it into a first and second byte.
- The first byte is used as an offset into the sub table pointer record.
- The value found at this location "points" to the appropriate record in the pool of sub tables.
- The second byte is then used as an offset into the selected sub table record.
- The value found at this location is the single-byte output code point.
- In the example shown in Figure 60 the following takes place:
- The first byte of the input value X'00' is taken and used as the offset.
- At location X'00' in the sub table pointer record the value 03 is found.
- The method locates record 03 in the sub table pool and uses the second byte of the input value, X'41', as the offset.
- The value found at this location, X'C1', is the output single-byte value.
- The resultant output string will be half the length of the input string (each double-byte is converted to a single-byte).
- The content of the conversion table used governs the accuracy of the output data.

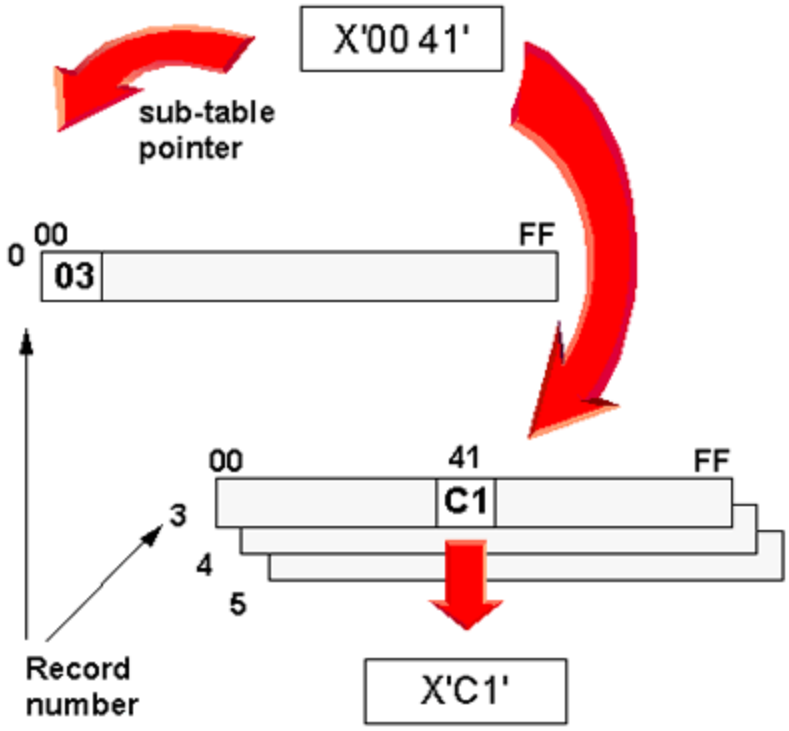


Figure 60. Method 6: DBCS to SBCS conversion table

Method 7 for Mixed Single/Double-byte to Double-byte

This method has the following characteristics:

- It is used for conversion between an input mixed single/double-byte and an output double-byte CCSID.
- The valid encoding schemes for the input data are X'1301', X'2300', X'2305' and X'3300', and for the output data are X'1200', X'2200', X'3200', X'5200', X'6200', X'7200', X'8200' and X'9200'.
- It uses a TYPE 2 conversion table, as described earlier for double-byte to double-byte conversions. See “Method 2 for Pure DBCS” for a description of the table and how it works.
- This method requires that the input data is normalized such that each input code point is two bytes long. This is done by prefixing each single-byte code point with a X'00'.
- Any code extension controls are also removed from the input data stream.
- The conversion then proceeds as any normal double-byte to double-byte conversion.
- This method is primarily used for converting data to UCS-2 (encoding scheme X'7200').
- The resultant output string will not necessarily be the same length as the input string, (each single-byte code point from the mixed input string is converted to a double-byte).
- The content of the conversion table used governs the accuracy of the output data.

Method 8 for mixed single/double-byte to double-byte

This method has the following characteristics:

- It is used for conversion between an input double-byte and an output mixed single/double-byte CCSID.
- The valid encoding schemes for the input data are X'1200', X'2200', X'3200', X'5200', X'6200', X'7200', X'8200' and X'9200', and for the output data are X'1301', X'2300', X'2305' and X'3300'.
- It uses a TYPE 2 conversion table, as described earlier for double-byte to double-byte conversions. See “Method 2 for Pure DBCS” for a description of the table and how it works.
- This method takes the two-byte input code points and uses the TYPE 2 conversion table to produce normalized (two-byte) output code points.
- The output data is then denormalized by removing the leading X'00' found on the normalized single-byte code points.
- Any necessary code extension controls are also added to the output data stream. The resultant string must be well formed as defined by the appropriate encoding structure. For more information see Appendix A.
- This method is primarily used for converting data from UCS-2 (encoding scheme X'7200').
- The resultant output string will not necessarily be the same length as the input string, (some of the input double-byte code points may map to single-byte code points in the output mixed CCSID).
- The content of the conversion table used governs the accuracy of the output data.

EUC and 2022 TCP/IP conversion tables

The Extended Unix Code (EUC) conversion tables are used to convert EUC encoded graphic character data from an EUC platform to or from a host or PC platform. The ISO 2022 TCP/IP (TCP) conversion tables are used to convert encoded graphic characters from the specific ISO 2022 format used by TCP/IP to or from a host or PC platform.

It is assumed at this point that the reader has some knowledge of the code extension techniques defined in the ISO 2022 standard. Both EUC and 2022 TCP/IP data streams make use of these techniques.

The conversion tables are constructed to achieve optimum character integrity after data conversion by using GCGID matching between source and target encodings. Both schemes can define up to four, character set and code page pairs to better enhance the character matching between source and target CCSIDs.

The EUC and TCP conversion tables use a normalized form of data. Input passed to and output generated from the conversion tables is normalized. The PC code points are normalized by placing a leading X'00' in front of each single-byte to yield a two-byte form. Host (EBCDIC) data must have the SO-SI control characters deleted during normalization and reinserted afterwards during denormalization. As with the PC data, a leading X'00' is inserted in front of any single-byte data. The EUC and TCP code points are normalized to four-byte values.

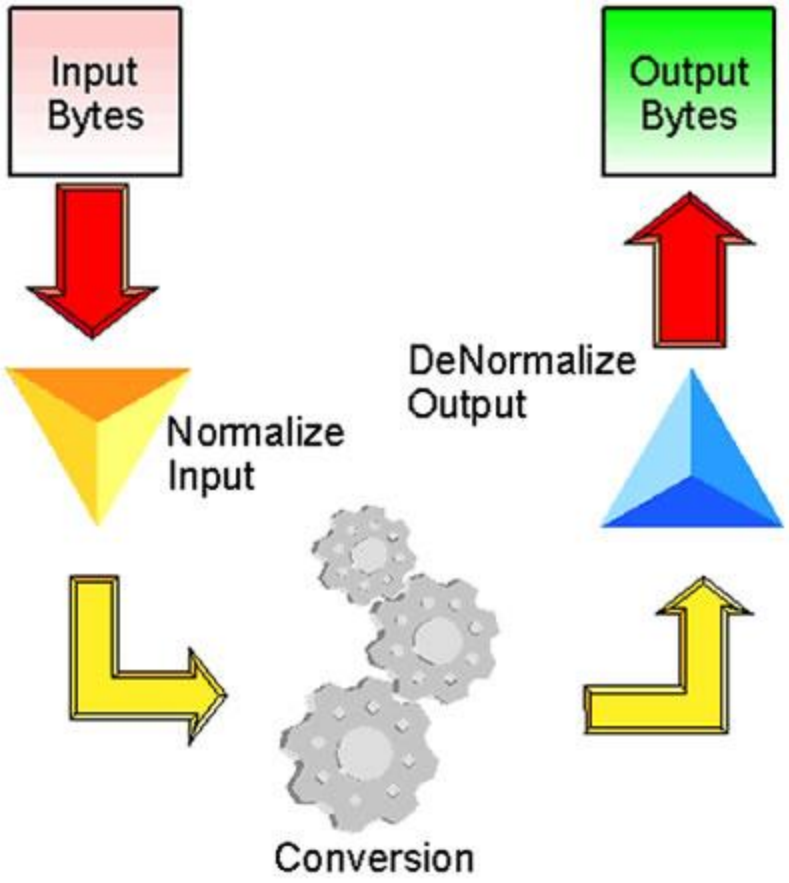


Figure 61. EUC and ISO 2022 TCP Conversion

Figure 61 shows the general use of the EUC and TCP conversion tables. The input byte or bytes, up to a maximum of four bytes per code point, are first normalized and used as input to the conversion table. The output (again, four bytes per code point maximum) from the conversion table is also normalized data, which must be denormalized prior to subsequent processing.

Normalization and denormalization services are not part of the CDRA-supplied conversion tables.

EUC conversions

The EUC encoding technique uses up to four coded graphic character sets. Each must be predefined, as the information is not carried in the text data stream. In CDRA, the CCSID determines the group of coded graphic character sets being used. Code points from the left half of the 8-bit encoding space (the high-order bit is OFF) are in the set G0. Code points that lie in the right half of the encoding space (the high-order bit is

ON) are in the set G1. The single-shift control characters, called SS2 and SS3, are used to invoke the other sets G2 and G3.

EUC conversions require that the EUC input or output contain not only the character to be converted, but the shift control character when applicable. All input EUC data needs to have the code point values padded with leading zeros to create a fixed length, normalized, four-byte encoding. The following example shows how a code point in G3 must be formatted for the conversion tables.

- Input code point X'A2C3' in set G3.
- SS3 character X'8F' must be present and included with the input value.
- Normalized input value becomes X'008FA2C3' (padded to a length of four bytes).

This means that when dealing with EUC data, the parser must recognize which G-set each character belongs to in order to build the correct normalized input for the conversion process. When converting from EUC, the de-normalizing process must strip off the leading zeros and concatenate the converted characters to build the correct output string.

2022 TCP/IP conversions

Conversion tables for TCP/IP are very similar to those for EUC, except that only one G-set is used, namely G0. To switch from one coded graphic character set to another cannot be accomplished using the EUC technique of a single shift. An explicit escape sequence is used to designate a new coded graphic character set being loaded into the G0 set. The escape sequence value itself cannot be carried in the conversion table entries, so the high-order byte of the TCP/IP code point value will be set to correspond to the position of the CGCSID within the CCSID. For example, CCSID 956 is defined for Japanese TCP/IP and contains four CGCSIDs corresponding to JIS X 201 Roman, JIS X 208-1983, JIS X 201 Katakana, and JIS 212. For Japanese host to TCP/IP conversion, a code point value from the JIS X 201 Katakana set would contain a value of X'03' in the high-order byte.

In ISO 2022, control characters are not part of the coded graphic character set; therefore, loading a new coded character set into G0 does not affect the set of control characters in C0. It thus does not make sense to have the high-order byte setting indicate a specific coded graphic character set. Control characters will then be normalized as follows;

1. For Host or PC to 2022 TCP/IP, the converted value will have X'00' in the high-order byte. This will indicate to the denormalization routine that it does not matter what the currently designated coded graphic character set is -- the control character may simply be placed directly into the output data stream.
2. For 2022 TCP/IP to Host or PC, the input normalized value may have X'00' as the high-order byte, or it may contain any valid value for the table. The normalization routine can then recognize the value as a control character and place a X'00' in the high-order byte or it may use the current active value as a result of the previous escape sequence.

Although the identifiers coded in the high-order bytes will correspond to the position of the coded graphic character set within the CCSID, each table will also contain the set of ESC sequences used to designate the coded graphic character set in the G0. This mapping will be contained in the first record of the conversion table in the following format:

L1 ESC1 L2 ESC2 ... Ln ESCn

where Li is a one-byte unsigned field containing the length of the following ESC sequence, and ESCi is the ESC sequence associated with the high-order byte id "i" in the table.

Method 9 for PC to EUC conversions

The method shown in Figure 62 has the following characteristics:

- It is used for conversion between an input PC CCSID and an output EUC CCSID.
- The valid encoding schemes for input are X'2100', X'3100', X'2200', X'2300', X'2305', X'3200', and X'3300' while X'4403' is valid for the output CCSID.
- The PC input data is always normalized to two bytes per code point.
- The conversion table created will handle either single-byte or double-byte code points from the input CS, CP pair to a possible single-byte, double-byte or triple-byte output CS, CP, as determined by the EUC encoding scheme.
- The content of the table will reflect:
    - CS, CP pair priorities for the EUC CCSID
    - Matched GCGID priority within a CS, CP pair
    - Mismatch management criteria
    - Space character management.
- Since many double-byte encodings do not use all available first-byte values as ward numbers, the conversion table will contain one record for each valid ward and one additional record for all invalid wards. Each record will contain 256 four-byte entries.
- Invalid single-byte code points will be mapped into the single-byte G0 set character SUB, at code point X'1A'. Invalid double-byte values will be mapped into the double-byte G1 set as a SUB.
- Each of the four-byte values will contain the appropriate single-shift character (SS2 or SS3), whenever the output is in G2 or G3.

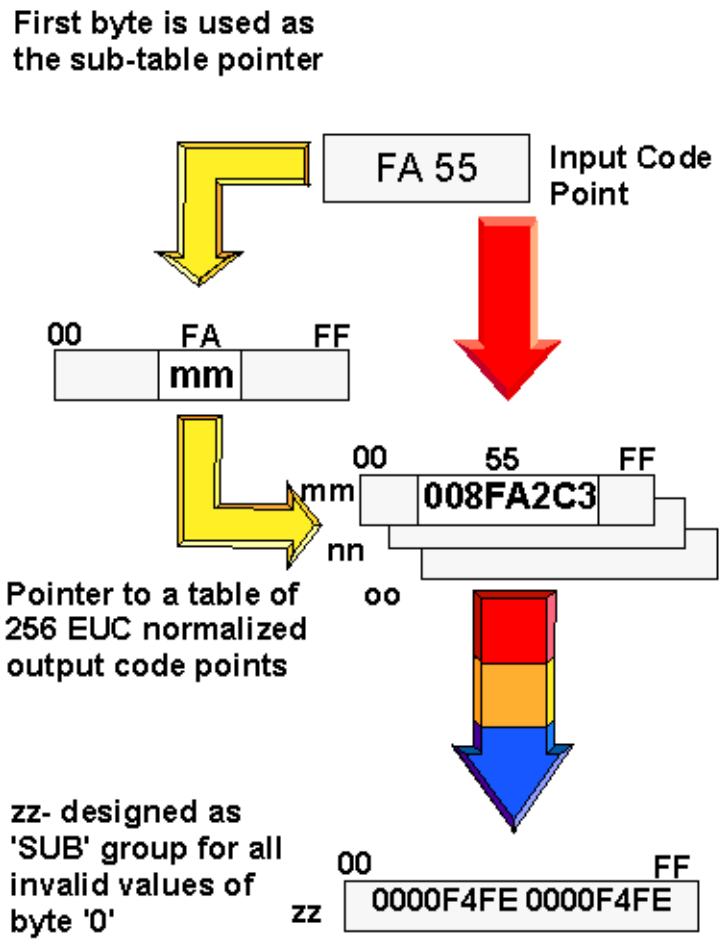


Figure 62. Method 9: PC to EUC Conversion

Method 10 for EUC to PC conversions

The input values for the conversion in this case are four bytes, rather than the two bytes used for the PC input in the previous example. This results in a conversion table construction that is more complex than the PC-to-EUC case. The following description applies equally to all tables dealing with four-byte input values, namely those of EUC and TCP/IP CCSIDs.

There are four levels of tables within the constructed conversion table, where each table corresponds to one input byte value of the four input bytes per character.

- Level 0 tables (B0): Only one table can be constructed at this level. Byte 0 (the first byte) of the input code point is used to index into the B0 table and retrieve a pointer to the B1 level tables. Table B0 is 256 bytes long.
- Level 1 tables (B1): There is one B1 table for each valid entry in the B0 table, plus one table to contain all of the invalid entries for B0. The first four bytes of each B1 table are used as a pointer, b2pt, to a corresponding group of B2 tables. The second byte of the input code point (byte 1) is used as an index into B1 to retrieve the index number for the B2 table within the group of B2 tables pointed to by the b2pt value. Each B1 table is 260 bytes long.
- Level 2 tables (B2): There is one group of B2 tables for each B1 table. The first four bytes of each B2 table are used as a pointer, b3pt, to a corresponding group of B3 tables. The third byte of the input code point (byte 2) is used as an index into B2 to retrieve the index number for the B3 table within the group of B3 tables pointed to by the b3pt value. Each B2 table is 260 bytes long.
- Level 3 tables (B3): There is one group of B3 tables for each B2 table. Use the fourth byte of the input code point (byte 3, where byte 0 is the first byte) to index into the B3 table to retrieve the final conversion value. Each B3 table is 512 bytes in length.

An index value of 0 corresponds to the first table in the group.

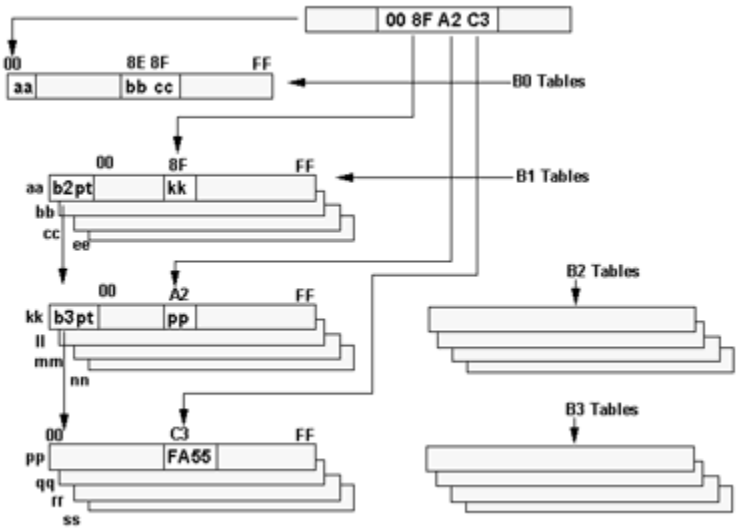


Figure 63. Method 10: EUC to PC Conversion

The method shown in Figure 63 has the following characteristics:

- It is used for the conversion of data between an input EUC CCSID and output PC CCSID.
- The valid encoding scheme for input data is X'4403', while the valid schemes for output data are X'2100', X'3100', X'2200', X'2300', X'2305' X'3200', and X'3300'.
- The input bytes are always received in a normalized four-byte format.
- The conversion table will accept single-byte, double-byte, or triple-byte code points from the input CS, CP pair as defined by the EUC encoding scheme to be converted to a possible single-byte, double-byte CS, CP code-point output value.
- The content of the table will reflect the criterion used for:
    - Matched GCGID priority within the target CS, CP
    - Mismatch management
    - Space character management.
- For most EUC four-byte codes, only a certain range of code point values are valid for the three high-order bytes, therefore the tables are organized as several sub-tables. Sub-table pointer tables contain entries that point to a pool of sub-tables. The lowest-level sub-table points to a series of records containing 256 double-byte code point values used as output.
- Invalid single-byte input values will be mapped to the single-byte SUB character for the PC, which is a X'7F'. All other invalid input values will be mapped to the double-byte SUB for the respective PC mixed CCSID.

- Only a triple-byte CS, CP pair will use all four bytes of the input code point.

Method 11 for Host to EUC conversions

The method shown in Figure 64 has the following characteristics:

- It is used for conversion between an input Host CCSID and an output EUC CCSID.
- The valid encoding schemes for input data are X'1100', X'1200', and X'1301'. The valid output encoding scheme is X'4403'.
- Input is always expected in a normalized two-byte format.
- The conversion table created will handle either single-byte or double-byte code points from the input CS, CP pair to a possible single-byte, double-byte or triple-byte output CS, CP, as determined by the EUC encoding scheme.
- CS, CP pair priorities for the EUC CCSID
- Matched GCGID priority within a CS, CP pair
- Mismatch management criteria
- Space character management.
- Since many double-byte encodings do not use all available first-byte values as ward numbers, the conversion table will contain one record for each valid ward and one additional record for all invalid wards. Each record will contain 256 four-byte entries.
- Invalid single-byte code points (X'00xx') will be mapped into the single-byte G0 set character SUB, at code point X'1A'. Invalid double-byte values will be mapped into the double-byte G1 set as a SUB.
- Host double-byte control characters will be mapped to the single-byte control characters after denormalization.

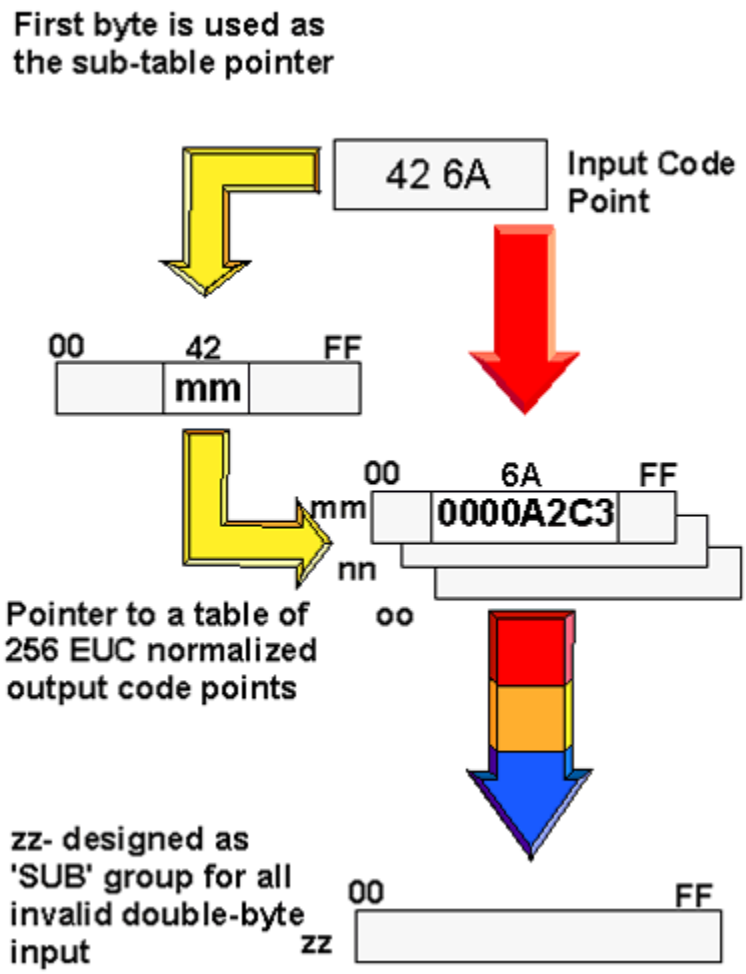


Figure 64. Method 11: Host to EUC Conversion

Method 12 for EUC to Host conversions

The method shown in Figure 65 has the following characteristics:

- See “Method 10 for EUC to PC Conversion” for a description of the table format.
- It is used for conversion between an input EUC CCSID and an output HOST CCSID.
- The valid encoding scheme for input data is X'4403'. The valid output encoding schemes are X'1100', X'1200', X'1301'.
- Input is always expected in a normalized four-byte format.
- The conversion table created will handle either single-byte, double-byte or triple-byte code points from the input CS, CP pair to a possible single-byte or double-byte CS, CP code point output.
- CS, CP pair priorities for the EUC CCSID
- Matched GCGID priority within a CS, CP
- Mismatch management criteria
- Space character management.
- Since most EUC four-byte encodings only use a certain range for the three high-order bytes, the conversion table is organized into several levels of sub-tables. These sub-tables in turn point to a pool of records containing 256 double-byte entries. There is a sub-table data record code point for each valid input code point.
- Invalid single-byte code points (X'00xx') will be mapped into the single-byte G0 set character SUB, at code point X'3F'. Invalid multi-byte values will be mapped into the double-byte host SUB, at X'FEFE'.
- Only a triple-byte CS, CP pair will use the high-order byte of the four-byte encoding space.

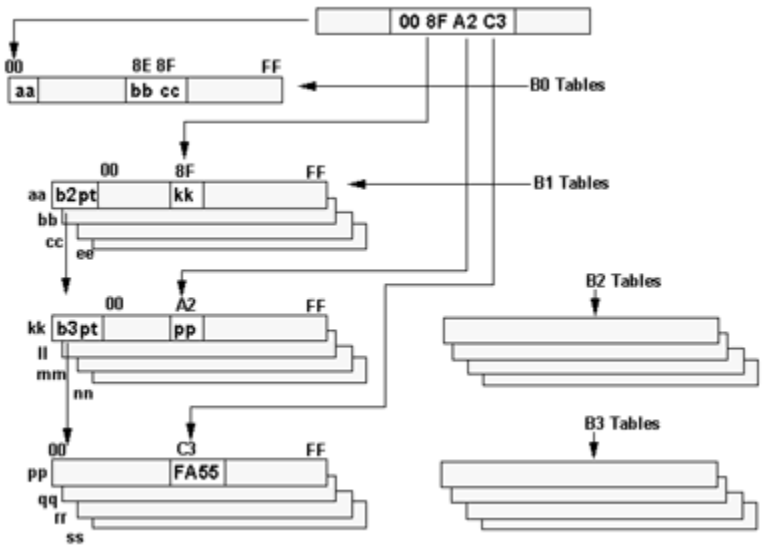


Figure 65. Method 12: EUC to Host Conversion

Method 13 for PC to TCP conversions

The method shown in Figure 66 has the following characteristics:

- It is used for conversion between an input PC CCSID and an output TCP CCSID
- The valid encoding schemes for input data are X'2100', X'3100', X'2200', X'2300', X'2305', X'3200', and X'3300'. The valid output encoding scheme is X'5404'
- Input is always expected in a normalized two-byte format
- The conversion table created will handle either single-byte or double-byte code points from the input CS, CP pair to a possible single-byte, double-byte or triple-byte output CS, CP, as determined by the TCP encoding scheme
- CS, CP pair priorities for the TCP CCSID
- Matched GCGID priority within a CS, CP pair
- Mismatch management criteria
- Space character management.
- Since many double-byte encodings do not use all available first-byte values as ward numbers, the conversion table will contain one record for each valid ward and one additional record for all invalid wards. Each record will contain 256 four-byte entries
- Invalid single-byte code points (X'00xx') will be mapped into the single-byte character SUB, at code point X'1A'. Invalid double-byte values will be mapped into the following:
- Japan - X'747E'
- Korea - X'2F7E'
- Traditional Chinese - X'7D7E'
- Simplified Chinese - X'2121'
- The high-order byte of each output code point value will contain the identifier 1 to 4 for graphics or 0 for a control character.

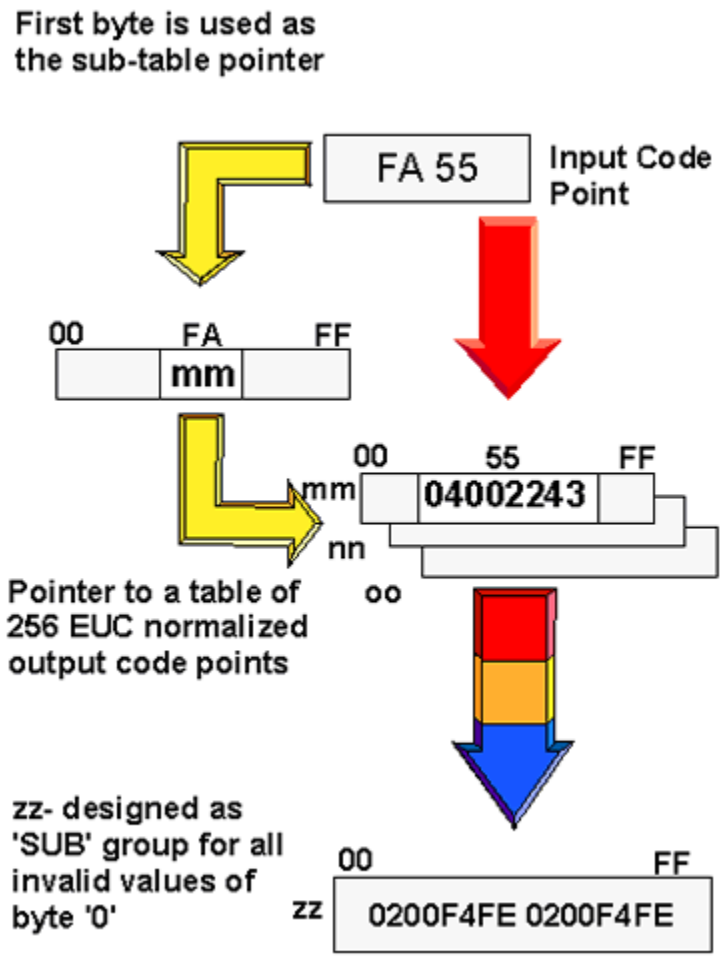


Figure 66. Method 13: PC to TCP Conversion

Method 14 for TCP to PC conversions

The method shown in Figure 67 has the following characteristics:

- See “Method 10 for EUC to PC Conversions” for a description of the type of table format used in this conversion method.
- It is used for conversion between an input TCP CCSID and an output PC CCSID
- The valid encoding scheme for input data is X'5404'. The valid output encoding schemes are X'2100', X'3100', X'2200', X'2300', X'2305', X'3200', and X'3300'
- Input is always expected in a normalized four-byte format, and it includes the identifier in the high-order byte indicating which coded graphic character set the code point was taken from.
- The conversion table created will handle either single-byte, double-byte, or triple-byte code points from the input CS, CP pair as defined by the TCP encoding scheme to a possible single-byte, or double-byte output CS, CP code point.
- Matched GCGID priority within a CS, CP
- Mismatch management criteria
- Space character management.
- For most TCP four-byte codes, only a certain range of values is valid for the three high-order bytes. To handle this situation, the table is organized as a series of sub-tables. Each sub-table level points to a lower level sub-table, until the last sub-table level points to the actual output code point records. Each of the records contain 256 double-byte code point values.
- Invalid single-byte code points will be mapped into the single-byte character SUB, at code point X'7F'. All other invalid values will be mapped to the double-byte SUB character for the respective country version of the encoding scheme.
- Only a triple-byte CS, CP pair will use all four bytes of the input code point.

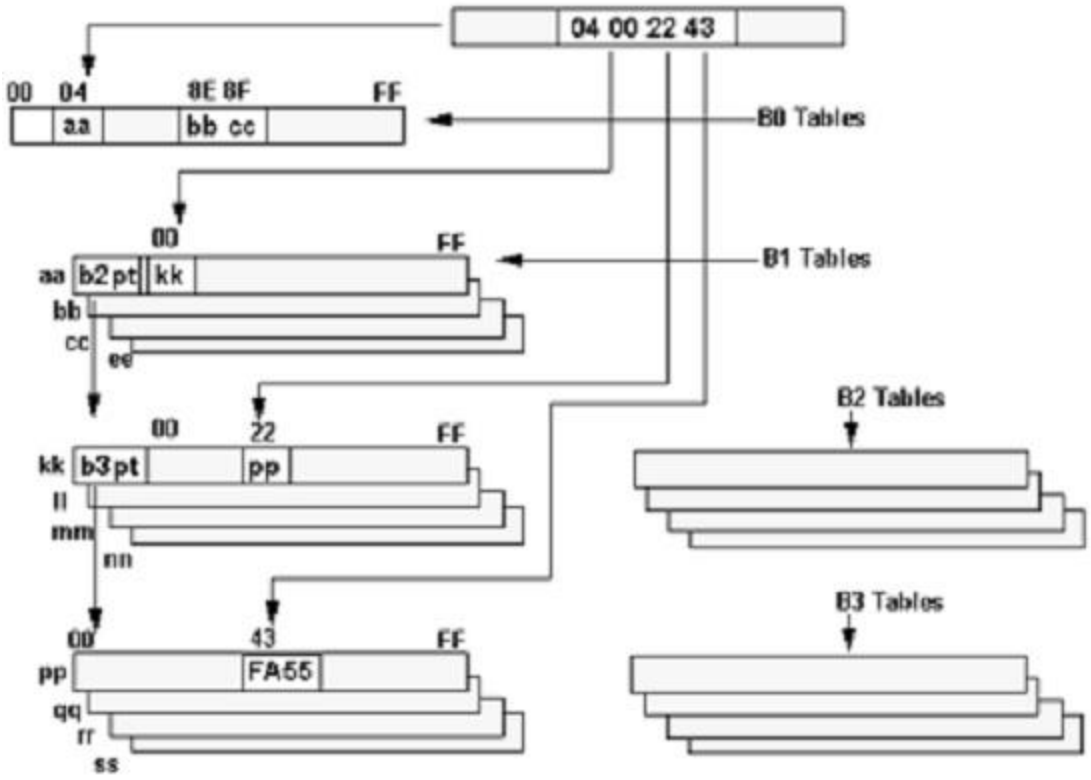


Figure 67. Method 14: TCP to PC Conversion

Method 15 for Host to TCP conversions

The method shown in Figure 68 has the following characteristics:

- It is used for conversion between an input Host CCSID and an output TCP CCSID.
- The valid encoding schemes for input data are X'1100', X'1200', and X'1301'. The valid output encoding scheme is X'5404'.
- Input is always expected in a normalized two-byte format.
- The conversion table created will handle either single-byte or double-byte code points from the input CS, CP pair to a possible single-byte, double-byte, or triple-byte output CS, CP code point as defined by the TCP encoding scheme.
- CS, CP pair priorities for the TCP CCSID
- Matched GCGID priority within a CS, CP pair
- Mismatch management criteria
- Space character management.
- Most double-byte encodings do not use all the available first bytes as valid ward numbers. To handle this situation and make effective use of table resource space, the table is organized as a series of sub-tables. Each sub-table contains 256 four-byte code point entries. There is a sub-table of output code points for each valid ward number of the input code points, and a single sub-table for the substitution entries for all of the invalid first-byte values.
- Invalid single-byte code points (X'00xx') will be mapped into the single-byte character SUB, at code point X'1A'.
- Invalid double-byte values will be mapped as follows:
- Japan - X'747E'
- Korea - X'2F7E
- Traditional Chinese - X'7D7E'
- Simplified Chinese - X'2121'The high order byte of each output code point contains the identifier from 1 to 4 for graphic characters, or 0 for control characters.
- Host double-byte control characters will be mapped to single-byte control characters after denormalization.

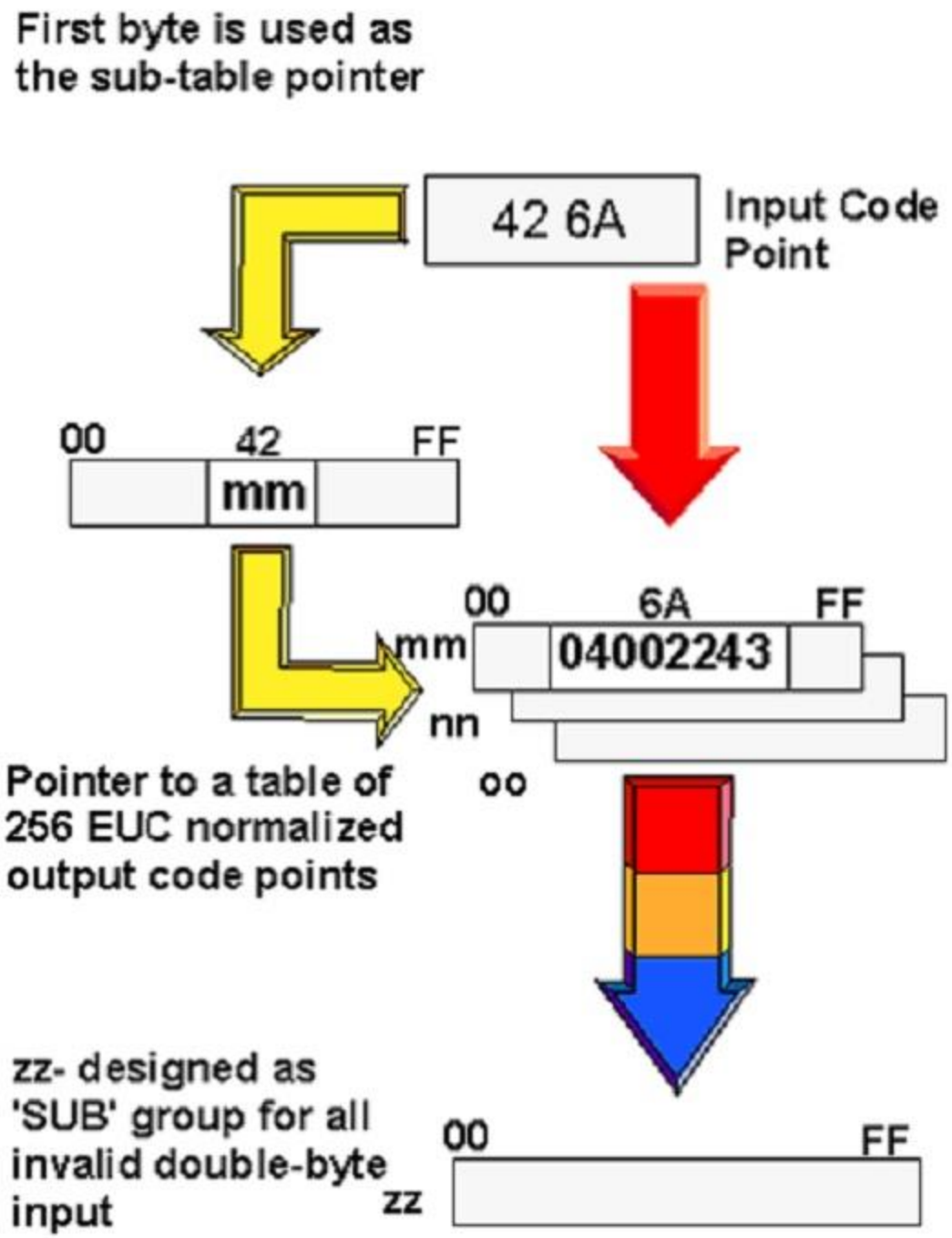


Figure 68. Method 15: Host to TCP Conversion

Method 16 for PC to TCP conversions

The method shown in Figure 69 has the following characteristics:

- See “Method 10 for EUC to PC Conversions” for a description of the type of table format used in this conversion method.
- It is used for conversion between an input TCP CCSID and an output Host CCSID.
- The valid encoding scheme for input data is X'5404'. The valid output encoding schemes are X'1100', X'1200', and X'1301'.
- Input is always expected in a normalized four-byte format and includes the identifier in the high-order byte that indicates the coded graphic character set of the output code point.
- The conversion table created will handle either single-byte, double-byte, or triple-byte code points from the input CS, CP pair as defined by the TCP encoding scheme to a possible single-byte, double-byte, or triple-byte output CS, CP code point.
- The content of the table will reflect:
  - Matched GCGID priority within a CS, CP
  - Mismatch management criteria
  - Space character management.
- For most TCP four-byte codes, only a certain range of values are valid for the three high-order bytes, causing the table to be organized as several sub-tables. Each of the sub-tables contains pointers to subsequent records in the table. Each of the subsequent records contains 256 double-byte output code point entries.
- Invalid single-byte code points will be mapped into the single-byte character SUB, at code point X'3F'.
- Invalid multi-byte input values will be mapped to the host double-byte SUB value of X'FEFE'.
- Only a triple-byte CS, CP pair will use all four bytes of the four-byte input code point value.

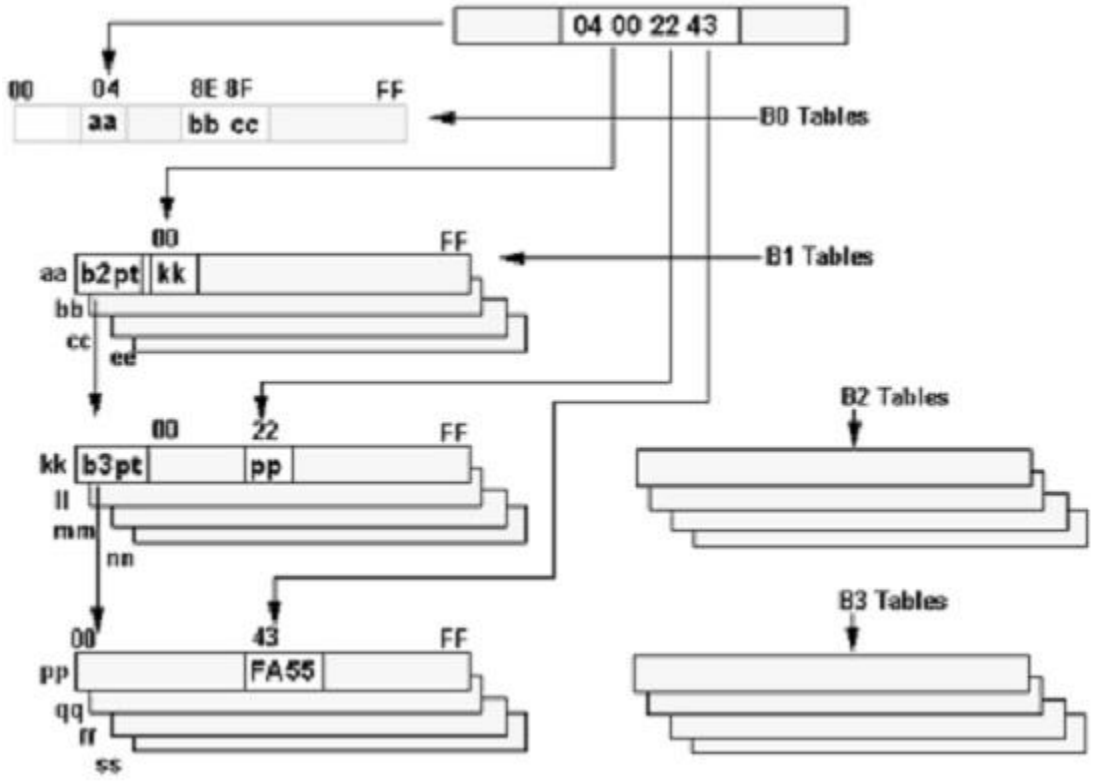


Figure 69. Method 16: TCP to Host Conversion

Conversion methods in support of GB18030

Chinese Standard GB18030 defines a complex code composed of one, two and four-byte components. To convert between GB18030 and other encodings including Unicode and mixed single- double-byte codes, conversion table methods and structures have been defined. Specifics of the methods and table structures are detailed below.

GB18030 Conversion Documentation

This document contains descriptions of conversion tables and associated methods for converting data between Chinese Standard GB18030, Unicode and EBCDIC encodings.

Table of Contents

Introduction

What is Unicode?

Unicode is an international standard for the universal character encoding scheme for written characters and text. It defines a consistent way of encoding multilingual text that enables the exchange of text data internationally, while also creating the foundation for global software. The Unicode standard is a superset of all characters in widespread use today. It contains the characters from major international and national standards as well as prominent industry character sets. Versions of the Unicode Standard are fully compatible and synchronized with the corresponding versions of International Standard ISO/IEC 10646. For example, Unicode 6.1 contains all the same characters and code points as ISO/IEC 10646:2012.The Unicode Standard provides additional information about the characters and their uses. Any implementation that is conformant to Unicode is also conformant to ISO/IEC 10646

A complete description of Unicode including code point assignments, encoding forms and the principles of the standard can be found on the Unicode web site.

What is GB18030?

GB 2312-80 (the primary collection of Chinese coded graphic characters published in 1981 as a national standard) covers only 6,763 Chinese characters. In 1995, GBK (Chinese Internal Code Specification) for GB Extension was published. It is the superset of GB and completely compatible with GB 2312-80. GBK expands its character set to 20,902 characters.

GB18030 was defined in order to meet the needs of Chinese customers such as financial institutions, insurance companies and postal services that require name and address information. The characters included in this super set meet these needs.

In GB 18030, one-byte, two-byte and four-byte encoding systems are adopted. The total capability is over 1.5 million code positions. Currently, GB 18030 contains more than 27,000 Chinese characters which have been defined in Unicode 3.0. This standard provides a solution for the urgent need for the Chinese characters used in names and addresses.

Table 1: Allocation of Code Ranges

Number of Bytes	Space of Code Positions				Number of Codes
One-byte	X'00'-X'80'				129 codes
Two-byte	First byte		Second byte		23,940 codes
	X'81'~X'FE'		X'40'~X'7E' X'80'~X'FE'		
Four-byte	First byte	Second byte	Third byte	Fourth byte	1,587,600 codes
	X' 81'~X'FE'	X' 30'~X'39'	X' 81'~X'FE'	X' 30'~X'39'	

Conversion Between GB18030 and Unicode

The GB 18030 standard contains all characters defined in Unicode, but they have completely different code assignments. All 1.1 million Unicode code points, U+0000-U+10FFFF (except for surrogates U+D800-U+DFFF), map to and from GB 18030 code points. Most of these mappings can be done algorithmically, except for parts of the BMP. This makes it an unusual mix of a Unicode encoding and a traditional code page.

This document provides descriptions for two sets of tables and their associated methods. The first set of tables and methods (identified as Combined GB18030 in this document) are the default CDRA conversion tables and methods for mapping between GB18030 and Unicode. The Combined GB18030 mapping uses three binary tables (1:2 byte, 2:2 byte and 4:2 byte) as well as transformation logic to do the conversion from GB18030 to Unicode. The conversion logic uses the Code Ranges (see Table 1) to select the appropriate binary table from the three existing tables. For the Unicode to GB18030 conversion, the Combined GB18030 method uses a single binary table along with the transformation logic. The binary table is a 2:4 byte mapping table where the input code is a 2 byte Unicode code and the output is a normalized 4 byte GB18030 code.

The second set of tables and methods (identified as Components GB18030 in this document) are custom CDRA conversion tables used by z/OS only. The Components GB18030 uses three binary tables (1:2 byte, 2:2 byte and 4:2 byte) as well as transformation logic to perform the conversion from GB18030 to Unicode. Another

three binary tables (2:1 byte, 2:2 byte and 2:4 byte) and the transformation logic are used to perform the conversion from Unicode to GB18030. In this case, choosing the appropriate binary table from the three available binary tables is done by trial and error. The conversion logic is explained in detail later in this document.

UTF-16 <-> GB18030 (1, 2 and 4-byte)

This section contains information required to perform conversions between UTF-16 formatted Unicode data and GB18030. The tables referenced in the following sections are available from the CDRA Conversion Table Repository.

Text Format Tables

The conversion table repository contains both text and binary conversion tables. The following are the text, human readable, conversion tables for conversions between UTF-16 and GB18030.

Combined GB18030 Tables

UCS_GB18030.TXT (2000-11-30) - Source mapping file between GB18030 and UTF-16 from Chinese Government sources

157004B0.TPMAP100 - GB18030 (CCSID 5488) to UTF-16 (CCSID 1200)

04B01570.RPMAP100 - UTF-16 (CCSID 1200) to GB18030 (CCSID 5488)

157004B0.UPMAP100 - GB18030 (CCSID 5488) to and FROM UTF-16 (CCSID 1200)

Component GB18030 Tables

24E404B0.TPMAP100 - Text table from GB 1-byte part (CCSID 9444) to UTF-16 (CCSID 1200)

04B024E4.RPMAP100 - Text table from UTF-16 (CCSID 1200) to GB 1-byte part (CCSID 9444)

24E404B0.UPMAP102 - Text table between GB 1-byte part (CCSID 9444) and UTF-16 (CCSID 1200)

256904B0.TPMAP100 - Text table from GB 2-byte part (CCSID 9577) to UTF-16 (CCSID 1200)

04B02569.RPMAP100 - Text table from UTF-16 (CCSID 1200) to GB 2-byte part (CCSID 9577)

256904B0.UPMAP102 - Text table between GB 2-byte part (CCSID 9577) and UTF-16 (CCSID 1200)

156F04B0.TPMAP100 - Text table from GB 4-byte part (CCSID 5487) to UTF-16 (CCSID 1200)

04B0156F.RPMAP100 - Text table from UTF-16 (CCSID 1200) to GB 4-byte part (CCSID 5487)

156F04B0.UPMAP102 - Text table between GB 4-byte part (CCSID 5487) and UTF-16 (CCSID 1200)

Table files with the extension RPMAPnnn, TPMAPnnn, and UPMAPnnn contain human readable formats. They have two columns containing the source code point value (in Hex), and the target code point value (in Hex). Each file contains a brief header and column descriptions. The header in each file contains information including the values of the defined SUB characters as well as any special handling requirements.

Converting from UTF-16 to GB18030

Combined GB18030 Tables

In the combined tables, all GB code points are normalized to 4 bytes by adding leading zero-bytes to the single and double-byte values. This normalization allows for the conversion table to be a fixed 2-byte to 4-byte structure. The logic of the associated conversion method strips out the leading zero-bytes before inserting target code point into the output data stream. Each single-byte target code point has 3 leading zero bytes while each double-byte target code point has 2 leading zero bytes. For example output target code point x'5B' will be represented as x'0000005B' in the conversion table while x'8147' will be represented as x'00008147'.

2-byte to 4-byte Conversion

04B01570.UGN-R-D

When converting a UTF-16 data stream to GB18030, it is done on a character by character basis. The first step when examining an input code point is to determine if it is a valid high order surrogate value. If it is, then next code point is taken from the input stream to determine if it is a valid low order surrogate value. If together the two points comprise a valid surrogate pair then the UTF-16 to GB18030 algorithmic transformation will be used to convert the pair to the appropriate GB18030 code. Otherwise, the two bytes will be fed into the 2:4 binary table as described later (see Method 2X).

In the algorithmic transformation the target code point is calculated as follows: (Note: * = multiplication; / = division; % = modular operation; - = subtraction; + = addition in the following equations)

index = (source_codepoint1 - X'D800')*1024+source_codepoint2-X'DC00';

b0=index/12600+X'90';

b1=index/1260%10+X'30';

b2=index/10%126+X'81';

b3=index%10+X'30';

target_code point=b0*X'1000000'+b1*X'10000'+b2*X'100'+b3;

For more details see the section on transformations.

Method 2X

The binary conversion table is similar to the tables used in existing CDRA Method 2, but extended to handle the normalized GB18030 4-byte code. The input data stream consists of UTF-16 2-byte code points. The output from the conversion table will be 4-byte normalized GB18030 code points. These 4-byte codes must be de-normalized before being inserted into the output data stream.

Assumed normalization for GB18030:

GB Byte	*Normalized*	*Comment*
xx (Single byte)	000000xx	Four byte, with 3 leading zero bytes
xxxx (Double byte)	0000xxxx	Four byte, with 2 leading zero bytes
xxxxxxxx (Four byte)	xxxxxxxx	Four byte

To describe the conversion method we first define the concept of a "ward". A ward is a section of a double-byte code page. It is equivalent to a "row" of code points in ISO/IEC 10646 and in Unicode. All of the code points contained in a specific ward begin with the same first byte. A ward is populated if there is at least one character in the double-byte code page (UTF-16 in this case) whose first byte is the ward value. There are 256 wards numbered X'00' to X'FF'.

This 2 to 4-byte binary conversion table is made up of several 1024-byte vectors. The first vector acts as an index into the rest of the table. It contains 256 two-byte vector numbers (corresponding to each of the 256 wards, for a total of 512 bytes) and the remaining 512 bytes are unused (filled with zeros). There is one 1024-byte vector for each populated ward in the source code page and one additional vector used for mapping all unassigned and invalid wards.

The method extracts two bytes at a time from the input data stream. The first byte is used as a pointer into the index vector (shown in Figure 1.1). Each vector number in the index vector is two bytes long. Therefore the first byte from the input code point is multiplied by two before calculating the offset into this index vector. The two-byte value found at the corresponding position in the index vector gives the vector number in which to perform the second lookup.

The second byte of the input code point is used as a pointer into the vector specified by the index vector. When calculating the offset into this vector there are two things to remember; first, the indexing starts at zero, and second, each entry is four bytes long (normalized GB18030 code point).

In the example shown in Figure 1.1 the input code point is X'4E02'. Taking the first byte, x’4E’ and multiplying by 2 you get x’9C’ as the pointer value into the index vector. The index vector specifies x’0050’ as the vector where the output code point will be found. Taking the second byte of the input code point, x’02’ and multiplying by 4 (each output code point is 4-bytes long), indicates that the output code point will be found in the specified vector (x’50’) at location x’0008’. In the example, the resultant 4-byte output code point is x’00008140’. This value would subsequently be de-normalized to the 2-byte value x’8140’ prior to being placed in the output data stream.

In the example vector (X'0001') is used for handling code points from invalid or unassigned wards in the input data. All of the 256 four-byte (normalized) code point values found in this vector are those of the "Substitute" (SUB) character of the

target code page (X'8431A437' for GB18030). All of the entries in the index vector for unused and invalid wards point to this "substitute" vector.

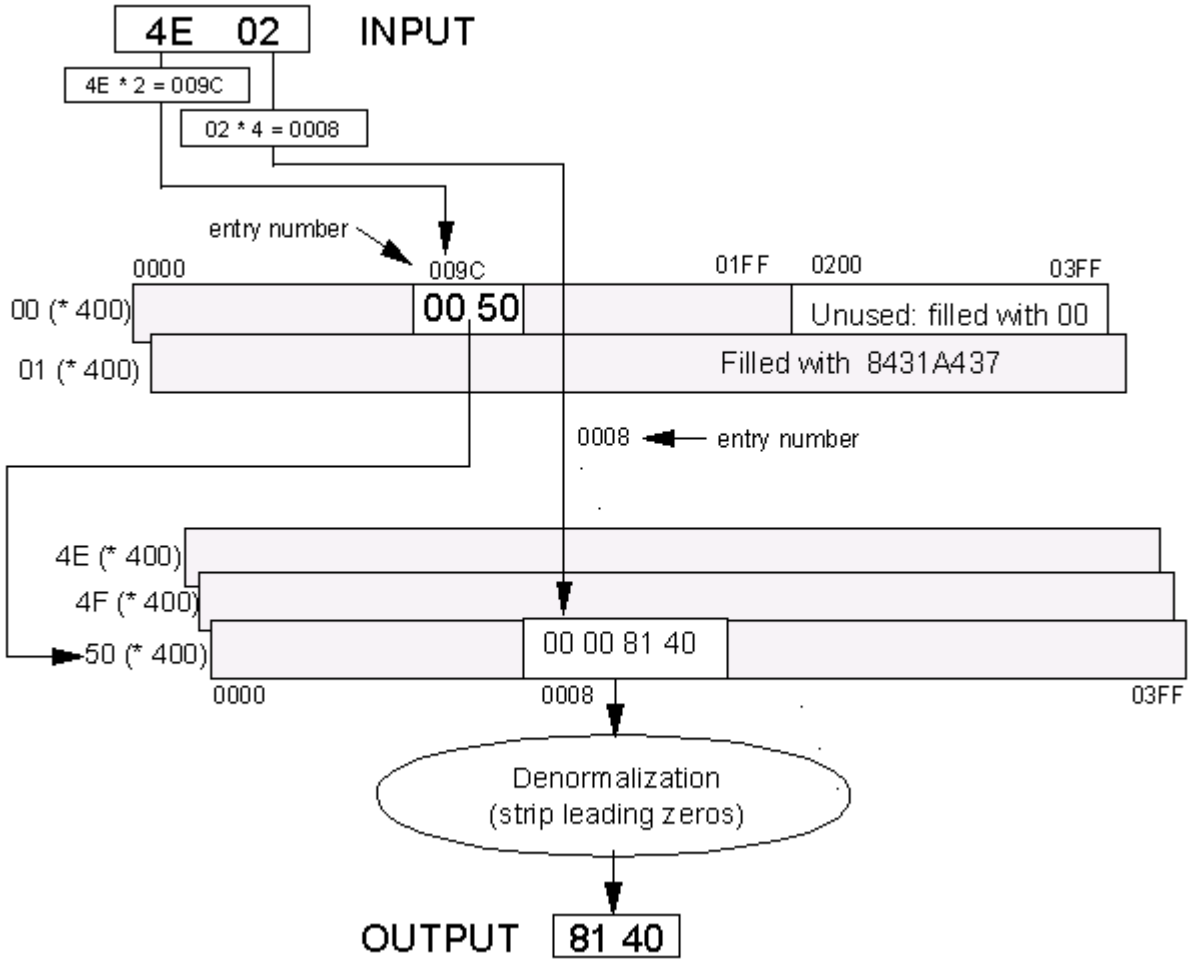


Figure 1.1 - **UCS-2 to GB18030 Binary Table & Conversion Method**

Note: For Phase 1, code points in the surrogate area (X'D800' through X'DFFF') for UCS-2 are invalid. If there is a need to distinguish between invalid and unassigned code points in the input, the method should detect and map input code points in this range to output SUB code points. In Phase 2, the surrogate area will be used for UTF-16 representation of Planes 1 to 16.

For the example shown in figure 1.1, the target four bytes could be found at the following positions in the binary table.

Byte	Value	Position (hex)	Position (decimal)
first	0	X'0050' * X'400' + X'0008'	80 * 1024 + 8 = 81928
second	0	X'0050' * X'400' + X'0008' + 1	80 * 1024 + 8 + 1 = 81929
third	81	X'0050' * X'400' + X'0008' + 2	80 * 1024 + 8 + 2 = 81930
fourth	40	X'0050' * X'400' + X'0008' + 3	80 * 1024 + 8 + 3 = 81931

De-normalization:

Since the resultant single-bytes and double-bytes in the table have been prefixed with leading zero-bytes, when composing the output string the leading zero-bytes are removed from the 4-byte output, (three zero-bytes for single-byte and two zero-bytes for double-byte).

Components for GB18030

Introduction

z/OS implementation does not use the combined tables. In the z/OS environment a call is made to CONVERT DATA indicating that the source is Unicode and the target CCSID is 5488. The converter logic looks up CCSID 05488 and gets the component conversion tables and sets up 2 to 1, 2 to 2 and 2 to 4 logic and resources. UTF-16 contains 2-byte code points and GB18030 contains 1, 2, and 4-byte code points. According to the z/OS conversion method, in order to do conversions from UTF-16 to GB18030 three binary tables are required. They are the following:

04B024E4.UG1-C0-A1 - Binary table that maps Unicode (1200) to GB 1-byte part (9444)
(CDRA 2-bytes to 1-byte conversion method, with X'FF' as STOP)

04B02569.UG2-C0-A1 - Binary table that maps Unicode (1200) to GB 2-byte part (9577)
(CDRA 2-byte to 2-byte conversion method, with X'FFFF' as STOP)

04B0156F.UG4-C0-A1 - Binary table that maps Unicode (1200) to GB 4-byte part (5487)
(CDRA 2-byte to 4-byte conversion method for GB18030, with X'FFFFFFFF' as STOP)

All of the Unicode code points located beyond the basic multilingual plane (BMP) are mapped to GB18030 4-byte code points. The GB18030 4-byte code points when converted to Unicode must be represented as either UTF-16 surrogate pairs or as 4-type, UTF-32 values.

The high level logic of this type of conversion is as follows:

The UTF-16 input stream (sequences of two-bytes) will be fed into the conversion logic. Within the conversion logic, the UTF-16 input stream will be passed through the following steps:

2:1 logic use UTF-16 to GB18030 single-byte binary table to get GB18030 single-byte output (see 2-byte to 1-byte conversion for more details)

2:2 logic use UTF-16 to GB18030 double-byte binary table to get GB18030 double-byte output (see 2-byte to 2-byte conversion for more details)

2:4 logic detects valid surrogate high and surrogate low pair, then use UTF-16 to GB18030 transformation to get GB18030 four-byte output. (see 2-byte to 4-byte conversion for more details)

2:4 logic use UTF-16 to GB18030 four-byte binary table to get GB18030 four-byte output (see 2-byte to 4-byte conversion for more details)

All invalid and incomplete surrogate pairs are dealt with separately.

The structure of the binary tables used for simulating the z/OS logic is described below:

The UTF-16 to single-byte GB18030 binary table consists of a collection of 256-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see 2-byte to 1-byte conversion for more details).

The UTF-16 to double-byte GB binary table is made up of a collection 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see 2-byte to 2-byte conversion for more details).

The UTF-16 to four-byte GB binary table is made up of a collection 1024-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see 2-byte to 4-byte conversion for more details).

Conversion Logic

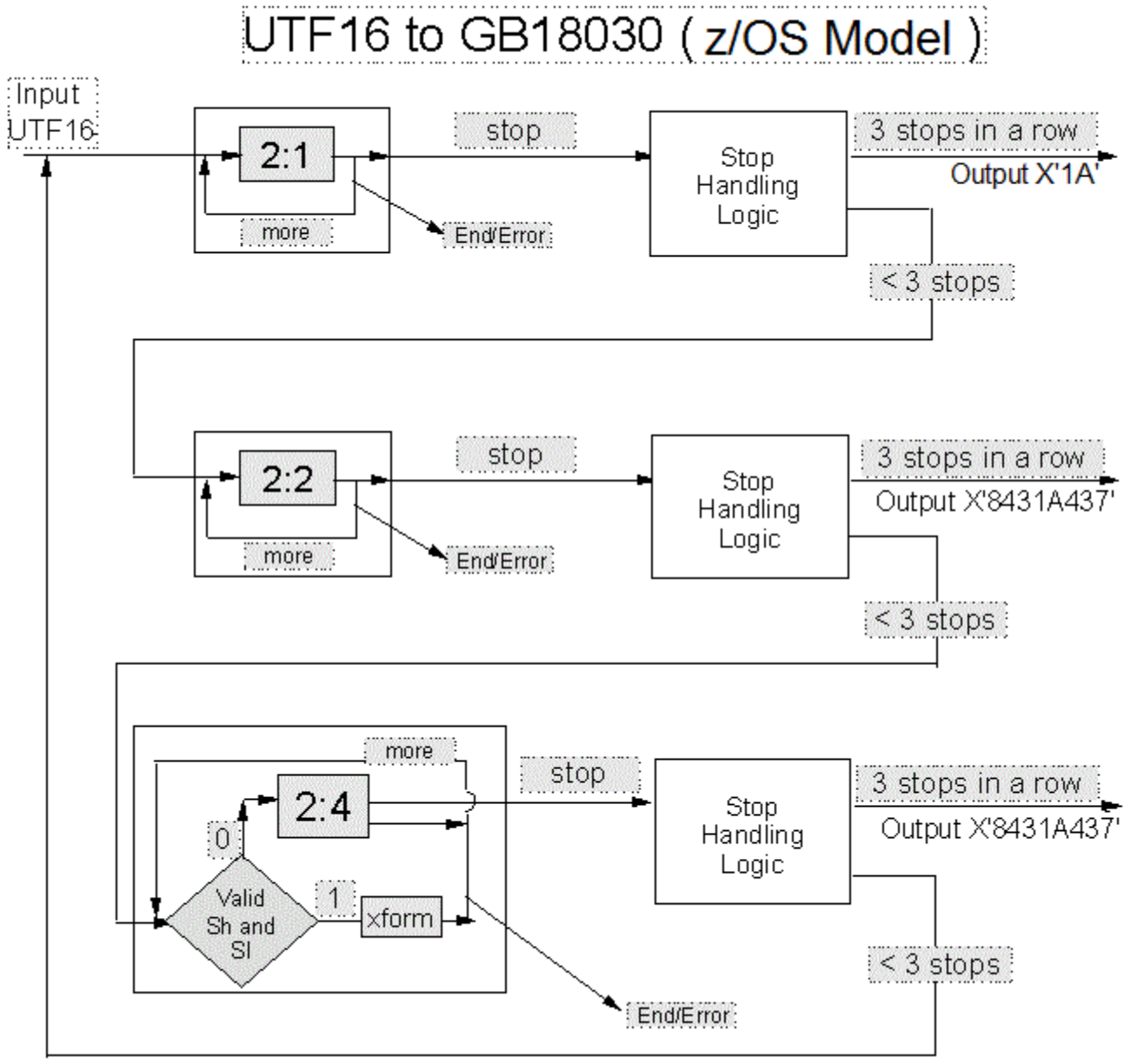


Figure 1.2 - **UTF-16 to GB18030 Conversion Method**

In understanding the UTF-16 to GB18030 z/OS conversion method, we refer to figure 1.2 for a graphical representation of the following description. Note that there exist 3 binary tables; separate tables for dealing with 2:1 byte, 2:2 byte and 2:4 byte mappings. Any input code point that has no defined mapping within any of the three tables will be detected and substitute. The precise means by which this is accomplished is described later in detail.

In this model, the UTF-16 input stream will first be fed into the conversion logic. The logic initially starts with the 2:1 mapping stage of the conversion operation. In this stage of the conversion, each two-byte segment from the UTF-16 input stream is entered into the 2:1 binary table in an attempt to convert. If the bytes are successfully converted, a valid single-byte output code is found, the conversion continues with the next two-byte input code point. This process continues until the

output code is the defined ‘stop’ (x’FF’) or the end of the input steam is encountered or an error in the input stream is detected. If an end or error condition is encountered, execution will be terminated. When a ‘stop’ character is encountered, the *stop handling* logic will be executed and the process will go on to the 2:2 mapping stage of the conversion operation.

In the 2:2 stage of operation, each two-byte segment from the UTF-16 input stream is entered into the 2:2 binary table in an attempt to convert. If the bytes are successfully converted, a valid 2-byte output code is found, the conversion continues with the next two-byte input code point. This process continues until the output code is the defined ‘stop’ (x’FFFF’) or the end of the input steam is encountered or an error in the input stream is detected. If an end or error condition is encountered, execution will be terminated. When a ‘stop’ character is encountered, the *stop handling* logic will be executed and the process will go on to the 2:4 mapping stage of the conversion operation.

In the 2:4 stage of operation, the first two bytes from the input stream will be taken to check whether it is valid surrogate high. If it is a valid surrogate high then the next two bytes will be taken from the input stream to check whether it is a valid surrogate low. If it is proven to be a valid surrogate low, then the UTF-16 to GB18030 algorithmic transformation converts the pair of valid surrogates. Otherwise, the two-byte segment from the UTF-16 input stream is entered into the 2:4 binary table in an attempt to convert. If the bytes are successfully converted, a valid 4-byte output code is found, the conversion continues with the next two-byte input code point. This process continues until the output code is the defined ‘stop’ (x’FFFFFFFF’) or the end of the input steam is encountered or an error in the input stream is detected. If an end or error condition is encountered, execution will be terminated. When a ‘stop’ character is encountered, the *stop handling* logic will be executed and the process will loop back to the 2:1 mapping stage of the conversion operation.

Figure 1.3 shows the ‘stop handling’ logic in detail

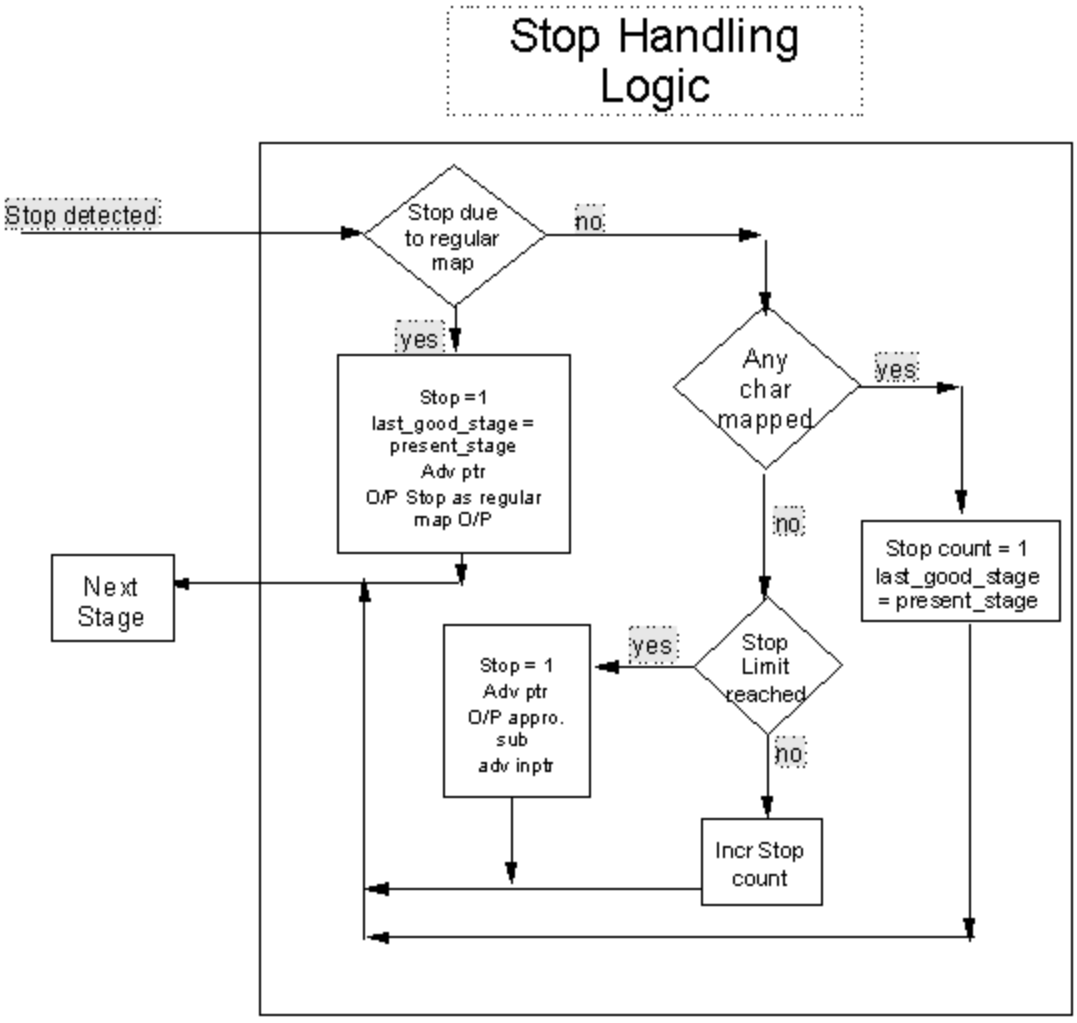


Figure 1.3 - **UTF-16 to GB18030 Conversion Method**

The stop handling logic is an integral part of the algorithm's ability to determine whether or not SUBs are to be output and detect the presence of regular mappings. This task is accomplished with the assistance of a 'stop limit' counter that effectively keeps track of the number of successive stops in the algorithm and 'last_good_stage' flag. It also remembers the last successful stage of conversion, and takes appropriate action.

In the stop handling logic, the logic first checks whether this stop is due to a regular mapping or not. If it is due to a regular mapping, the stop count will be reset to 1 and the last successful stage of operation is also set to the current mode of operation. The input pointer will be moved to the next position in the input stream while the stop (X'FFFF') character goes to the output as a target code point.

An important point to be aware of in this logic is that stop handling logic is stage dependent. The regular mapping could differ from stage to stage. So the stop handling logic in each stage should take care of the cases related to that particular stage.

If the stop is not due to the regular mapping, then the stop handling logic checks whether any successful conversion has taken place in this stage of operation. This

can be done by checking whether the input pointer has moved or not. If there is any successful conversion then the stop count will be reset to 1 and the last successful stage of operation is also set to the current stage of operation before we move on to the next stage of operation. If there is no mapping in this stage of operation then the logic checks whether the stop limit has been reached or not. The stop limit has been reached this indicates that each stage of the conversion operation (2:1, 2:2 and 2:4) has been executed without finding a successful mapping for the input 2 byte value. If this is the case then a sub is output as target according to the last successful stage of conversion. The stop count will be reset to 1 and the input pointer will be advanced to the next position.

If the stop limit is not reached, then the stop count will be incremented and the execution goes to the next stage of operation.

2-byte to 1-byte Conversion

File: 04B024E4.UG1-C0-A1

This table with its associated method is used to find the target code point when it is a single-byte.

Table format:

The binary conversion table is created by using the existing CDRA Method 6. Figure 1.4 illustrates the table format and the associated method.

The UCS-2 to single-byte table consists of a several 256-byte vectors (256 entries). The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector. The method takes each double-byte source code point and separates it into a first and second byte. The first byte is used as an offset into the index vector. The value found at this location "points" to the appropriate vector in the pool of vectors. The second byte is then used as an offset into the selected vector. The value found at this location is the single-byte target code point. Each target code point in the vector is one byte long. The binary conversion table is equivalent to existing CDRA US-R-D binary tables.

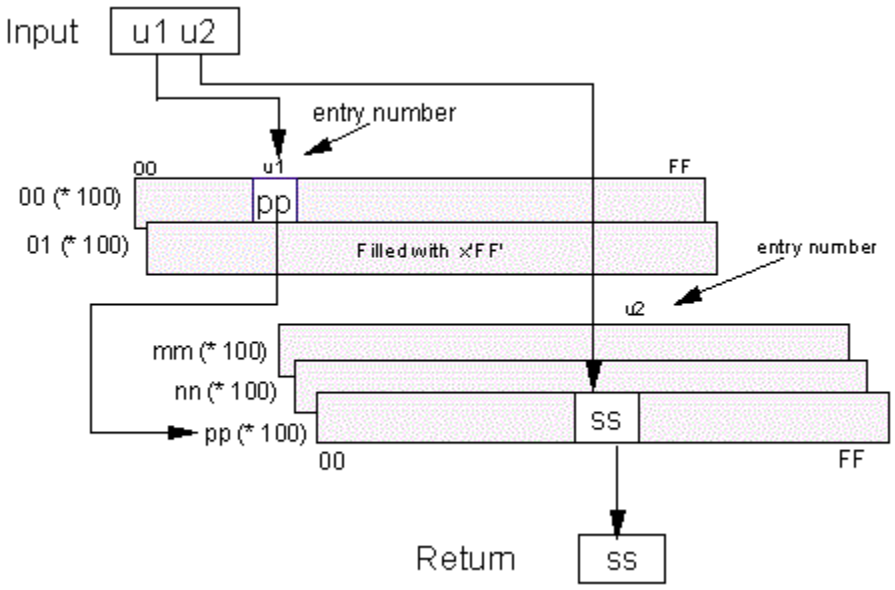


Figure 1.4 - UTF-16 to GB18030 Single-byte Binary Table & Conversion Method

2-byte to 2-byte Conversion

File: 04B02569.UG2-C0-A1

This table is used to find the target code point when the target code point is a double-byte.

Table format:

The binary conversion table is created by using the existing CDRA Method 2, a two-step vector lookup method. The conversion table and the associated method are illustrated in Figure 1.5 below.

This 2-byte to 2-byte table is made up a collection of 512-byte vectors. The first vector contains 256 single-byte pointers (vector numbers) into the rest of the table, followed by 256 unused bytes. The second vector, (the "substitute" vector) contains the mapping for code points in unassigned wards, and is filled with 256 2-byte SUB code points (X'FFFF' stop code point in this case). These are followed by additional 512-byte vectors, one for each populated ward in the source encoding.

The table is used as follows. The first byte of the input code point is used as a pointer into the index vector. The single-byte value found at the corresponding position in the index is the vector number in which to perform the second lookup. The second byte of the input code point is used as a pointer into the vector specified by the index vector.

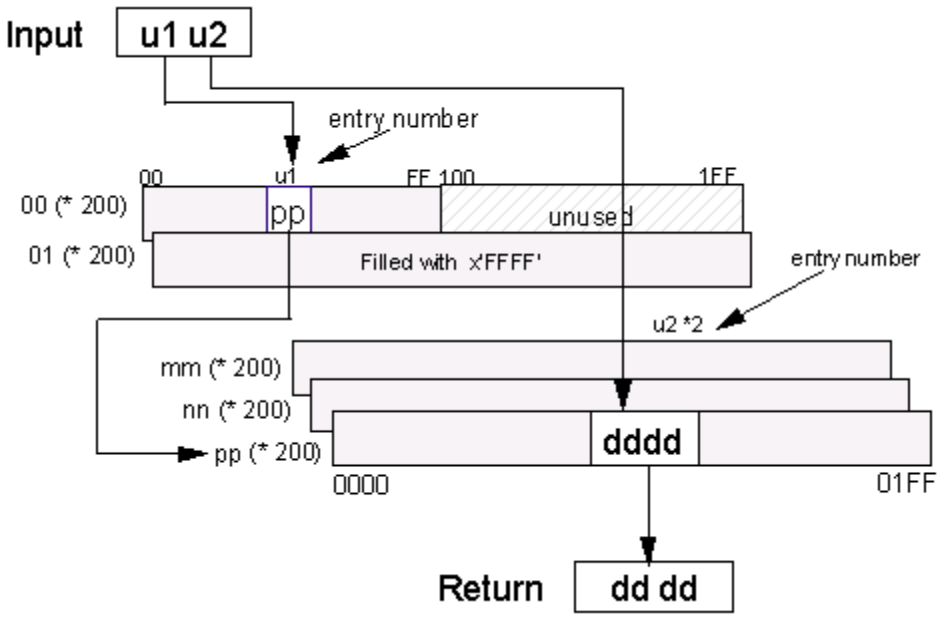


Figure 1.5 - UTF-16 to GB18030 Double-byte Binary Table & Conversion Method

In Figure 1.5 the input code point is X'u1u2'. The first byte, x'u1', is used as a pointer into the index to determine which vector in the table to use to obtain the output code point. In this case the output vector is x'pp'. The second byte, x'u2' is then used as a pointer into the specified vector to obtain the resultant double-byte output code point. Note that this value must be multiplied by 2 before the lookup is performed since each output entry is 2 bytes long.

The second vector mentioned for handling unassigned wards works as follows. All of the 256 double-byte code point values found in this vector (vector X'01') are those of the "Substitute (SUB)" character of the target encoding (in this case Stop character X'FFFF'). All the entries in the index vector for unassigned wards point to this "substitute" vector.

The binary conversion table is equivalent to existing CDRA UM-R-D binary tables.

2-byte to 4-byte conversion

File: 04B0156F.UG4-C0-A1

This binary table contains the mapping from UCS-2 to four-byte GB18030.

In this section before looking into the binary table, each UTF-16 input value is checked to see whether it is a valid high surrogate. If it is then the next two bytes segment will be taken from the input stream to check whether it is a valid low surrogate. If the pair is determined to be a valid surrogate pair, then the UTF-16 to GB18030 transformation will take place.

In the transformation the target code point will be calculated as follows:

(Note: * = multiplication; / = division; % = modular operation; - = subtraction; + = addition in the following equations)

index = (source_codepoint1 - X'd800')*1024+source_codepoint2-X'dc00';

b0=index/12600+X'90';

b1=index/1260%10+X'30';

b2=index/10%126+X'81';

b3=index%10+X'30';

target_code point=b0*X'1000000'+b1*X'10000'+b2*X'100'+b3;

For more details on transformation please see the transformations section of this document.

If the pair is not a valid surrogate pair, then the two bytes will be fed into the 2:4 binary table as described below.

Method 2X

The binary conversion table is similar to the tables used in existing CDRA Method 2, but extended to handle the GB18030 4-byte code. The input data stream consists of UTF-16 2-byte code points. The output from the conversion table will be 4-byte GB code points.

To describe the conversion method we first define the concept of a "ward". A ward is a section of a double-byte code page. It is equivalent to a "row" of code points in ISO/IEC 10646 and in Unicode. All of the code points contained in a specific ward begin with the same first byte. A ward is populated if there is at least one character in the double-byte code page (UTF-16 in this case) whose first byte is the ward value. There are 256 wards numbered X'00' to X'FF'.

This binary conversion table is made up of several 1024-byte vectors. The first vector acts as in index into the rest of the table. It contains 256 two-byte vector numbers (corresponding to each of the 256 wards for a total of 512 bytes) and the remaining 512 bytes are unused (filled with zeros). There is one 1024-byte vector for each populated ward in the source code page and one additional vector used for mapping all unassigned and invalid wards.

The method extracts two bytes at a time from the input data stream. The first byte is used as a pointer into the index vector (shown in Figure 1.6). Each vector number in the index vector is two bytes long. Therefore the first byte from the input code point is multiplied by two before calculating the offset into this index vector. The two-byte value found at the corresponding position in the index vector gives the vector number in which to perform the second lookup.

The second byte of the input code point is used as a pointer into the vector specified by the index vector. When calculating the offset into this vector there are two things to remember; first, the indexing starts at zero, and second, each entry is four bytes long (GB18030 code point).

In the example shown in Figure 1.6 the input code point is X'4D02'. Taking the first byte, x'4D' and multiplying by 2 you get x'9A' as the pointer value into the index vector. The index vector specifies x'004F' as the vector where the output code point will be found. Taking the second byte of the input code point, x'02' and multiplying by 4 (each output code point is 4-bytes long), indicates that the output code point will be found in the specified vector (x'4F') at location x'0008'. In the example, the resultant 4-byte output code point is x'8234F437'. This value would placed in the output data stream.

The one additional vector (X'0001') mentioned for handling code points from invalid or unassigned wards in the input data is used as follows. All of the 256 four-byte code point values found in this vector are those of the "Substitute" (SUB) character of the target code page (here in this case X'FFFFFFFF' stop character). All of the entries in the index vector for unused and invalid wards point to this "substitute" vector.

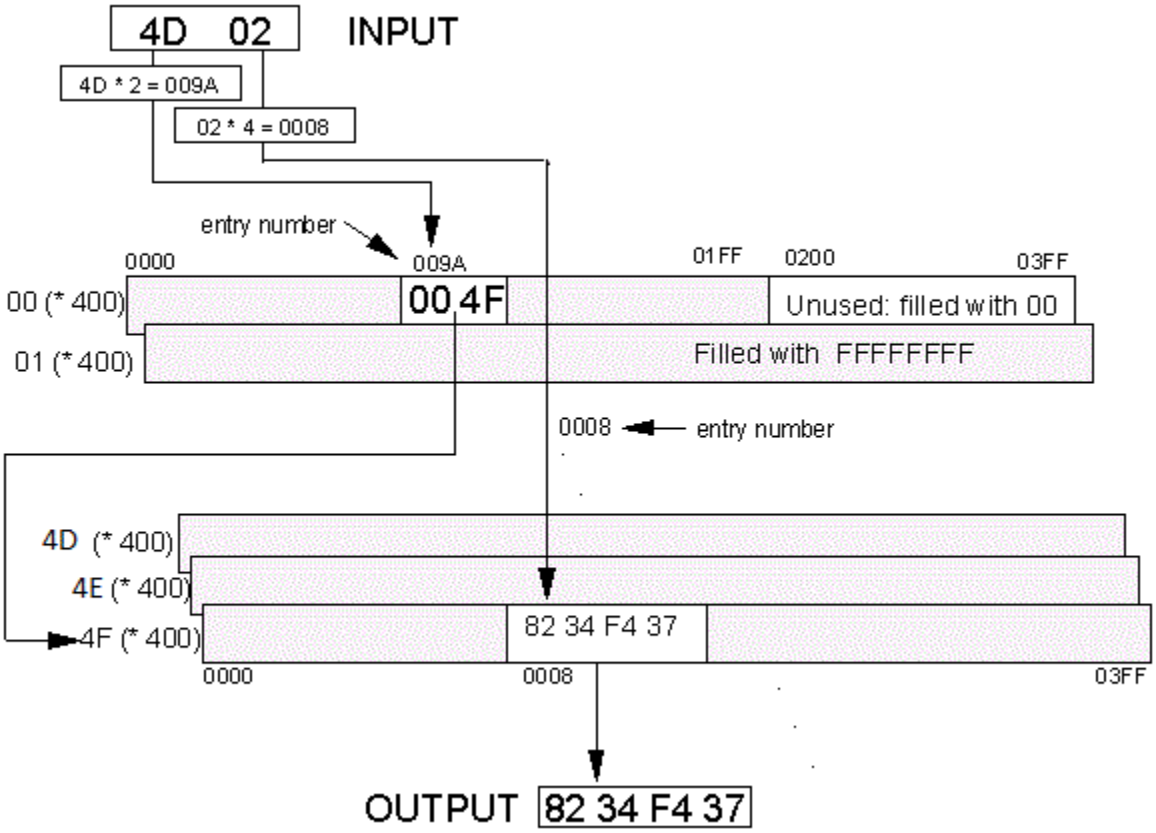


Figure 1.6 - UTF-16 to GB18030 Four-byte Binary Table & Conversion Method

In the binary table, target four-bytes could be found at the following positions.

Byte	Value	Position (hex)	Position (decimal)
first	82	X'004F' * X'400' + X'0008'	79 * 1024 + 8 = 80896
second	34	X'004F' * X'400' + X'0008' + 1	79 * 1024 + 8 + 1 = 80897
third	F4	X'004F' * X'400' + X'0008' + 2	79 * 1024 + 8 + 2 = 80898
fourth	37	X'004F' * X'400' + X'0008' + 3	79 * 1024 + 8 + 3 = 80899

Converting from GB18030 to UTF-16

Combined GB18030 Tables

Introduction

Following is a description of a GB to UCS conversion, as an alternative to following the CDRA's EUC normalization method and the resulting table structure. The normalization steps of detecting when it is a single, double, or four-byte is followed. However, instead of normalizing the data as in the EUC case, this method takes three branches in the logic and comes up with associated data structures. The data structures are linear arrays – one for each of the input types – single, double and four-bytes. The indexing operations get into these arrays only for valid ranges of code points. All other code points and broken multi-byte sequences are trapped and are dealt with separately in the logic.

The high level logic is:

- Detect valid single, double or four-byte code points (see Detecting Valid (Single, Double, and Four-byte) or Invalid Code Points for more details)

- Use single-byte to UCS-2 array for single-byte input (see 1-byte to 2-byte conversion for more details)

- Use double-byte to UCS-2 array for double-byte input (see 2-byte to 2-byte conversion for more details)

- Use four-byte to UCS-2 compact array for four-byte input (see 4-byte to 2-byte conversion for more details)

- All invalid and incomplete sequences dealt with separately (see 2-byte to 4-byte conversion for more details)

The main advantage of this method is to be able to keep the conversion tables as compact as possible (especially for the 4-byte to 2-byte part) and to be able to get at the converted UCS-2 code points in a relatively fast manner.

The single-byte to UCS-2 array consists of a single 512-byte vector (256 2-byte entries).

The double-byte GB to UCS-2 array is made up of several 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see 2-byte to 2-byte conversion for more details)

The four-byte to UCS-2 compact array is made up of:

- 256 bytes of flag values (followed by 3*256 bytes of unused bytes)
- Four 1024-byte long vectors, each containing 256 4-byte index values, and
- A long compact array containing target two-byte UCS-2 code points.

Detecting Valid (single, Double, and Four-byte) or Invalid Code Points

Refer to Figure 1.7 below. The method fetches one byte at a time from the input stream. The first byte, b0, is checked to see whether it is in the valid range of single-byte or start of a double-byte or four-byte code point. If it is not, then a SUB code point (X'001A' in this case) is inserted in the output data stream, and the pointer to the input stream is incremented by one.

If b0 is in the valid range of single-byte (X'00' - X'80'), then it is passed to the 1-byte to 2-byte conversion as a valid single-byte code point. The resultant 2-byte output is placed in the output buffer, and the pointer into the input stream is incremented by one.

(Note: X'80' is a valid but unassigned code point, per confirmation received through IBM China from Chinese sources. In one of the early conversion tables, it was mapped to the EURO SIGN (U+20AC) in UCS-2. In subsequent tables this has been removed.)

If b0 is in the range X'81' - X'FE' for a valid first byte of a double-byte or four-byte code point, then the next byte, b1, is fetched and checked to see whether it is in the valid range for a second byte of a double-byte (X'40' - X'7E' or X'80' - X'FE') or a second byte of a four-byte (X'30' - X'39') code point. If it is not, then the SUB code point X'001A' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b1 is within the valid range for a second byte of a double-byte code point (X'40' - X'7E' or X'80' - X'FE'), the sequence b0 b1 (or the input pointer) is passed to the 2-byte to 2-byte conversion (see 2-byte to 2-byte conversion). The resultant two-byte output is placed in the output stream, and the pointer into the input stream is incremented by two.

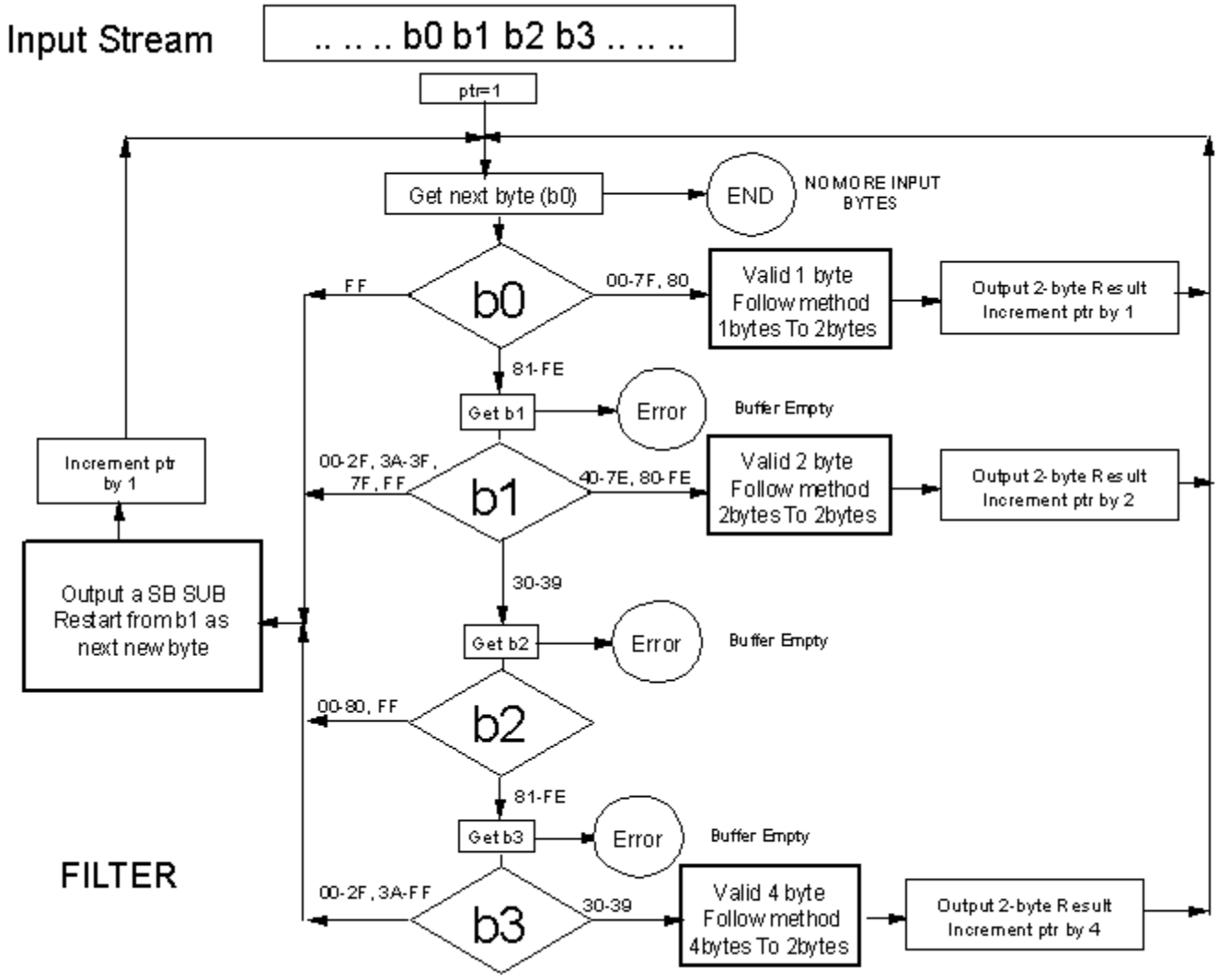


Figure 1.7 - Input Filter for GB18030-1 stream<

If b1 is within the valid range for a second byte of a four-byte code point (X'30' - X'39'), then the next byte in the input stream, b2, is checked whether its in the valid range for a third byte of a four-byte code point (X'81' - X'FE'). If it is not, then the SUB code point X'001A' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b2 is within the valid range for a third byte of four-byte code point, the next input byte, b3, is checked whether it is in the valid range for the fourth byte (X'30' - X'39'). If it is not, then the SUB code point X'001A' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b3 is within the valid range for the fourth byte then the sequence of bytes, b0 b1 b2 b3, (or the input pointer) is passed to the 4-byte to 2-byte conversion (see 4-byte to 2-byte conversion) as valid four-byte code point. The resultant two-byte output is

placed in the output stream, and the conversion pointer into the input stream is incremented by four.

The above steps are repeated until there is no more data in the input stream. If the input stream is exhausted during fetching of any of the second, third or fourth bytes in the above filtering logic, then the conversion logic cannot proceed and there will be some unconverted data in the input stream. The pointer into to the input stream indicates the first of the remaining bytes that have not been converted. An implementation may choose to deal with such a situation in any suitable manner.

The insertion of X'001A' for the first byte of a broken sequence permits locating where such a sequence might have been in the input by examining the output stream.

1-byte to 2-byte Conversion

File: 157004B0.G1U-R-D

This table with its associated method is used to find the target code point when the source code point is a valid single-byte GB18030 code.

Table format:

The binary conversion table is created by using the existing CDRA Method 5. Figure 1.8 illustrates the table format and the associated method.

The single-byte to UCS-2 table consists of one 512-byte vector (256 2-byte entries). The source code point is used as a pointer to determine which 2 bytes in the vector represent the target code point. Each target code point in the vector is two bytes long. Therefore the input code point is multiplied by two before calculating the offset into this vector. The binary conversion table is the same format as existing CDRA SU-R-D binary tables.

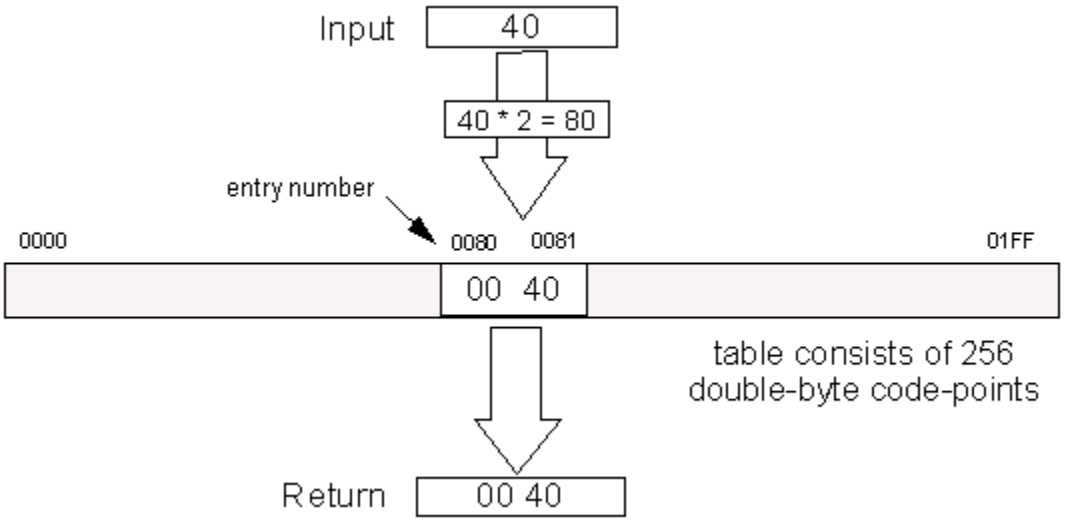


Figure 1.8 - GB 1-byte to UCS-2 2-byte Table and Method

2-byte to 2-byte Conversion

File: 157004B0.G2U-R-D

This table is used to find the target code point when the source code point is a valid double-byte GB18030 code.

Table format:

The binary conversion table is created by the existing CDRA Method 2, a two-step vector lookup method. This is similar to the one described in 2-byte to 4-byte conversion except here each vector is 512 bytes long (256 double-bytes) instead of 1024 bytes (256 four-bytes).

The conversion table and the associated method are illustrated in Figure 1.9 below.

This 2-byte to 2-byte table is made up of several 512-byte vectors. The first vector contains 256 single-byte indices (vector numbers) into the rest of the table, followed by 256 unused bytes. The second vector, the "substitute" vector contains the mapping for code points in unassigned wards, and is filled with 256 2-byte SUB code points (X'FFFD' in this case). These are followed by 512-byte vectors, one for each populated ward in the source encoding.

The table is used as follows. The first byte of the input code point is used as a pointer into the index vector. The single-byte value found at the corresponding position in the index is the vector number in which to perform the second lookup. The second byte of the input code point is used as a pointer into the vector specified by the index vector.

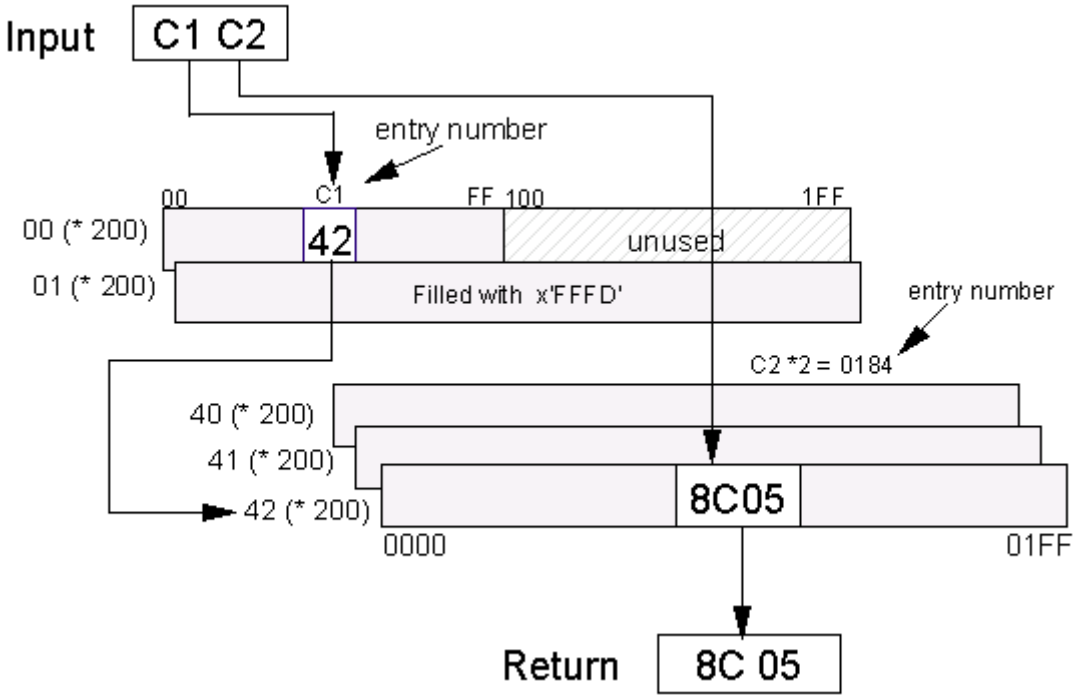


Figure 1.9 - GB 2-byte to UCS-2 2-byte Table and Method

For example (see Figure 1.9) if the input code point is X'C1C2', you would find a vector number at the X'C1' position in the index vector. The resultant double-byte output code point is found in the specified vector number (x'42') beginning at position X'0184' (which is two times X'C2'), counting into the vector starting at zero.

The second vector mentioned for handling unassigned wards works as follows. All of the 256 double-byte code point values found in this vector (vector X'01') are those of

the "Substitute (SUB)" character of the target encoding (X'FFFD' for UCS-2). All the entries in the index vector for unassigned wards point to this "substitute" vector.

The binary conversion table is the same format as existing CDRA MU-R-D binary tables.

4-byte to 2-byte Conversion

File: 157004B0.G4U-R-D

Table format:

The binary conversion table format and the associated method are detailed below.

This binary table consists of five 1024-byte vectors followed by a large linear array corresponding to the table structure as shown in Figure 1.10. There will be one three-dimensional sub array for each first byte (b0) value used in the table definition. Each sub array will contain the minimum number of cells - 12600 cells - needed to map the valid ranges of the second (b1, 10 values), the third (b2, 126 values) and the fourth (b3, 10 values) bytes of four-byte code points in GB. Each cell contains the corresponding 2-byte UCS-2 code point.

The first 256 bytes of this binary table, called the b0_used_array, is used to check if a particular value of b0 byte is used in the conversion table definition. The next 768 bytes (=3*256) are unused. These are followed by four 1024-byte vectors, K40, K41, K42, and K43. These vectors contain values used in computing an index into the rest of the table to determine the location of the output code point.

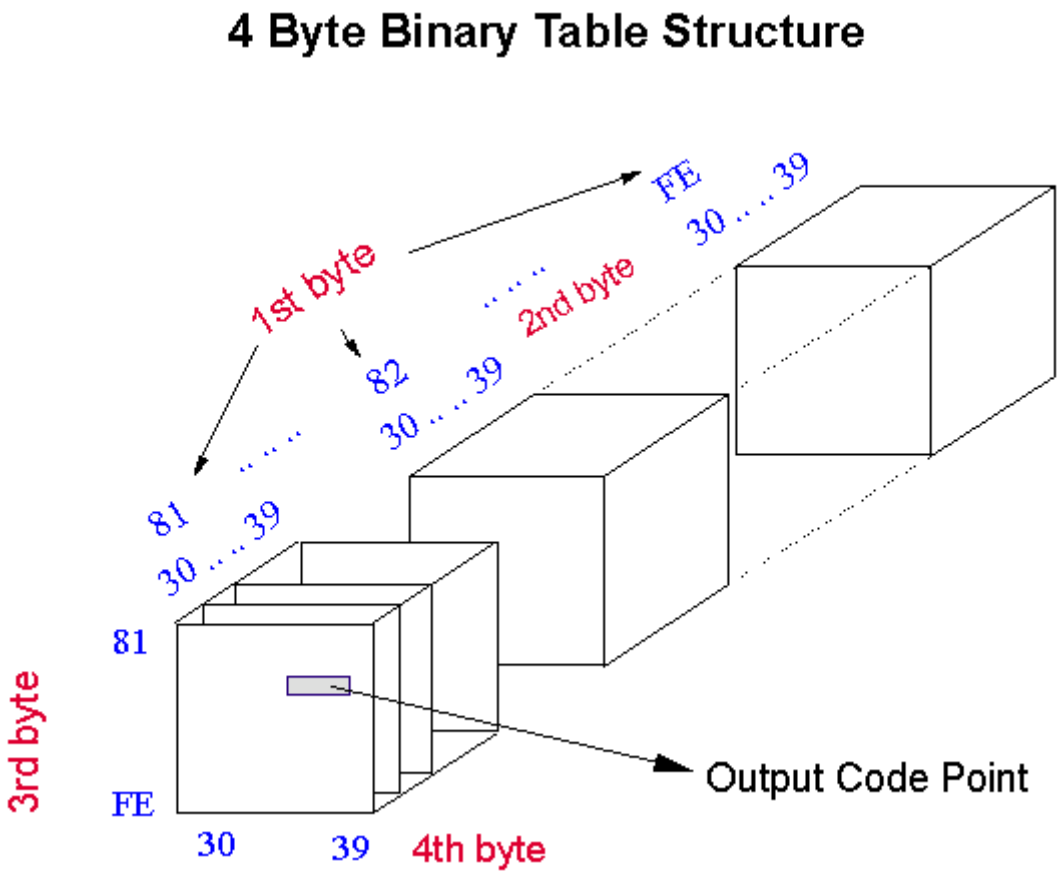


Figure 1.10 - GB 4-byte to UCS-2 2-byte table structure

For a given 4-byte code point (b0 b1 b2 b3), if the b0 is not used then there will not be a sub array populated for it. Therefore, before trying to find the target code point, the b0 value is checked to confirm that it has been used. This is checked by using the b0_used_array. As shown in Figure 1.11, b0 is used as an offset into b0_used_array. If the b0 value has been used, then the corresponding entry in the b0_used_array will be 01. If it has not been used the entry will be 00. If it is a 00, then the code point (b0 b1 b2 b3) is unassigned and its mapping is not defined in the conversion table. In this case a double-byte SUB code point (X'FEFE') is inserted by the conversion logic into the output data stream and the pointer into the input stream is incremented by 4.

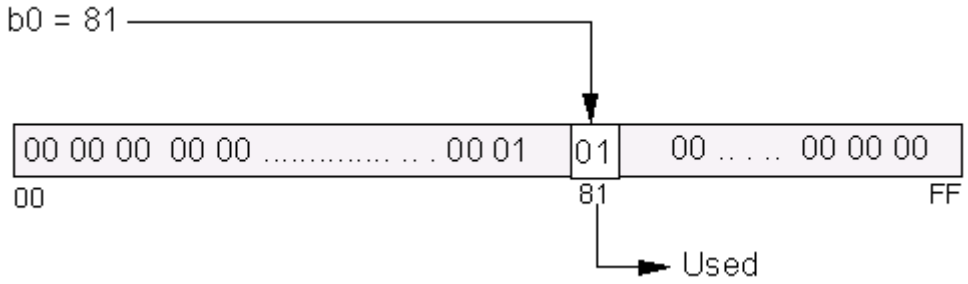


Figure 1.11 - b0_used_array

If b0 is equal to 01, then proceed with the steps to find target code point, as illustrated in Figure 1.12, and described below.

Get the computed index values from the four index vectors, K40, K41, K42, and K43 using b0, b1, b2, and b3 byte values of the input code point. Each byte of the valid four-byte source code point is used as an offset into the corresponding K4x table to get a part of the index value. When calculating the offset into these vectors there are two things to remember; first, you must begin counting at zero, and second, each entry is four bytes long. This means you have multiply the b0, b1, b2, or b3 by four before looking into the vectors. The resulting 4 values are added to get the location of the first byte of the target code point in the binary table.

The values in the index vectors are computed based on the following formulae:

K40(b0) = 25200 * (block number assigned to the b0 group)

K41(b1) = 2* (b1 - X'30') * (126 * 10)

K42(b2) = 2* (b2 - X'81') * 10

K43(b3) = 2* (b3 - X'30')

This eliminates the multiplication steps during conversion execution, improving the performance. Once these indices are added it will be the index or pointer value to the first byte of the cell containing the output code point in Figure 1.10. Figure 1.12 shows the mapping table of Figure 1.10 in as a linear structure.

Also, note that entries in these index arrays for illegal values of b0, b1, b2, or b3, are filled with zero-bytes. They will never be accessed if the filter logic has been followed correctly. Figure 1.12 also shows a possible entry for b0 = X'FD', Private Use Area, as a possible fifth block. The binary table currently defined contains mapping tables only for b0 values of X'81' to X'84', as defined in the conversion requirements received from Chinese Government sources.

The two bytes of the output code point are inserted into the output data stream. The pointer into the input data stream is incremented by four in the main filter logic described earlier in the section on detecting valid (single, double, and four-byte) or invalid code points.

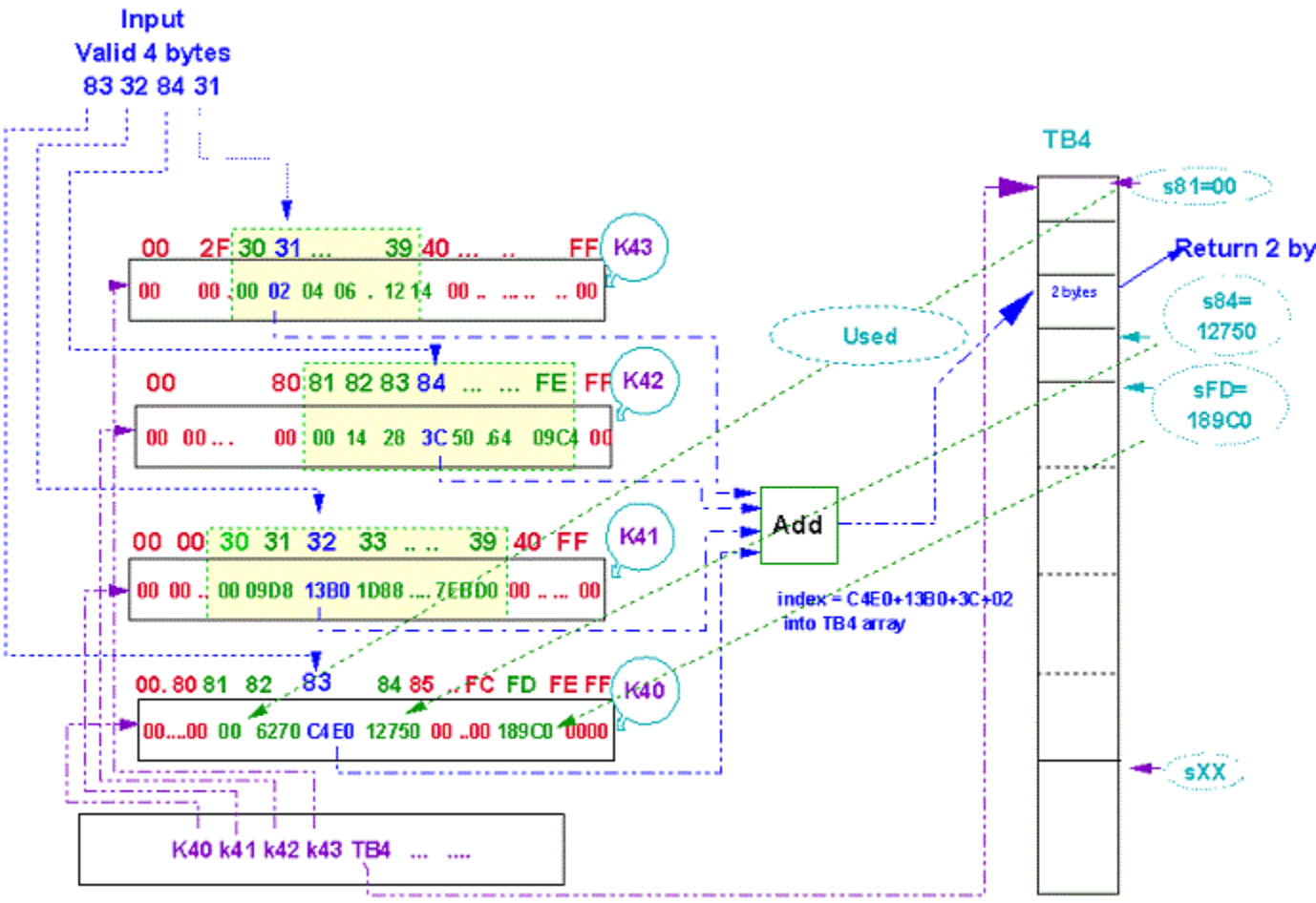


Figure 1.12 - Use of Index Arrays in GB 4-byte to UCS-2 2-byte conversion method

If b0 is not equal to 01 but the four-byte source code point is greater than X'90308130' and less than X'E3329A35', then proceed with the steps to find target code point using the GB18030 to UTF-16 transformation logic described below.

In this example i is a four-byte GB18030 code point.

(Note: * = multiplication; / = division; % = modular operation; - = subtraction; + = addition in the above equations)

b0= (((i >> 24)- 0x90) * 12600);

b1= ((((i >> 16) & 0xFF) - 0x30)*1260);

b2= ((((i >> 8) & 0xFF) - 0x81)*10) ;

b3= ((i & 0xFF)- 0x30);

temp=b0+b1+b2+b3;

utf_32 = temp + 0x10000;

surrogate high=(temp >> 10)+0xD800;

surrogate low=(temp & 0x3FF)+0xDC00;

For more detail see the section on Transformations.

Component GB18030 Tables

Introduction

Gb18030 contains 1, 2, and 4-byte code points and UTF-16 contains 2-byte code points. According to the z/OS logic, in order to do conversion from GB18030 to UTF-16, three binary tables are required. They are the following:

24E404B0.G1U-C0-A1 - Binary table from GB 1-byte part (9444) to UCS (1200)

(CDRA 1-byte to 2-byte conversion method, with X'FFFF' as STOP character)

256904B0.G2U-C0-A1 - Binary table from GB 2-byte part (9577) to UCS (1200)

(CDRA 2-byte to 2-byte conversion method, with X'FFFF' as STOP character)

156F04B0.G4U-C0-A1 - Binary table from GB 4-byte part (5487) to UCS (1200)

(CDRA 4-byte to 2-byte conversion method for GB18030, with X'FFFF' as STOP character)

GB18030 1-byte or 2-byte code points are mapped to the Unicode BMP (plane 0). The GB18030 4-byte code points that map beyond the BMP can be transformed into UTF-16 (surrogate high and low pair) or be represented as a 4-byte UTF-32 code using pure calculations which will be described in detail later.

The high level logic is:

First, the GB18030 input stream (sequence of 1, 2 and 4-bytes) will be fed into the conversion logic. Within the conversion logic, the GB18030 input stream will be passed through the following logic:

- 1:2 logic use GB18030 single-byte to UTF-16 binary table to get UTF-16 double-byte output (see 1-byte to 2-byte Conversion for more details)

- 2:2 logic use GB18030 double-byte to UTF-16 binary table to get UTF-16 double-byte output (see 2-byte to 2-byte Conversion for more details)

- 4:2 logic detects valid four-byte GB18030 code which can be transformed to a valid surrogate pair, using the transformation technique (see 4-byte to 2-byte Conversion for more details).

- 4:2 logic use GB18030 four-byte to UTF-16 binary table to get UTF-16 double-byte output (see 4-byte to 2-byte Conversion for more details)

- All invalid and incomplete sequences are dealt with separately.

The single-byte to UCS-2 array consists of a single 512-byte vector (256 2-byte entries).

The double-byte GB to UCS-2 array is made up of several 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see 2-byte to 2-byte Conversion for more details).

The four-byte to UCS-2 compact array is made up of:

- 256 bytes of flag values (followed by 3*256 bytes of unused bytes)
- Four 1024-byte long vectors, each containing 256 4-byte index values, and
- A long compact array containing target two-byte UCS-2 code points.

GB18030 to UTF-16 Conversion Logic

The following model is similar to the one described in the conversion logic for conversions from UTF-16 to GB18030 except the reverse direction binary tables are used in places where forward direction binary tables were used in the section mentioned above.

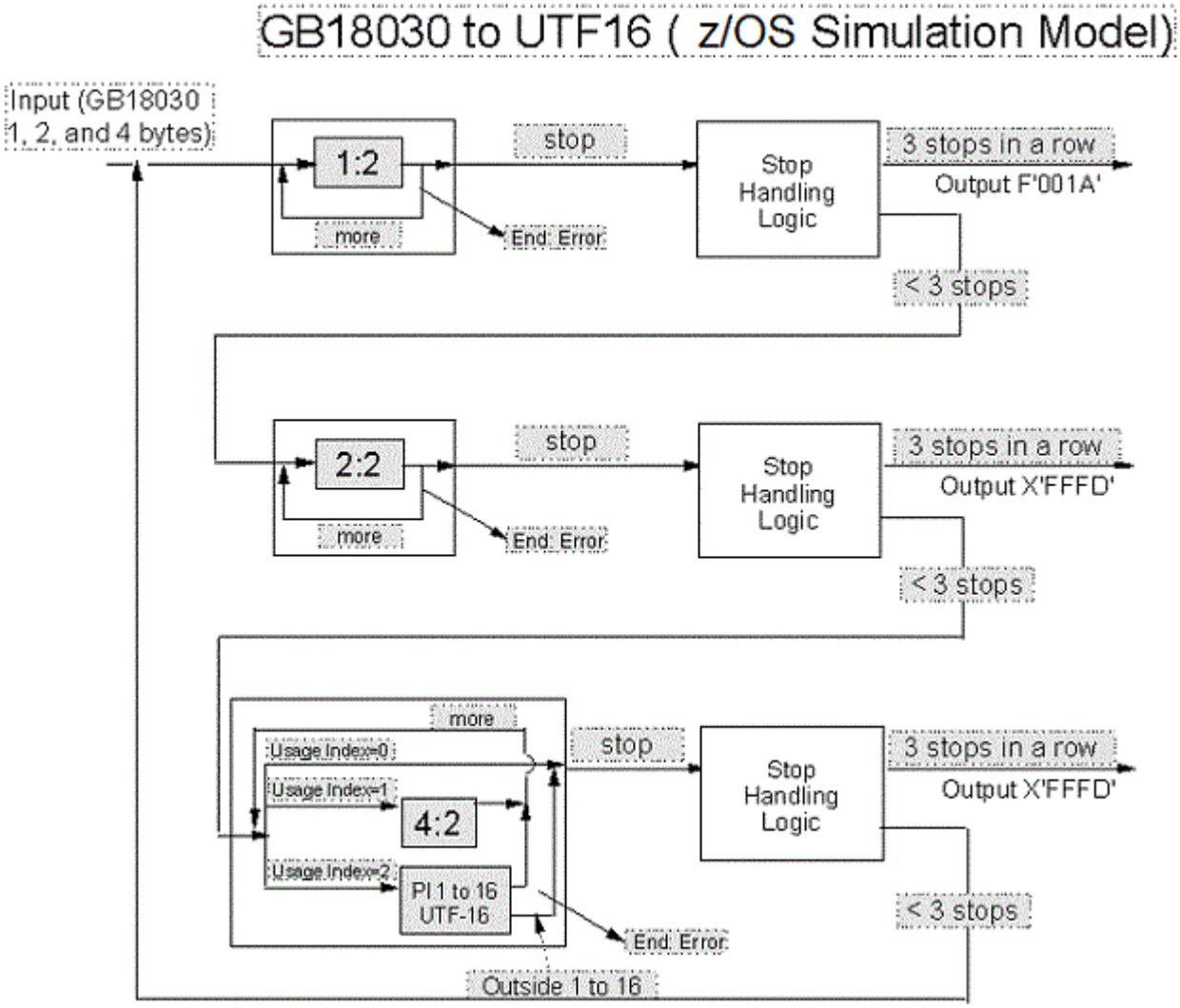


Figure 1.13 - GB18030 to UTF-16 Conversion Method

In understanding the GB18030 to UTF-16 z/OS conversion method, we refer to figure 1.13 above in order to obtain a graphical representation of the following description. Upon first glance at figure 1.13, note that there exists 3 binary tables for dealing with 1:2 byte, 2:2 byte and 4:2 byte mappings respectively. The presence of input that has no existing mapping within any of the three tables can be detected and

substituted upon passing input through the three stages. The precise means by which this is accomplished is described later in detail.

In this model, first the GB18030 input stream will be fed into the conversion logic. The conversion logic initially starts with the 1:2 stage of operation. In the step, the GB18030 input stream will be run through the 1:2 binary table by fetching one byte at a time from the input stream until it hits a stop character in the binary table or End/Error condition. In the case of an End/Error condition, execution will be terminated.

In the case that a stop character is encountered (X'FFFF' in this case) the stop handling logic will be executed. Then the execution goes to the next step in the process, which is 2:2 stage of operation.

In the 2:2 stage of operation, the remaining GB18030 input stream will be passed through the 2:2 binary table by fetching two bytes at a time from the input stream until it hits a stop character or End/Error condition. In the case of an End/Error condition, execution will be terminated.

In the case that a stop character is encountered (X'FFFF' in this case) the stop handling logic will be executed. Then the execution goes to the next step in the process, which is 4:2 stage of operation in this case.

In the 4:2 stage of operation, the remaining GB18030 input stream will be passed through the 4:2 conversion logic by fetching four bytes at a time from the input stream. Before attempting to find a target code point for an input 4 byte code point (b0 b1 b2 b3), the b0 value is checked to confirm that it has been used. This is checked by using the b0_used_array (see 4-byte to 2-byte conversion). If the entry corresponding to the b0 in b0_used_array is a 00, then the code point(b0 b1 b2 b3) is unassigned and its mapping is not defined in the conversion logic. This will be treated as hitting a stop character in the logic. If the entry corresponding to the b0 in b0_used_array is a 01, then the 4:2 binary table look up will take place by fetching 4 bytes at a time from the input stream. If the entry corresponding to the b0 in b0_used_array is a 02, then the GB18030 to UTF-16 transformation logic will be executed to output valid surrogate high and low pair. These three types of operations will take place until one of the three operations produce a stop character or End/Error condition. In the case of an End/Error condition, the execution will terminate.

In the case that a stop character is encountered (X'FFFF' in this case) the stop handling logic will be executed. Then the execution loops back to the starting step of the process, which is 1:2 stage of operation in this case. This loop of operations will go on until the end of input stream is reached.

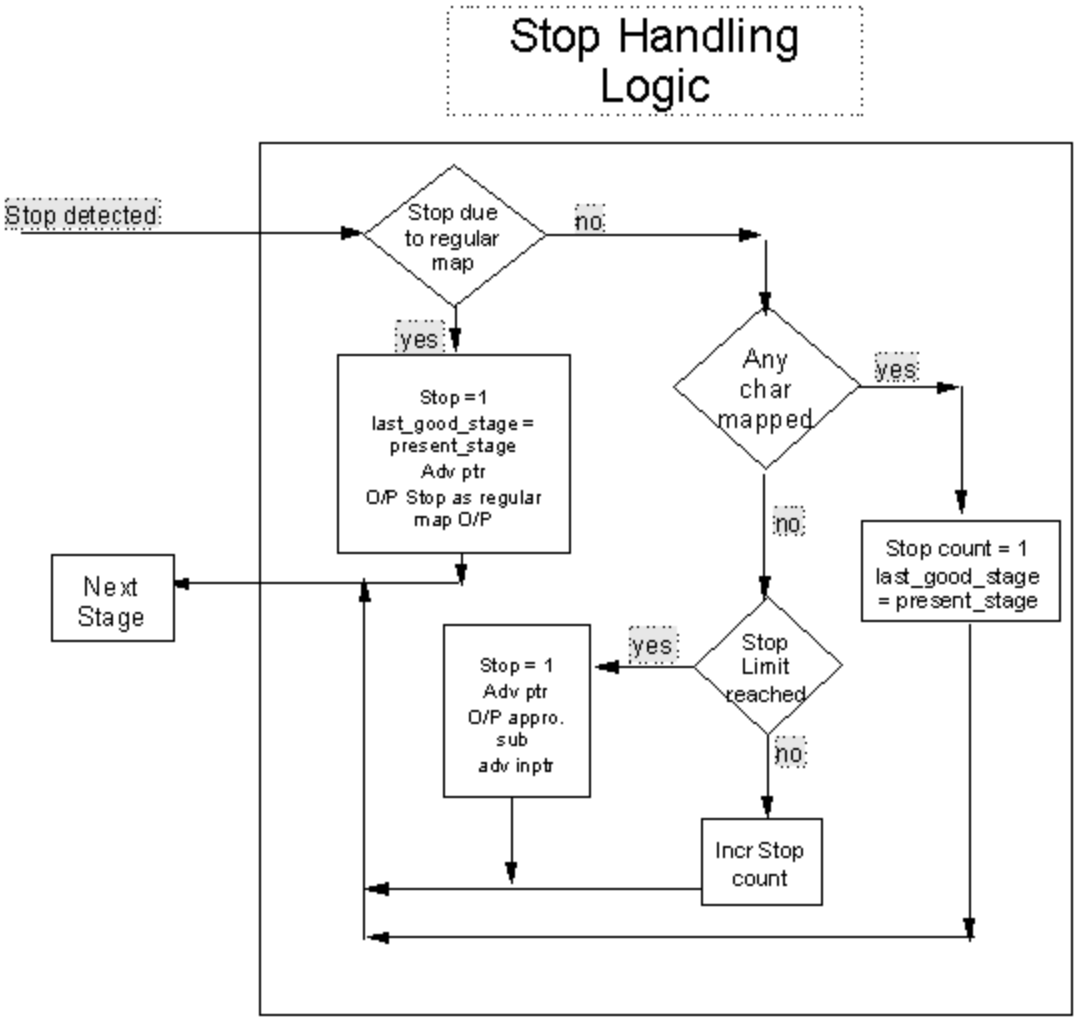


Figure 1.3 - **UTF-16 to GB18030 Conversion Methods**

The stop handling logic is illustrated in Figure 1.3 above (duplicate of Figure 1.3 presented here again for the reader's convenience). It is an integral part to the algorithm's ability to determine whether or not SUBs are to be output and to detect the presence of regular mappings. This task is accomplished with the assistance of a 'stop limit' counter that effectively keeps track of the number of successive stops in the algorithm and 'last_good_stage' flag. It also remembers the last successful stage of conversion, and takes appropriate action.

In the stop handling logic, the logic first checks whether this stop is due to a regular mapping or not. If it is due to a regular mapping, the stop count will be reset to 1 and the last successful stage of operation is also set to the present stage of operation. The input pointer will be moved to the next position in the input stream while the stop (X'FFFF') character goes to the output as a target code point.

An important point to be aware of in this logic is that stop handling logic is stage dependent. For example, GB18030 code point X'8437A439' maps to UTF-16 code point X'FFFF' which is same as the stop code point in this case. So the stop handling logic after 4:2 logic must check whether this stop is due to a regular mapping (for input X'8431A439') or not. This regular mapping could differ from stage to stage. This case has to be taken care of in the stop handling logic for each stage. (In the

OS390GB2U.C sample program this case has been taken care of in the 4:2 logic which is slightly different than what is described in here).

If the stop is not due to the regular mapping, then the stop handling logic checks whether any successful conversion has taken place in this stage of operation. This can be done by checking whether the input pointer has moved or not. If there is any successful conversion then the stop count will be reset to 1 and the last successful stage of operation is also set to the current stage of operation before we move on to the next stage of operation. If there is no mapping in this stage of operation then the logic checks whether the stop limit has been reached or not. If the stop limit is reached this indicates that each stage of the conversion operation (1:2, 2:2 and 4:2) has been executed without finding a successful mapping for the input code point. In this case a sub is output as target according to the last successful stage of conversion. The stop count will be reset to 1 now and input pointer also advanced to the next position.

If the stop limit is not reached, then the stop count will be incremented and the execution goes to the next stage of operation.

1-byte to 2-byte Conversion

The conversion method used for converting GB 1-byte to UTF-16 2-byte is identical to the previously mentioned method for converting GB 1-byte to UCS-2 2-byte. Please refer to 1-byte to 2-byte Conversion for GB to UCS-2 for details.

2-byte to 2-byte Conversion

The conversion method used for converting GB 2-byte to UTF-16 2-byte is identical to the previously mentioned method for converting GB 2-byte to UCS-2 2-byte. Please refer to 2-byte to 2-byte Conversion for GB to UCS-2 for details.

4-byte to 2-byte Conversion

The conversion method used for converting GB 4-byte to UCS-2 2-byte has previously been discussed in a prior section. Please refer to 4-byte to 2-byte Conversion for GB to UCS-2 for details.

Transformations

Transformation between GB18030 and UTF-32

In UTF-32 the first byte is always 00, and the second byte represents the plane number. Planes 0 through 16 are defined in UCS. GB18030 1-byte (all), 2-byte (all) and some of the 4-byte (X'81308130' to X'8431A439') code points fill all of the possible mapping points in UCS Plane 0. All of the UCS code points other than those found in BMP/Plane 0 (that is all the code points from Plane 1-16) are mapped to GB18030 4-byte code points only. No GB18030 1-byte or 2-byte code points are mapped to the Plane 1-16 code points.

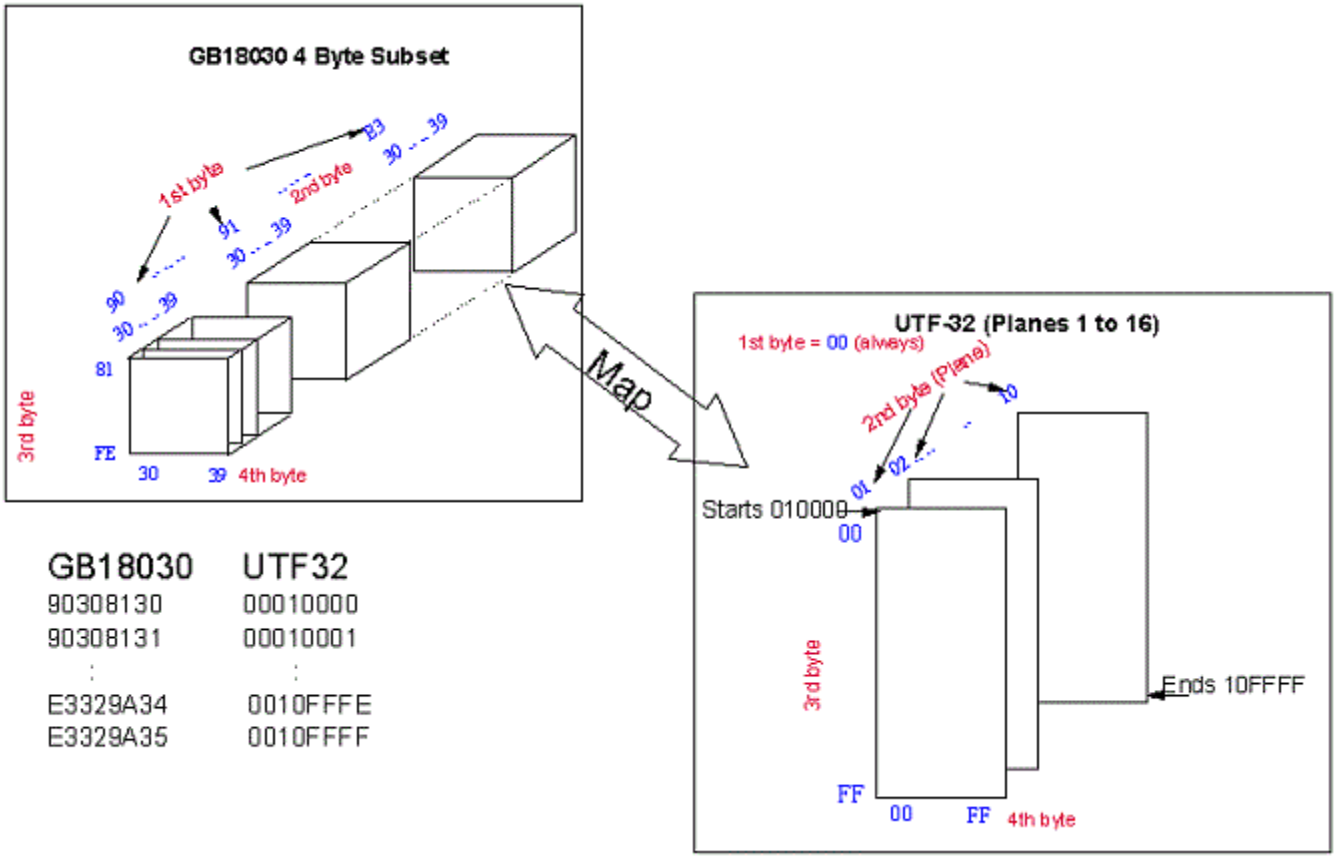


Figure 1.14: GB 4-byte to UTF-32 4-byte Mapping Structures

The mapping between GB18030 code points (4-byte subset) and the UCS code points from Plane 1-16 are done algorithmically. Starting from X'90308130' up to X'E3329A35', all the valid GB18030 4-byte code points are linearly mapped into the UCS code points in Planes 1-16 starting from X'00010000' up to X'0010FFFF' (Please refer to Figure 1.14).

GB18030	**UTF-32**
90308130	00010000
90308131	00010001
:	:
E3329A34	0010FFFE
E3329A35	0010FFFF

So given the valid GB18030 4-byte code point X'90308130', the corresponding UTF-32 value is calculated as follows:

Let 4 bytes of the GB18030 code point be b0 b1 b2 b3.

u0 = (b0 - X'90') * number of possible b1 * number of possible b2 * number of possible b3

u1 = (b1 - X'30') * number of possible b2 * number of possible b3

u2 = (b2 - X'81') * number of possible b3

u3 = (b3 - X'30')

UTF-32 code point = u0 + u1 + u2 + u3 + X'00010000'

Using the same principles for a UTF-32 code point in Plane 1 to Plane 16, the corresponding GB18030 value can be calculated as follows:

Let b0 b1 b2 b3 be the 4 bytes of GB18030 code point.

Let UTF-32 code point - X'00010000' = u0 u1 u2 u3

b0 = u0/(number of possible b1 * number of possible b2 * number of possible b3) + X'90'

b1 = u1/ (number of possible b2 * number of possible b3) % number of possible b1+ X'30'

b2 = u2/ (number of possible b3) % number of possible b2+ X'81'

b3 = u3 % number of possible b3 + X'30'

4-byte GB18030 = b0b1b2b3

(Note: * = multiplication; / = division; % = modular operation; - = subtraction; + = addition in the above equations)

Transformations between UTF-32 and UTF-16 Encoding Forms of Unicode

Each UTF-32 code point will be transformed into a UTF-16 surrogate pair as follows:

UTF-32	**UTF-16**
X'0000 0000'	X'0000'
:	:
X'0000 FFFF'	X'FFFF'
X'0001 0000'	X'D800 DC00' (see Figure 1.15 below)
:	:
X'0010 FFFF'	x'DBFF DFFF'
X'0011 0000' or greater	unmapped

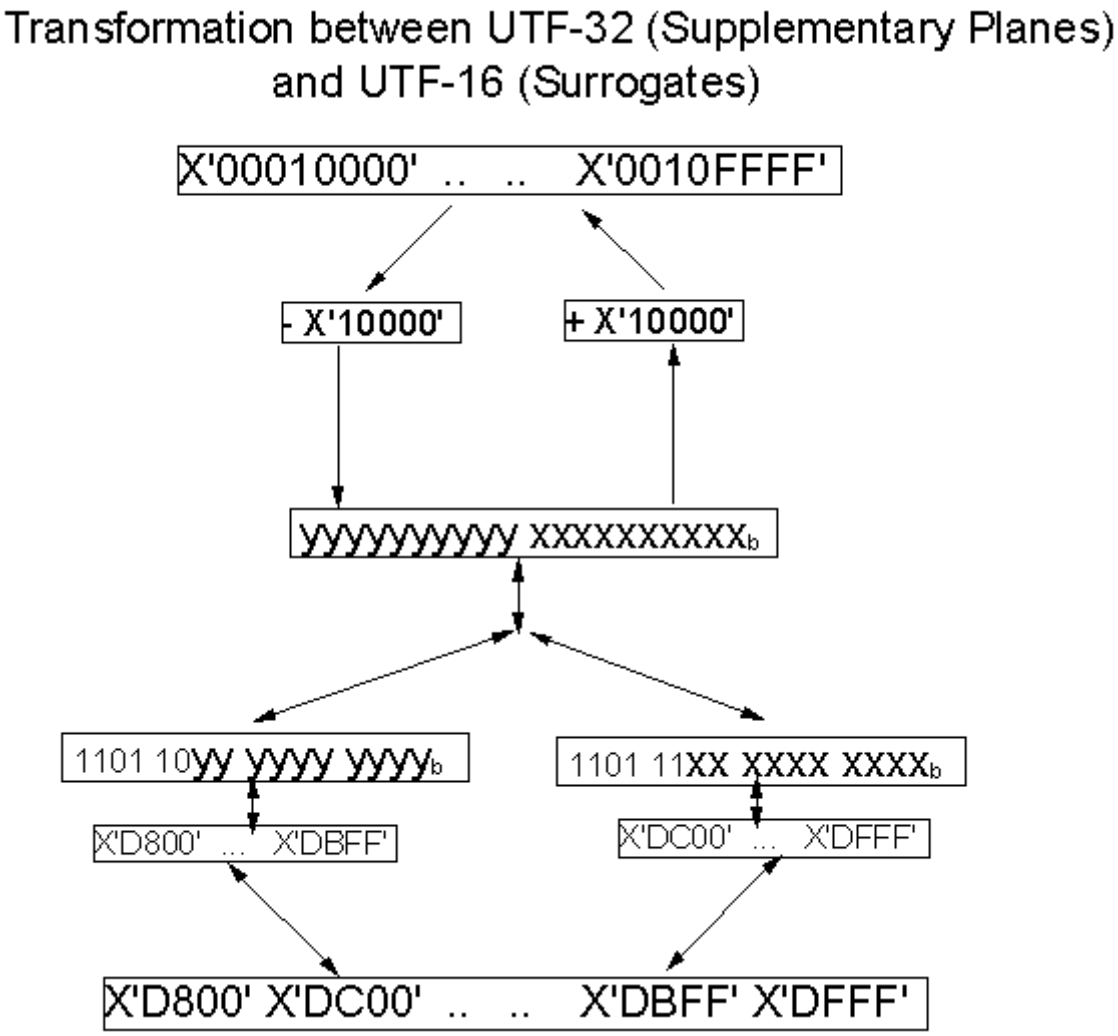


Figure 1.15 - UTF-32 (Supp. Planes) and UTF-16 (Surrogates)

Section 3: Conversions Between Unicode and Host Encodings

Conversion Between UCS-2 (CCSID 17584) <--> Host(1 and 2-byte, CCSID 1388)

This Section contains a description of the conversion tables and associated methods for converting data between S-Ch Host CCSID 1388 encodings and UCS-2 encodings.

Text Format Tables

Combined GB18030 Host Tables

056C44B0.TPMAP100 - S-Ch Host (CCSID 1388) To UCS-2 (CCSID 17584)

44B0056C.RPMAP100 - UCS-2 (CCSID 17584) To S-Ch Host (CCSID 1388)

056C44B0.UPMAP100 - S-Ch Host (CCSID 1388) To and FROM UCS-2 (CCSID 17584)

Component GB18030 Host Tables

334444B0.TPMAP100 - Text table from Host 1-byte part (4933) to UCS-2 (17584)

44B03344.RPMAP100 - Text table from UCS-2 (17584) to Host 1-byte part (13124)

334444B0.UPMAP102 - Text table between Host 1-byte part (13124) and UCS-2 (17584)

134544B0.TPMAP100 - Text table from Host 2-byte part (13124) to UCS-2 (17584)

44B01345.RPMAP100 - Text table from UCS-2 (17584) to Host 2-byte part (13124)

134544B0.UPMAP102 - Text table between Host 2-byte part (13124) and UCS-2 (17584)

Table files with the extension RPMAPnnn, TPMAPnnn, and UPMAPnnn contain human readable formats. They have two columns containing the source code point value (in Hex), and the target code point value (in Hex). Each file contains a brief header and column descriptions. Please ensure you read the header files for values of SUB used and any special handling required.

Conversion From UCS-2 (CCSID 17584) to S-Ch Host Extended (CCSID 1388)

Combined S-Ch Host Conversion

To be able to indicate whether a double-byte UTF-16 code point is mapped to a single-byte or double-byte code point in Host, in a binary conversion table, some indication will be needed as to the width of the target code point. This is accomplished by using a normalized Host code point in the conversion tables. Each output code point entry will have two bytes, single-byte Host code point values will be normalized by inserting a leading 00 byte. This permits a fixed width 2-byte to fixed width 2-byte conversion method and associated table structure to be used. The conversion logic strips out the leading zero-byte and adds any necessary code extension controls (Shift-IN/Shift-Out) before placing the value in the output stream. The following sections give more details.

2-byte To 2-byte Conversion

File: 44B0056C.UM-E-D

This binary table maps UCS-2 to normalized S-Ch Host Extended.

Method 8:

The binary conversion table is same to the tables used in existing CDRA Method 8. The input data stream consists of UCS-2 2-byte code points. The output from the conversion table will be 2-byte normalized Host code points. The resultant code points are de-normalized before being inserted into the output data stream.

Assumed normalization:

Host Byte	*Normalized*	*Comment*
xx (Single byte)	00xx	Two bytes, with 1 leading zero byte; SO/SI removed.
xxxx (Double byte)	xxxx	Two bytes

To describe the conversion method we first define the concept of a "ward". A ward is a section of a double-byte code page. It is equivalent to a "row" of code points in ISO/IEC 10646 and in Unicode. All of the code points contained in a specific ward begin with the same first byte. A ward is populated if there is at least one character in the double-byte code page (UCS-2 in this case) whose first byte is the ward value. There are 256 wards numbered X'00' to X'FF'.

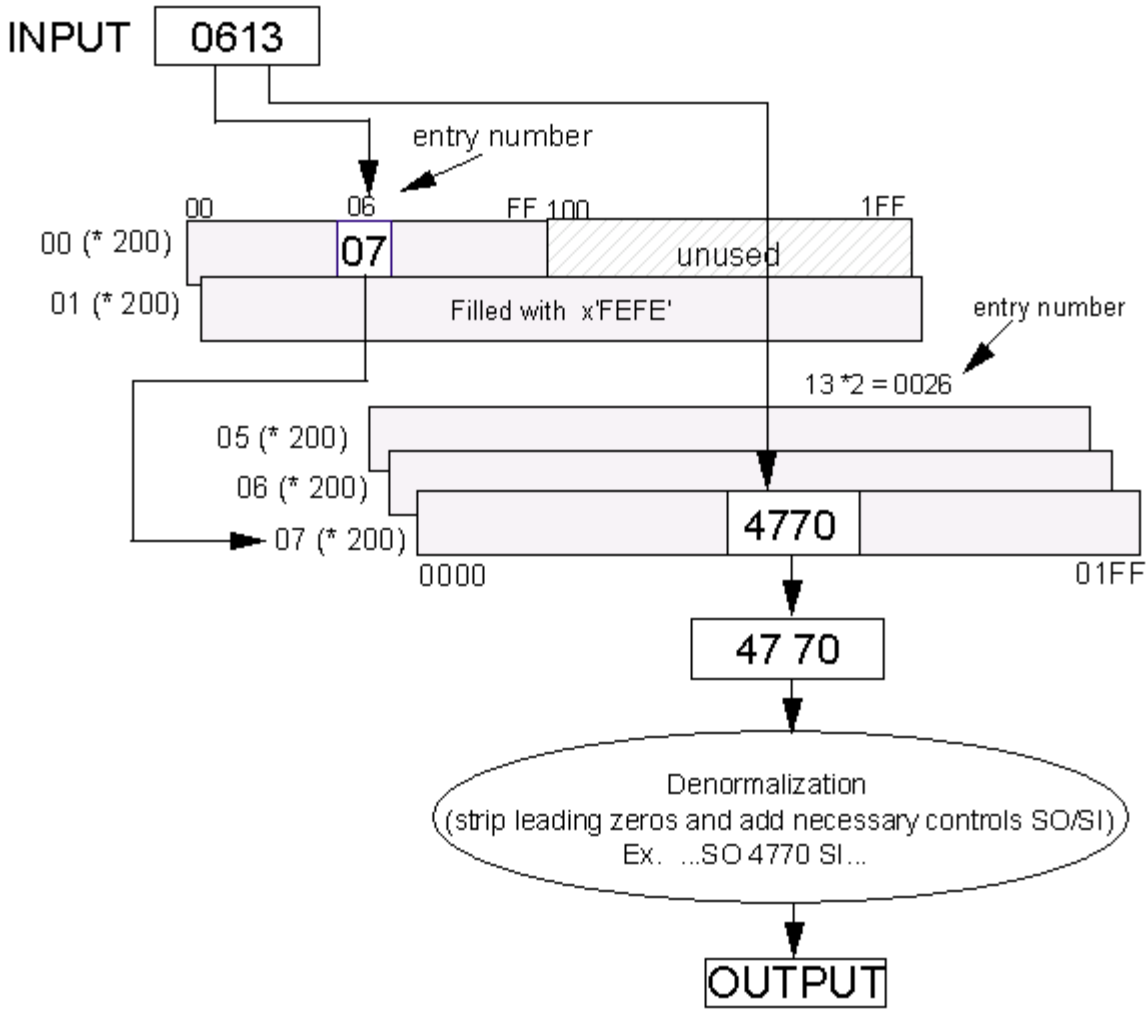


Figure 2.1 - **UCS-2 to S-Ch Host Extended - Binary Table and Conversion Method**

The double-byte binary conversion table is made up of several 512-byte vectors. The first vector contains 256 single-byte indices (vector numbers) into the rest of the table, followed by 256 unused bytes. The second vector, the "substitute" vector is used for mapping code points in unassigned wards, and is filled with 256 2-byte SUB code points (X'FEFE' in this case). These are followed by a collection of 512-byte vectors, one for each populated ward in the source code page.

The table is used as follows. The first byte of the input code point is used as a pointer into the index vector. The single-byte value found at the corresponding position in the index is the vector number in which to perform the second lookup. The second byte of the input code point is used as a pointer into the specified vector.

For example in Figure 2.1 the input code point is X'0613'. Using x’06’ as a pointer into the index vector, we find x’07’ as the specified vector in the table structure. Then the second byte, x’13’ is used to calculate the pointer value into the specified vector. The resultant double-byte output code point is found in vector number x'07' beginning at position X'0026' (which is two

times X'13'), counting into the vector starting at zero. In this example the output double-byte value is x'4770'.

The second vector mentioned for handling unassigned wards works as follows. All of the 256 double-byte code point values found in this vector (vector X'01') are those of the "Substitute (SUB)" character of the target code page (X'FEFE' for S-Ch Host Extended). All the entries in the index vector for unassigned wards point to this "substitute" vector.

The binary conversion table is the same format as the CDRA UM-E-D binary tables.

Denormalization:

Since resultant single-byte values in the table have been normalized with a leading zero-byte, when composing the output string the leading zero-bytes must be removed. In a properly formed Host Mixed data stream the single-byte strings and the double-byte strings must be bracketed with appropriate controls (SO/SI). The denormalization process must perform both tasks.

Example: SO 41 41 40 40 SI 41 SO 72 01 SI

Component S-Ch Host Conversions

Introduction

The z/OS converter does not use the combined tables. In the z/OS environment a call is made to CONVERT DATA tagged with CCSID 01388 to Unicode CCSID. The converter logic looks up 01388 information and gets the component conversion tables and sets up 2 to 1, 2 to 2 logic and resources and starts executing the conversion. The z/OS implementation needs the component tables not the combined ones. In this case two binary tables are required to perform the conversion. They are the following:

44B03344.US-C0-A1 - Binary table from UCS (17584) to Host 1-byte part (13124)
(CDRA 2-bytes to 1-byte conversion method, with X'3F' as SUB/STOP)

44B01345.UM-C0-A1 - Binary table from UCS (17584) to Host 2-byte part (4933)
(CDRA 2-byte to 2-byte conversion method, with X'FEFE' as SUB/STOP)

The high level logic is:

The UCS-2 input stream (sequences of double-bytes) will be fed into the conversion logic. Within the conversion logic, the UCS-2 input stream will be passed through the following logic:

- 2:1 logic use UCS-2 to S-Ch Host single-byte binary table to get S-Ch Host single-byte output (see the section on 2-byte to 1-byte Conversion for more details)

- 2:2 logic use UCS-2 to S-Ch Host double-byte binary table to get S-Ch Host double-byte output (see the section on 2-byte to 2-byte Conversion for more details)

The UCS-2 to single-byte S-Ch Host binary table consists of several 256-byte vectors. The first vector (00) acts as an index into the rest of the table and

the second vector (01) is the "substitute" vector (see the section on 2-byte to 1-byte Conversion for more details).

The UCS-2 to double-byte S-Ch Host binary table is made up of several 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see the section on 2-byte to 2-byte Conversion for more details).

UCS-2 to S-Ch Host conversion Logic

To understand the UCS-2 to S-CH Host z/OS conversion method, refer to figure 2.2 for a graphical representation of the following description. In figure 2.2, note that there exists 2 binary tables for dealing with 2:1 byte and 2:2 byte mappings respectively. The presence of input that has no existing mapping within either of the two tables can be detected and substituted upon passing input through the two stages. The precise means by which this is accomplished is described later in detail.

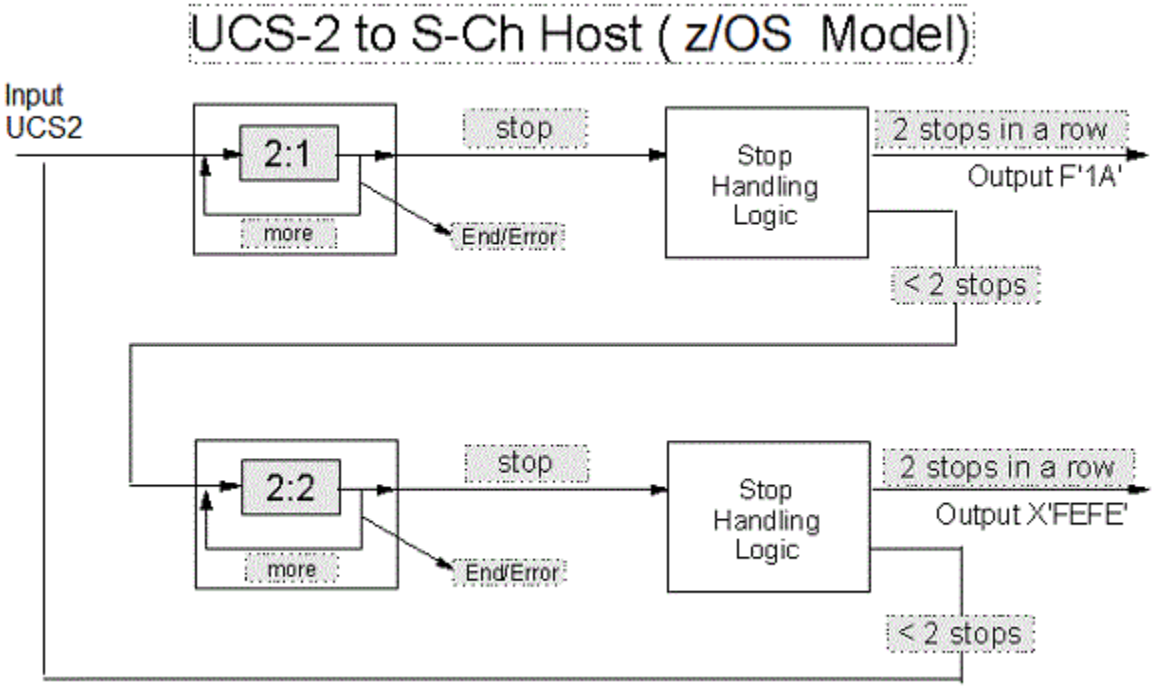


Figure 2.2 - UCS-2 to S-Ch Host Conversion Method

In this model, the UCS-2 input stream will first be fed into the conversion logic. The conversion logic initially starts with the 2:1 stage of operation. In the 2:1 stage of operation, the UCS-2 input stream will be run through the 2:1 binary table by fetching two bytes at a time from the input stream until it hits a stop character in the binary table or End/Error condition. In case of End/Error condition, execution will be terminated.

In the case that a STOP/SUB character is encountered (X'3F' in this case) the stop handling logic will be executed. Then the execution goes to the next stage of operation, which is 2:2 stage of operation in this case.

In the 2:2 stage of operation, the UTF-16 input stream will be run through the 2:2 binary table by fetching two bytes at a time from the input stream until it hits a stop character or End/Error condition. In case of End/Error condition, execution will be terminated.

In the case that a STOP/SUB character is encountered (X'FEFE' in this case) the stop handling logic will be executed. Then the execution goes to the next stage of operation, which is 2:1 stage of operation in this case.

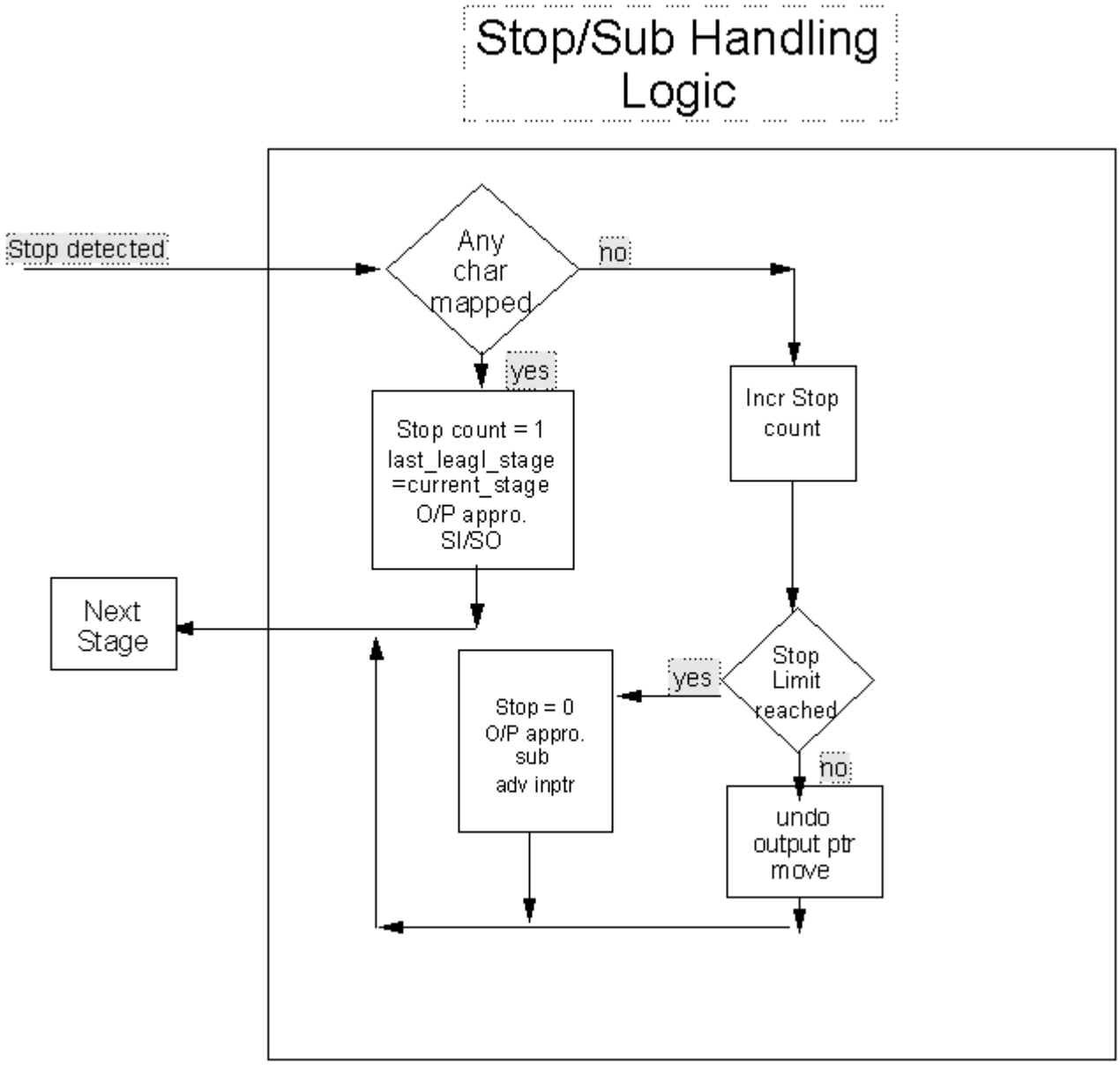


Figure 2.3 - **UCS-2 to S-Ch Host Conversion Methods**

The stop handling logic is illustrated in Figure 2.3 above. It is an integral part of the algorithm's ability to determine whether or not a SUB or an appropriate control character (SI/SO) is to be output when a STOP/SUB character is detected. This task is accomplished with the assistance of a 'stop limit' counter that effectively keeps track of the number of successive stops in the

algorithm and 'last_good_stage' flag which remembers the last successful stage of conversion.

In the stop handling logic, the logic first checks whether any successful conversion has taken place in this stage of operation. This can be done by checking whether the input pointer has moved or not. If there is any successful conversion, then the stop count will be reset to 1. The last successful stage of operation is also set to the current stage of operation and the appropriate control character (SI/SO) is placed in the output stream before we move on to the next stage of operation. If there is no mapping in this stage of operation then the stop count will be incremented and the logic checks whether the stop limit has been reached. If the stop limit has been reached then a sub is output as target according to the last successful stage of conversion. The stop count will be reset to 0 now and input pointer also advanced to the next position. If the stop limit is not reached, then and the execution goes to the next stage of operation after removing any inappropriate control characters (SI/SO).

2-byte to 1-byte Conversion

The conversion method used for converting UCS-2 to S-Ch single-byte is identical to the previously mentioned method for converting UTF-16 to GB18030 single-byte. Please refer back to the earlier section on 2-byte to 1-byte Conversion for details.

2-byte to 2-byte Conversion

The conversion method used for converting UCS-2 to S-Ch double-byte is identical to the previously mentioned method for converting UTF-16 to GB18030 double-byte. Please refer back to the earlier section on 2-byte to 2-byte Conversion for details.

Conversion From S-Ch Host Extended (CCSID 1388) to UCS-2 (CCSID 17584)

Combined S-Ch Host

This table is used to find the target code point when the source code point is single/double-byte. The method used to create this table requires that the input data is normalized such that each input code point is two bytes long. This is done by prefixing each single-byte code point with a zero-byte (X'00). Any code extension controls *(Shift-IN/Shift-Out)* are removed from the input data stream.

2-byte to 2-byte Conversion

File: 056C44B0.MU-R-D

Table format: The binary conversion table created by using the existing CDRA Method 7 - a two-step vector lookup method.

Method 7:

The conversion table and the associated method are illustrated in Figure 2.6 below.

The double-byte binary conversion table is made up of several 512-byte vectors. The first vector contains 256 single-byte indices (vector numbers) into the rest of the table, followed by 256 unused bytes. The second vector, the "substitute" vector is used for mapping code points in unassigned wards, and is filled with 256 2-byte SUB code points (X'FFFD' in this case). These are followed by 512-byte vectors, one for each populated ward in the source code page.

The table is used as follows. The first byte of the input code point is used as a pointer into the index vector. The single-byte value found at the corresponding position in the index is the vector number in which to perform the second lookup. The second byte of the input code point is used as a pointer into the vector specified by the index vector.

For example in Figure 2.6 the input code point is X'4760'. The first byte of the code point (x'47') is used as a pointer into the index vector, resulting in x'0A' being identified as the vector containing the output code point. The resultant double-byte output code point is found in the specified vector number (x'0A') by calculating the location from the second byte of the input code point. The second byte x'60' is multiplied by 2 (since each resultant code point is two bytes long). Thus the output code point will be found beginning at position X'00C0' (which is two time X'60'), in the specified vector. Following this process results in an output code point of x'0603'.

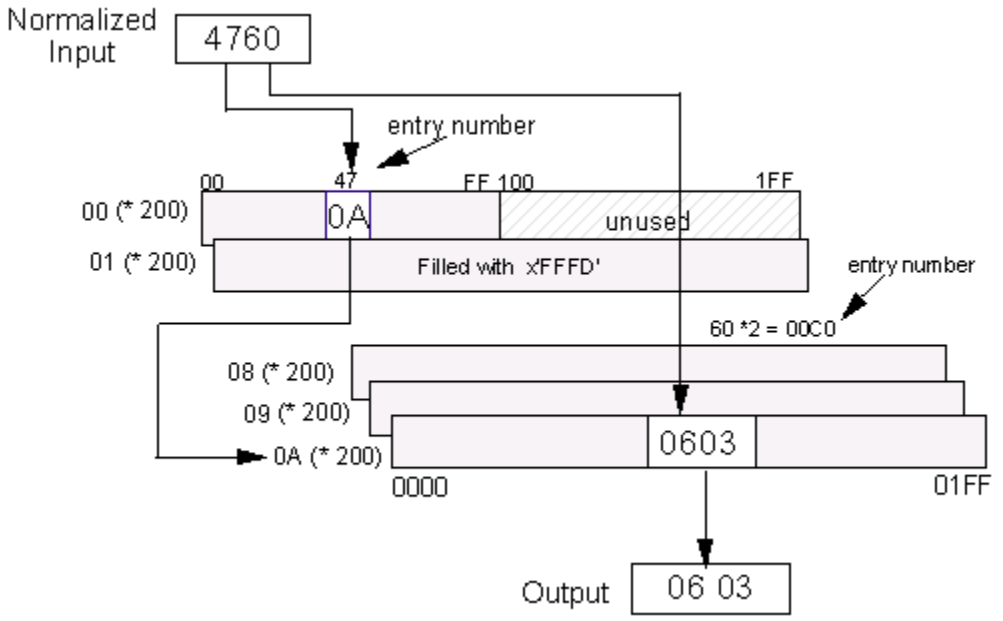


Figure 2.6: Normalized S-Ch Host Extended 2-byte to UCS-2 2-byte Table and Method

The second vector mentioned for handling unassigned wards works as follows. All of the 256 double-byte code point values found in this vector (vector X'01') are those of the "Substitute (SUB)") character of the target code page (X'FFFD' for UCS-2). All the entries in the index vector for unassigned wards point to this "substitute" vector.

The binary conversion table is the same format as CDRA MU-R-D binary tables.

Components S-Ch Host (z/OS Usage)

Introduction

S-Ch contains 1, and 2-byte code points and UCS-2 contains 2-byte code points. According to the z/OS logic, in order to do conversion from S-Ch to UCS-2, there are two binary tables needed. They are the following:

334444B0.MU-C0-A1 - Binary table from S-Ch Host 1-byte part (13124) to UCS-2 (17584)
(CDRA 1-byte to 2-byte conversion method, with X'FFFD' as STOP/SUB character)

134544B0.MU-C0-A1 - Binary table from S-Ch Host 2-byte part (4933) to UCS-2 (17584)
(CDRA 2-byte to 2-byte conversion method, with X'FFFD' as STOP/SUB character)

The high level logic is:

The S-Ch Host input stream (sequence of 1 and 2-bytes) will be fed into the conversion logic. Within the conversion logic, the S-Ch Host input stream will be passed through the following logic:

- 1:2 logic use S-Ch Host single-byte to UCS-2 binary table to get UCS-2 double-byte output

- 2:2 logic use S-Ch Host double-byte to UCS-2 binary table to get UCS-2 double-byte output

The single-byte to UCS-2 array consists of a single 512-byte vector (256 2-byte entries).

The double-byte GB to UCS-2 array is made up of several 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see section 1.3d for more details).

S-Ch to UCS-2 Conversion Logic

To understand the S-CH to UCS-2 Host z/OS conversion method, refer to figure 2.7 for a graphical representation of the conversion. Looking at figure 2.7, note that there exist 2 binary tables for dealing with 1:2 byte and 2:2 byte mappings respectively. The presence of input that has no existing mapping within any of the two tables can be detected and substituted upon passing input through the two stages. The precise means by which this is accomplished is described later in detail.

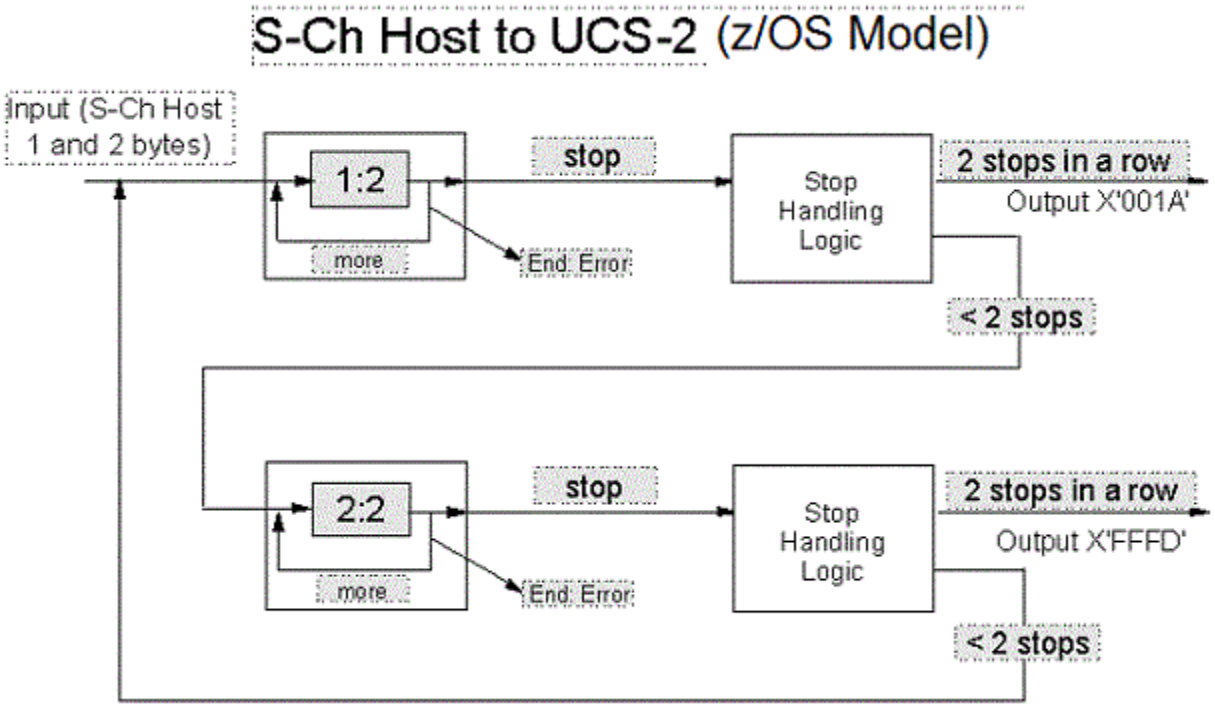


Figure 2.7 - S-Ch to UCS-2 Host Conversion Method

In this model, the S-Ch input stream will first be fed into the conversion logic. The conversion logic initially starts with the 1:2 stage of operation. In the 1:2 stage of operation, the S-Ch input stream will be run through the 1:2 binary table by fetching 1 byte at a time from the input stream until it hits a stop character in the binary table or End/Error condition. In case of End/Error condition, execution will be terminated.

In the case that a stop character is encountered (X'000E' in this case) the stop handling logic will be executed. Then the execution goes to the next stage of operation, which is 2:2 stage of operation in this case.

In the 2:2 stage of operation, the UTF-16 input stream will be run through the 2:2 binary table by fetching two bytes at a time from the input stream until it hits a stop character or End/Error condition. In case of End/Error condition, execution will be terminated.

In the case that a stop character is encountered (X'FFFD' in this case) the stop handling logic will be executed. Then the execution goes to the next stage of operation, which is 2:1 stage of operation in this case.

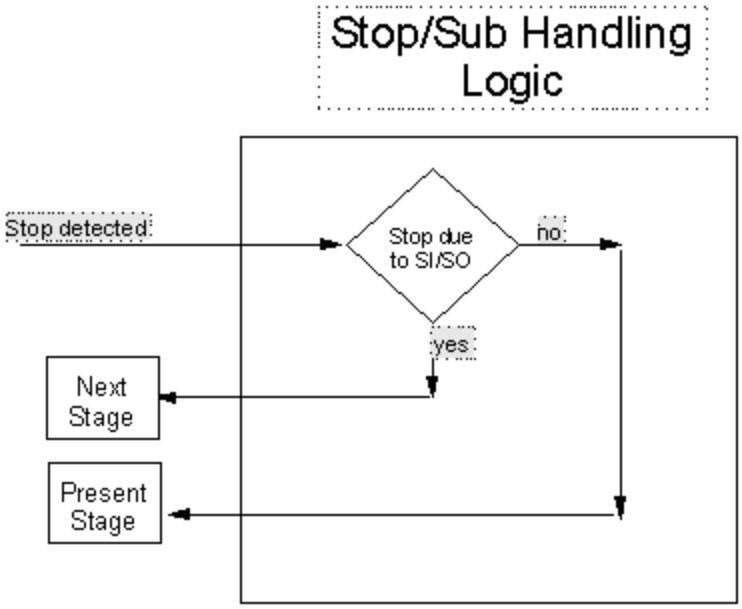


Figure 2.8 - S-Ch to UCS-2 Host Conversion Method

The stop handling logic is illustrated in Figure 2.8 above. Unlike the previous stop logic implementation, in performing UCS-2 to S-Ch conversions, the C-Sh to UCS-2 stop logic is much simpler. Its sole objective is to determine whether or not the stop was due to a control character (SI/SO). If it was due to a control character (SI/SO), then execution continues to the appropriate stage.

1-byte to 2-byte Conversion

The conversion method used for converting S-Ch Host 1-byte to UCS-2 2-byte is identical to the previously mentioned method for converting GB 1-byte to UCS-2 2-byte.

2-byte to 2-byte Conversion

The conversion method used for converting S-Ch Host 2-byte to UCS-2 2-byte is identical to the previously mentioned method for converting GB 2-byte to UCS-2 2-byte.

Section 4: Conversions Between GB18030 and Host Encodings

S-Ch Host Extended (CCSID 1388) <-> GB18030 (1,2 and 4-byte, CCSID 5488)

Introduction

This document contains description of conversion tables and associated methods for converting data between S-Ch Host extended CCSID 1388 encodings and GB 18030 (CCSID 5488) encodings. These (GB 18030 to and from Host) conversion tables use a normalized form of data. The Host code points are normalized by placing a leading zero-byte (X'00) in front of each single-byte to yield a two-byte form. Host data must have the SO-SI control characters deleted during normalization and reinserted afterwards during de-normalization. The GB code points are normalized to four-bytes by inserting leading zero-bytes for the single- and double-byte GB code point values. Figure 3.1 shows the use of the Host to and from GB18030 conversion tables.

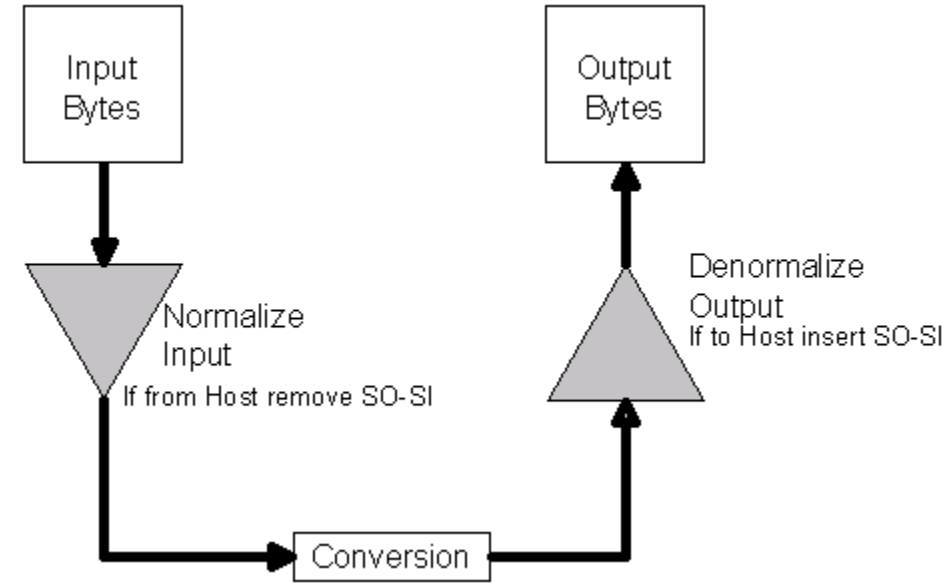


Figure 3.1 - Host and GB 18030 Conversion

The input byte or bytes, up to maximum of four bytes per code point, are first normalized and used as input to the conversion table. The output (again, four bytes per code point maximum) from the conversion table is also normalized data, which must be de-normalized prior to subsequent processing. Host data must have the SO-SI control characters deleted during normalization and reinserted afterwards during de-normalization.

Source mapping file between GB18030 and Host

File: HOST2GB.ALL (2000-12-07)

File: GB2HOST.ALL (2000-12-07)

Conversion From S-Ch Host (CCSID 1388) to GB18030 (CCSID 5488)

Combined GB18030 Tables

The GB code points are normalized to four-bytes by inserting leading zero-bytes for the single and double-byte GB code point values. The Host code points are normalized to two-bytes by inserting a leading zero-byte for the single-byte Host code point values.

2-byte to 4-byte Conversion

File 056C1570.MGN-R-D

This binary table contains the mapping from normalized Host to normalized GB18030.

Method 2x:

The binary conversion table is similar to the tables used in existing CDRA Method 2, but extended to handle the normalized GB18030-1 4-byte code. The input data stream consists of normalized Host 2-byte code points. The output from the conversion table will be 4-byte normalized GB code points which are de-normalized before being inserted into the output data stream.

Assumed normalization for GB18030-1:

GB Byte	*Normalized*	*Comment*
xx (Single byte)	000000xx	Four byte, with 3 zero byte leading zeros
xxxx (Double byte)	0000xxxx	Four byte, with 2 zero byte leading zeros
xxxxxxxx (Four byte)	xxxxxxxx	Four byte

To describe the conversion method we first define the concept of a "ward". A ward is a section of a double-byte code page. It is equivalent to a "row" of code points in ISO/IEC 10646 and in Unicode. All of the code points contained in a specific ward begin with the same first byte. A ward is populated if there is at least one character in the double-byte code page (normalized Host in this case) whose first byte is the ward value. There are 256 wards numbered X'00' to X'FF'.

This binary conversion table is made up of several 1024-byte vectors. The first vector acts as in index into the rest of the table. It contains 256 two-byte vector numbers (corresponding to each of the 256 wards) and the remaining 512 bytes are unused (filled with zeros). There is one 1024-byte vector for each populated ward in the source code page and one additional vector used for mapping all unassigned and invalid wards.

The method fetches two bytes at a time from the input data stream. The first byte is used as a pointer into the index vector -- as shown in Figure 3.2. Each vector number in the index vector is two bytes long. Therefore the first byte from the input code point is multiplied by two before calculating the offset into this index vector. The two-byte value found at the corresponding position in the index vector gives the vector number in which to perform the second lookup.

The second byte of the input code point is used as a pointer into the vector specified by the index vector. When calculating the offset into this vector

there are two things to remember -- first, the indexing starts at zero, and second, each entry is four bytes long (normalized GB18030 code point).

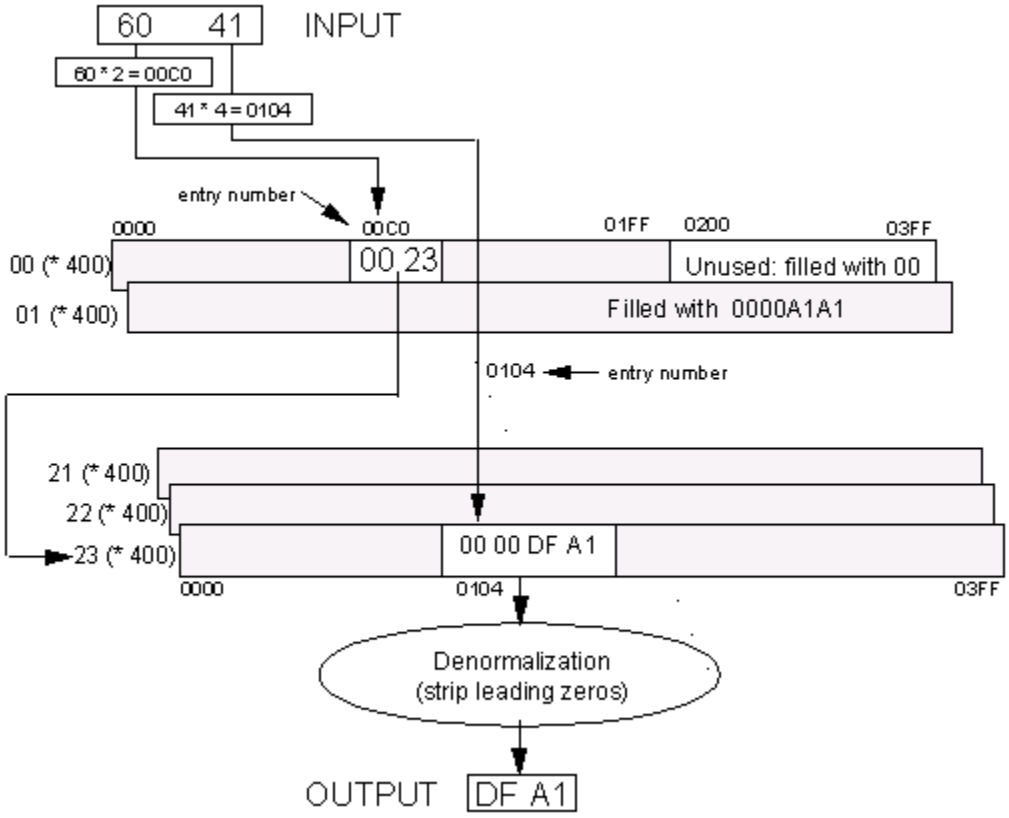


Figure 3.2 -S-Ch Host to GB18030-1 - Binary Table & Conversion Method

For example in Figure 3.2 the input code point is X'6041' you would find the two-byte vector number (X'0023') starting at byte position X'00C0' (which is two times X'60') in the index vector. The resultant four-byte output code point would be found in the specified vector number (X'0023') beginning at byte position X'0104' (which is four times X'41').

The one additional vector (X'0001') mentioned for handling code points from invalid or unassigned wards in the input data is used as follows. All of the 256 four-byte (normalized) code point values found in this vector are those of the "Substitute" (SUB) character of the target code page (X'0000A1A1' for GB18030-1). All of the entries in the index vector for unused and invalid wards point to this "substitute" vector.

In the binary table, target four-bytes could be found at the following positions.

Byte	Value	Position (hex)	Position (decimal)
first	0	X'0023' * X'400' + X'0104'	35 * 1024 + 260 = 36100
second	0	X'0023' * X'400' + X'0104' + 1	35 * 1024 + 260 + 1 = 36101

third	DF	X'0023' * X'400' + X'0104' + 2	35 * 1024 + 260 + 2 = 36102
fourth	A1	X'0023' * X'400' + X'0104' + 3	35 * 1024 + 260 + 3 = 36103

Denormalization:

Since the resultant single-bytes and double-bytes in the table have been prefixed with leading zero-bytes, when composing the output string the leading zero-bytes must be removed from the 4-byte output (three zero-bytes for single-byte and two zero bytes for double-byte).

Conversion From GB18030 (CCSID 5488) to S-Ch Host (CCSID 1388)

Combined GB18030 Tables

Following is a proposal for GB to HOST conversion, as an alternative to following the CDRA's EUC normalization method and the resulting table structure. The normalization steps for detecting when input is a single, double, or four-byte are followed. However, instead of normalizing the data as in the EUC case, the assumption here is to take three branches in the logic and come up with associated data structures. The data structures are linear arrays, one for each of the input types; single, double and four-byte. The indexing operations get into these arrays only for valid ranges of code points. All other code points and broken multi-byte sequences are trapped and are dealt with separately in the logic.

The high level logic is:

- Detect valid single, double or four-byte code points

- Use single-byte to HOST array for single-byte input

- Use double-byte to HOST array for double-byte input

- Use four-byte to HOST compact array for four-byte input

- All invalid and incomplete sequences dealt with separately

The main advantage of this method is to be able to keep the conversion tables as compact as possible (especially for the 4-byte to 2-byte part) and be able to get at the converted HOST code points in a relatively fast manner.

The single-byte to HOST array consists of a single 512-byte vector (256 Single-byte entries with a leading zero byte).

The double-byte GB to HOST array is made up of several 512-byte vectors. The first vector (00) acts as an index into the rest of the table and the second vector (01) is the "substitute" vector (see section 1.3d for more details).

The four-byte to HOST compact array is made up of:

- One vector, 1024 bytes long, first 256 bytes contain flag values, remaining 768 bytes are unused.

- Four vectors, 1024 bytes long, each containing 256 4-byte index values, and
- A long compact array containing all of the target two-byte HOST code points.

Detect a Valid Single-, Double-, Four-byte or Invalid code point

Refer to Figure 3.3 below. The method fetches one byte at a time from the input stream. The first byte, b0, is checked to see whether it is in the valid range of single-byte or start of a double-byte or four-byte code point. If it is not, then a SUB code point (X'003F' in this case) is inserted in the output data stream, and the pointer to the input stream is incremented by one.

If b0 is in the valid range of single-byte (X'00' to X'80'), then it is passed to the 1-byte to 2-byte conversion as a valid single-byte code point. The resultant two-byte output is placed in the output buffer, and the pointer into the input stream is incremented by one.

If b0 is in the range X'81' to X'FE' for a valid first byte of a double-byte or four-byte code point, then the next byte, b1, is checked to see whether it is in the valid range for a second byte of a double-byte (X'40' to X'7E' or X'80' to X'FE') or a second byte of a four-byte (X'30' to X'39') code point. If it is not, then the SUB code point X'003F' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b1 is within the valid range for a second byte of a double-byte code point (X'40' to X'7E' or X'80' to X'FE'), the sequence b0 b1 (or the input pointer) is passed to the 2-byte to 2-byte conversion. The resultant two-byte output is placed in the output buffer, and the pointer into the input stream is incremented by two.

If b1 is within the valid range for a second byte of a four-byte code point (X'30' to X'39'), then the next byte in the input stream, b2, is checked to determine whether it is in the valid range for a third byte of a four-byte code point (X'81' to X'FE'). If it is not, then the SUB code point X'003F' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b2 is within the valid range for a third byte of four-byte code point, the next input byte, b3, is checked to determine whether it is in the valid range for the fourth byte (X'30' to X'39'). If it is not, then the SUB code point X'003F' is inserted into the output stream and the pointer into the input stream is incremented by one. The pointer should point to byte b1 now. The first byte b0 is considered to be the start of a broken sequence of bytes in the input.

If b3 is within the valid range for the fourth byte then the sequence of bytes, b0 b1 b2 b3, (or the input pointer) is passed to the 4-byte to 2-byte conversion as valid four-byte code point.

The resultant two-byte output is placed in the output buffer, and the conversion pointer into the input stream is incremented by four.

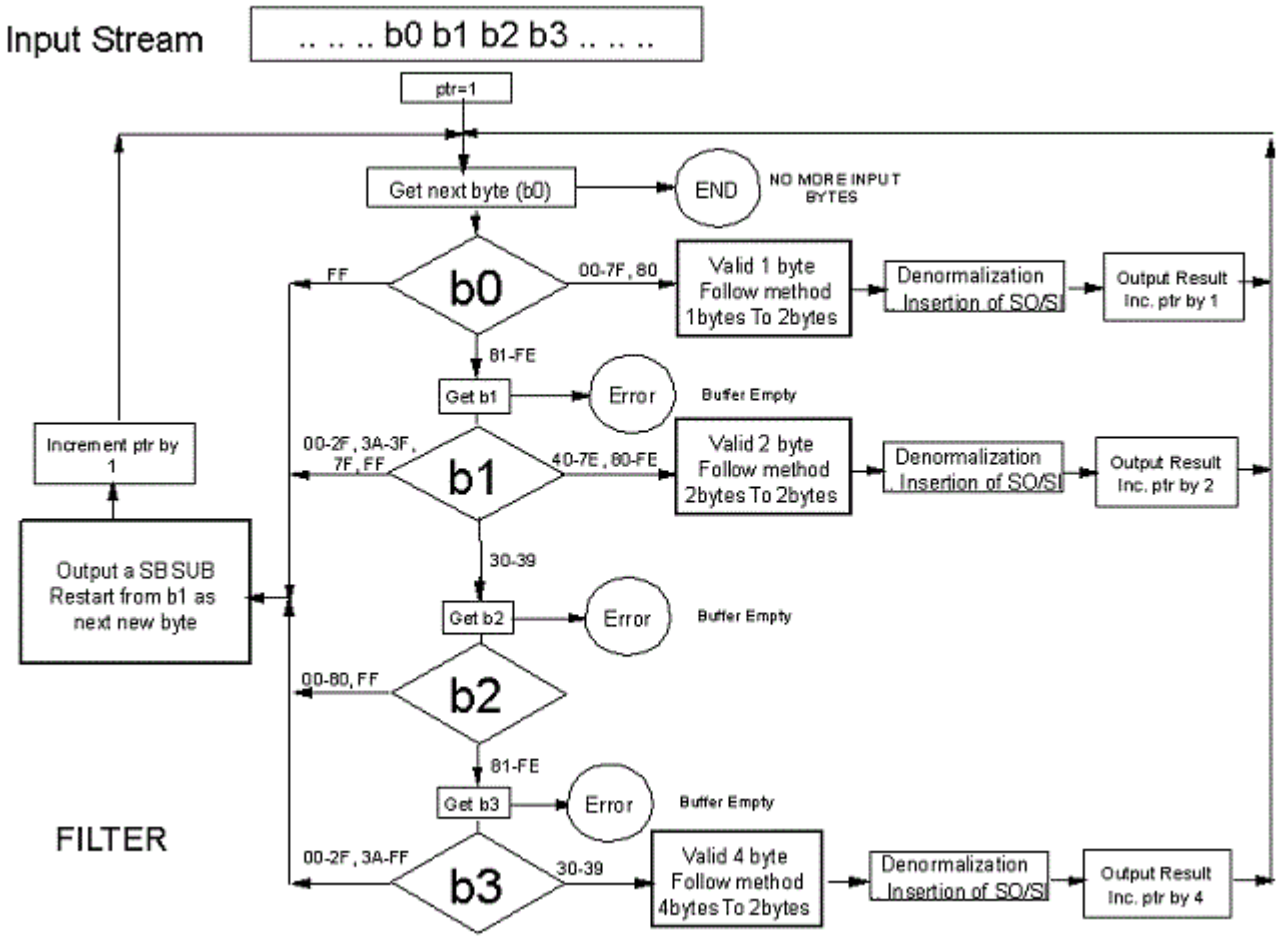


Figure 3.3: Input Filter for GB18030-1 stream

The above steps are repeated until there is no more data in the input stream. If the input stream is exhausted during fetching of any of second, third or fourth bytes in the above filtering logic, the conversion logic cannot proceed, and there will be some unconverted data in the input stream. The pointer into the input stream indicates the first of the remaining bytes that have not been converted. An implementation may choose to deal with such a situation in any suitable manner.

The insertion of X'003F' for the first byte of a broken sequence permits locating where such a sequence might have been in the input by examining the output stream.

Denormalization:

Since the resultant single-bytes in the table have been prefixed with a leading zero-byte, when composing the output string the leading zero-

byte must be removed. Also, all the single-byte sub-strings and the double-byte sub-strings must be bracketed with appropriate controls (SO/SI).

Example: SO 41 41 40 40 SI 41 SO 72 01 SI

1-byte to 2-byte Conversion

File: 1570056C.G1M-R-D

This table with its associated method is used to find the target code point when the source code point is a valid single-byte.

Table format:

The binary conversion table is created by using the existing CDRA Method 5. Figure 3.4 illustrates the table format and the associated method.

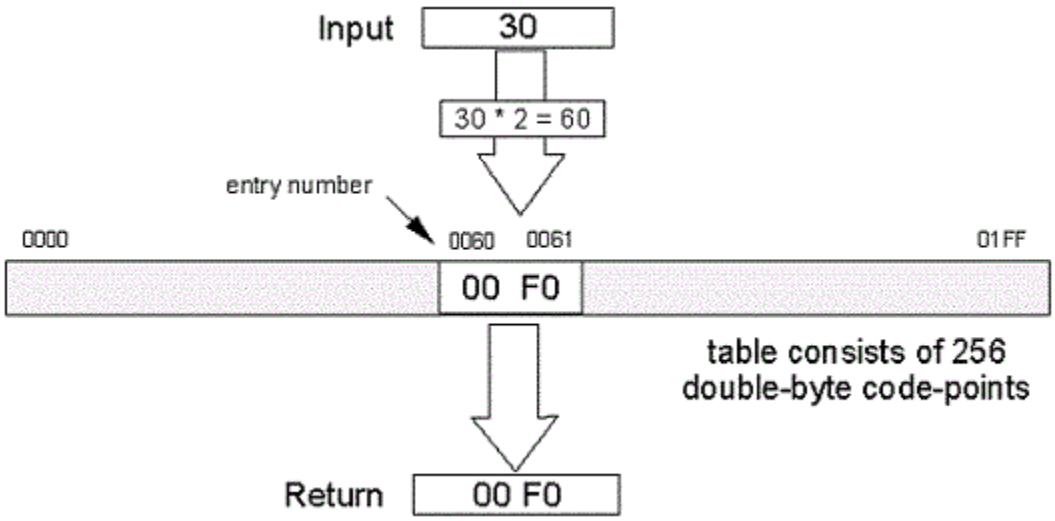


Figure 3.4: GB 1-byte to normalized S-Ch HOST 2-byte Table and Method

The single-byte to normalized HOST table consists of a single 512-byte vector (256 2-byte entries). The source code point is used as a pointer to determine which 2 bytes in the vector represent the target code point. Each target code point in the vector is two bytes long. Therefore the input code point is multiplied by two before calculating the offset into this vector. The binary conversion table is equivalent to existing CDRA SU-R-D binary tables.

2-byte to 2-byte Conversion

File: 1570056C.G2M-R-D

This table is used to find the target code point when the source code point is a valid double-byte.

Table format:

The binary conversion table is created by using CDRA Method 2, a two-step vector lookup method.

The conversion table and the associated method are illustrated in Figure 3.5 below.

This 2-byte to 2-byte table is made up of several 512-byte vectors. The first vector contains 256 single-byte indices (vector numbers) into the rest of the table, followed by 256 unused bytes. The second vector, the "substitute" vector contains the mapping for code points in unassigned wards, and is filled with 256 2-byte SUB code points (X'FEFE' in this case). These are followed by one 512-byte vector for each populated ward in the source encoding.

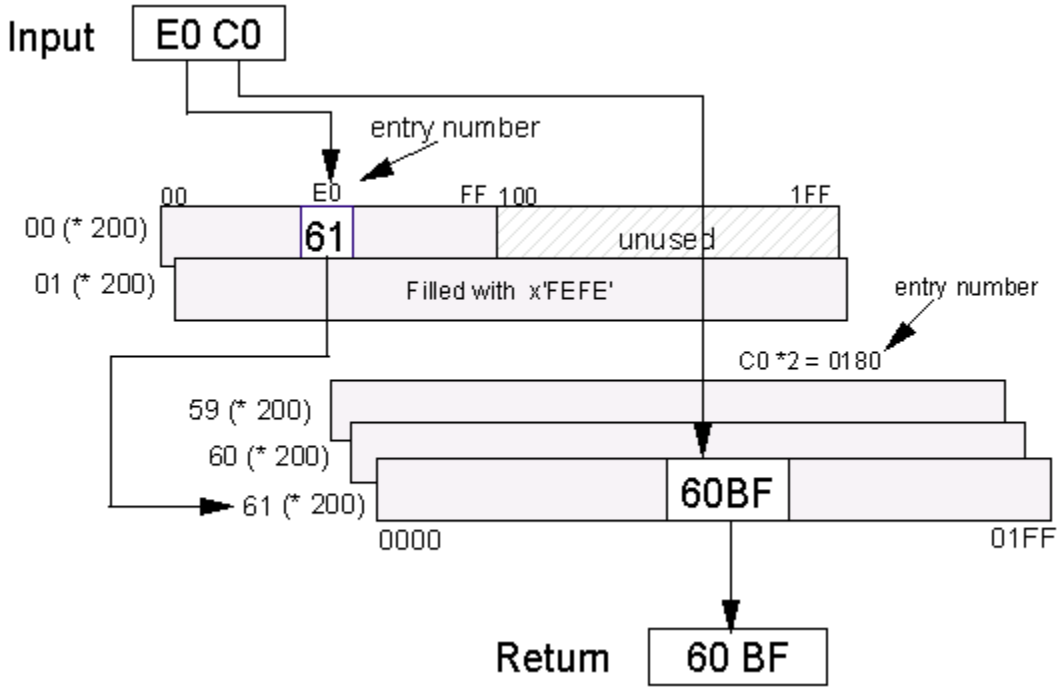


Figure 3.5: GB 2-bytes to normalized S-Ch HOST 2-bytes Table and Method

The table is used as follows. The first byte of the input code point is used as a pointer into the index vector. The single-byte value found at the corresponding position in the index is the vector number in which to perform the second lookup. The second byte of the input code point is used as a pointer into the vector specified by the index vector.

For example in Figure 3.5 the input code point is X'E0C0', you would find a vector number at the X'E0' position in the index vector. The resultant double-byte output code point is found in the specified vector number (x'61') beginning at position X'0180' (which is two times X'C0'), counting into the vector starting at zero.

The vector for handling unassigned wards works as follows. All of the 256 double-byte code point values found in this vector (vector X'01') are those of the "Substitute (SUB)" character of the target encoding (X'FEFE' for S-Ch Host). All the entries in the index vector for unassigned wards point to this "substitute" vector.

The binary conversion table is equivalent to existing CDRA MU-R-D binary tables.

4-byte to 2-byte Conversion

File: 1570056C.G4M-R-D

Table format:

The binary conversion table format and the associated method are detailed below.

This binary table consists of five 1024-byte vectors followed by a large linear array corresponding to the table structure as shown in Figure 3.6. There will be one three-dimensional sub array for each first byte (b0) value used in the table definition. Each sub array will contain the minimum number of cells -- 12600 cells -- needed to map the valid ranges of the second (b1, 10 values), the third (b2, 126 values) and the fourth (b3, 10 values) bytes of four-byte code points in GB. Each cell contains the corresponding 2-byte normalized HOST code point.

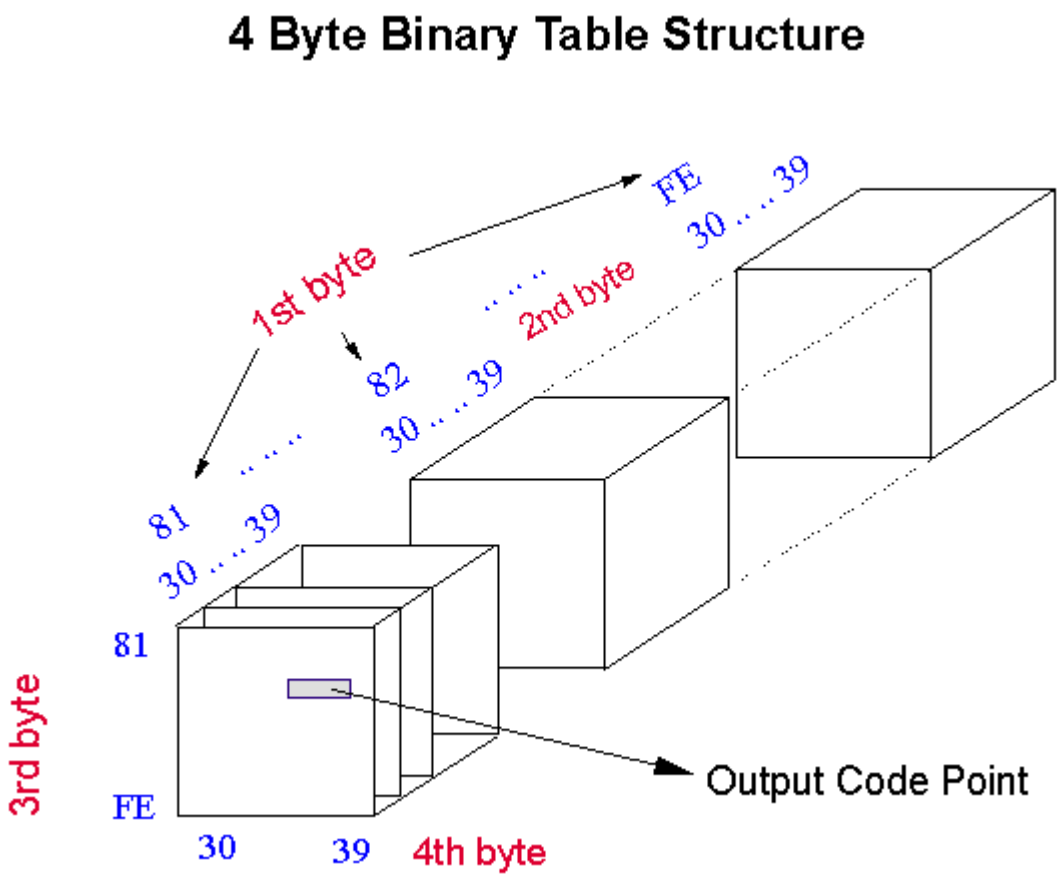


Figure 3.6: GB 4-byte to S-Ch HOST 2-byte table structure

The first 256 bytes of this binary table, called the b0_used_array, is used to check if a particular value of b0 byte is used in the conversion table definition. The next 768 bytes (=3*256) are unused (for now, may change later). These are followed by four 1024-byte vectors -- K40, K41, K42, and K43. These vectors contain values used in computing an index into the rest of the table to get at the output code point.

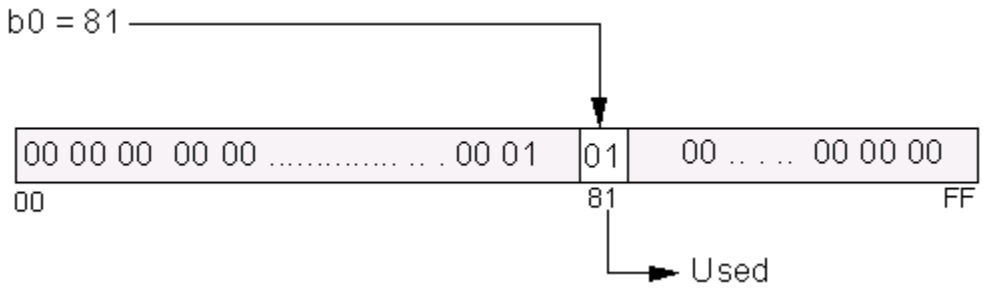


Figure 3.7: b0_used_array

For a given 4-byte code point (b0 b1 b2 b3), if the b0 is not used then there will not be a sub array populated for it. Therefore, before trying to find the target code point, the b0 value is checked to confirm that it has been used. This is done using the b0_used_array shown in Figure 3.7. If the entry at the appropriate location in the b0_used_array is 01 it is valid, if it is 00 it is not used. If it is a 00, then the code point (b0 b1 b2 b3) is unassigned and its mapping is not defined in the conversion table, a double-byte SUB code point (X'FEFE') is inserted into the output data stream. The pointer into the input stream is incremented by 4 in the main filter logic described earlier

If b0 has been used, then proceed with the steps to find target code point, as illustrated in Figure 3.8, and described below.

Get the computed index values from the four index vectors, K40, K41, K42, and K43 using b0, b1, b2, and b3 byte values of the input code point respectively. Each byte of the valid four-byte source code point is used as an offset into the corresponding K4x table to get a part of the index value. When calculating the offset into these vectors there are two things to remember; first, you must begin counting at zero, and second, each entry is four bytes long. This means you have multiply the b0, b1, b2, or b3 by four before looking into the vectors. The resulting 4 values are added to get the location of the first byte of the target code point in the binary table.

The values in the index vectors are computed based on the following formulae:

K40(b0) = 25200 * (block number assigned to the b0 group)

K41(b1) = 2* (b1 - X'30') * (126 * 10)

K42(b2) = 2* (b2 - X'81') * 10

K43(b3) = 2* (b3 - X'30')

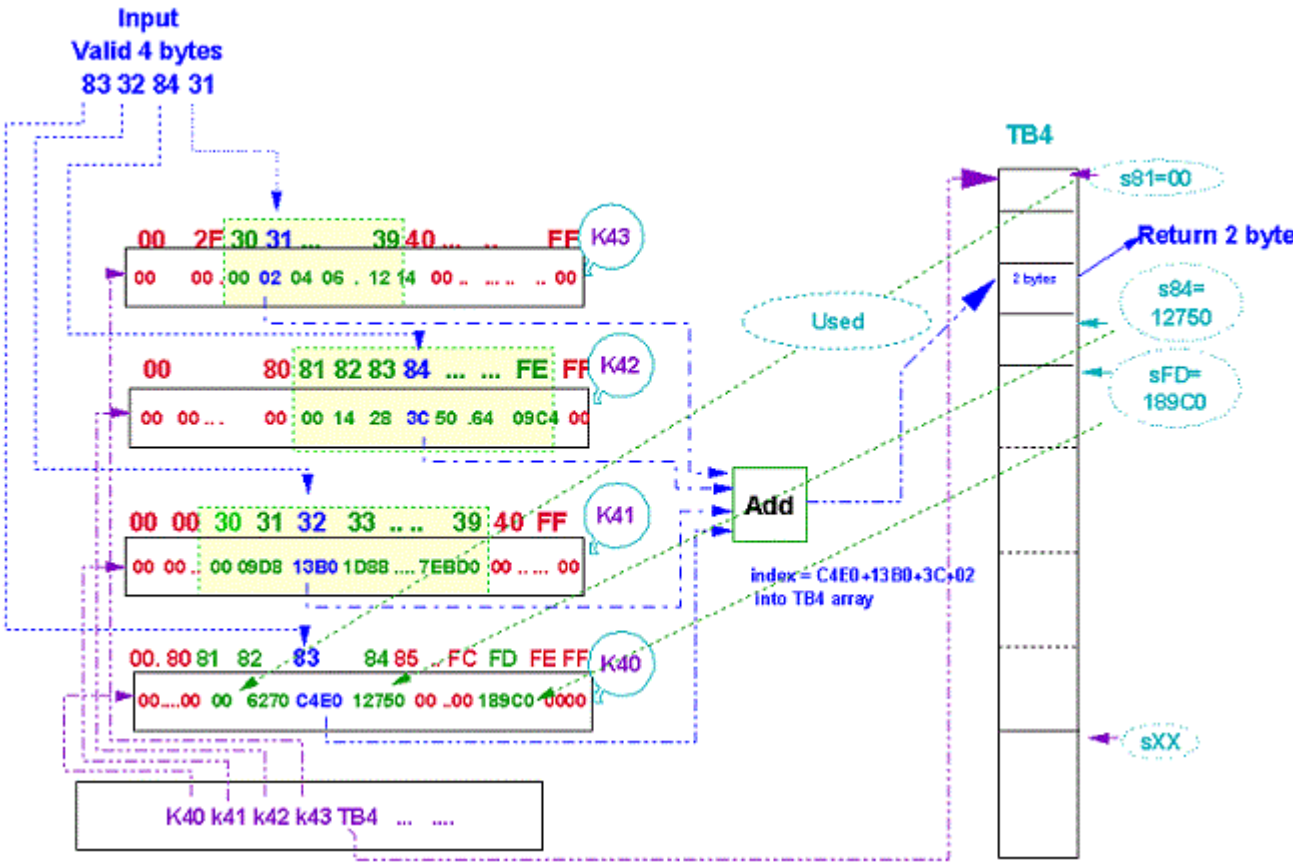


Figure 3.8: Use of Index Arrays in GB 4-byte to normalized S-Ch HOST 2-byte conversion method

This eliminates the multiplication steps during conversion execution, improving the performance. These values are added together to calculate the index (or pointer) to the first byte of the cell containing the output code point (see Figure 3.6). Figure 3.8 shows the mapping table of Figure 6 in a linearized structure.

Also, note that entries in these index arrays for illegal values of b0, b1, b2, or b3, are filled with zero-bytes. They will never be accessed if the filter logic has been followed correctly. Figure 3.8 also shows a possible entry for b0 = X'FD' -- Private Use Area - as a possible fifth block. The binary table currently defined contains mapping tables only for b0 values of X'81' to X'84', as defined in the conversion requirements received from Chinese Government sources.

The two bytes of the output code point are inserted into the output data stream. The pointer into the input data stream is incremented by four in the main filter logic described earlier.

Section 5: Annexes

ANNEX A - CCSIDs for Phase 1 and Phase 2 (as of 2001-06-14)

Note that complete definitions for all CCSIDs can be found in the CCSID repository.

GB 18030 - Phase 1

CCSID		ESID	Comments
Decimal	**Hex**		
05488	1570	2A00	S-ch PC Data mixed for GB 18030
05487	156F	2900	S-ch 4 byte part PC Data for GB 18030(Fixed UCS2 Subset)
09577	2569	2200	S-ch double-byte PC Data double-byte part of GB 18030 (Fixed UCS2 Subset) (*Four-byte SUB to be used)
09444	24E4	4105	S-ch single-byte part of GB 18030

GB 18030 - Phase 2

CCSID		ESID	Comments
Decimal	**Hex**		
01392	0570	2A00	S-ch PC Data mixed for GB 18030
01391	056F	2900	S-ch PC Data 4-byte part of GB 18030(Includes UCS Plane 1-16, and Plane 0 subset)

Host Mixed S-Ch Extended (for GB18030 support)

S-Ch DBCS-Host Data GBK mixed, all GBK character set and other growing chars(GB18030 / Unicode 3.0)

CCSID		ESID	Comments
Decimal	**Hex**		
01388	056C	1301	S-Ch Host mixed for GB 18030-1
13124	3344	1100	S-Ch Host single-byte part of GB 18030-1
04933	1345	1200	S-Ch Host double-byte part of GB 18030-1

09580	256C	1301	S-Ch Host mixed for GBK
00836	0344	1100	S-CH Host single-byte part of GBK
13125	3345	1200	S-Ch Host double-byte part of GBK

Use of Shadow Flags

In chapter 7 under the definition of the Graphic Character Conversion Table (GCCT) Resource, there is a section entitled GCCT Shadow Flag Element which describes the *shadow flags* associated with a conversion table. The structure of the shadow flag tables and the method of getting the shadow flag value are the same as the associated conversion tables and methods, except that the entries in the shadow flag sub-tables will always be single-bytes having one of the shadow flag values X'00' or X'FF', where:

00	Characters match for this code point.
01 - FE	Reserved
FF	Characters do not match for this code point.

Figure 70 shows how the shadow flags are accessed and used in conjunction with a Type 1 table. In this example the input code point X'53' is converted to X'67' as the output code point using the conversion table. Using the associated shadow flag table, the shadow flag entry of X'FF' indicates that the output character is not the same as the input character. The GCGIDs do not match.

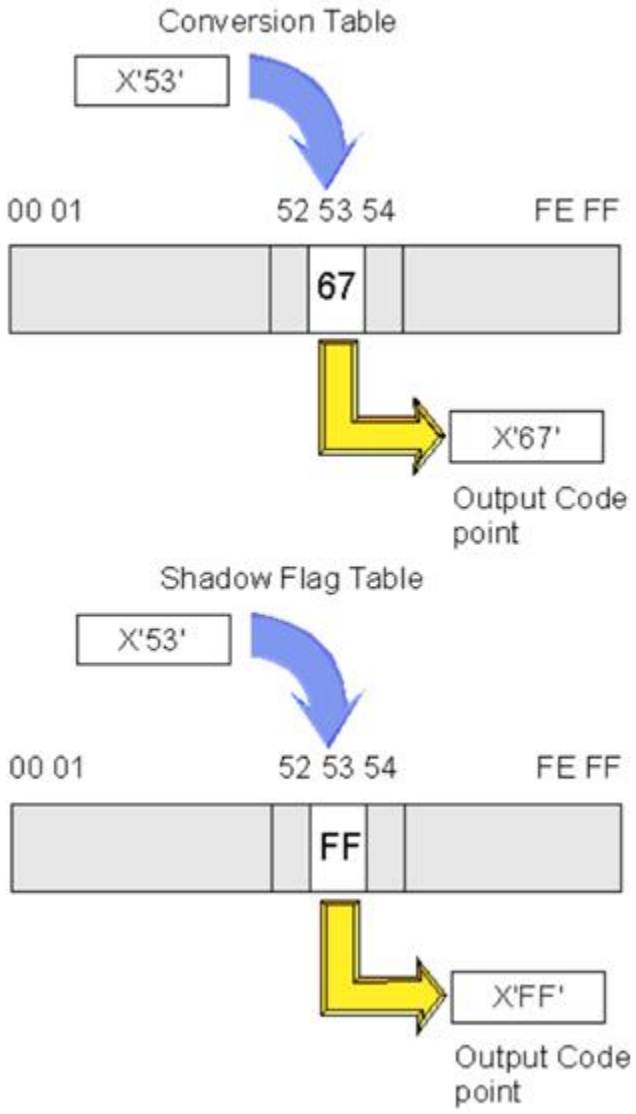


Figure 70. Method 1: Single-Byte to Single-Byte Conversion with Shadow Flags

Figure 71 shows how shadow flags are accessed and used in conjunction with a conversion method; Method 2 in this example, and tables to reflect exact matching of input and output characters. This table can be used to verify that no character replacement has occurred when converting a coded graphic character string.

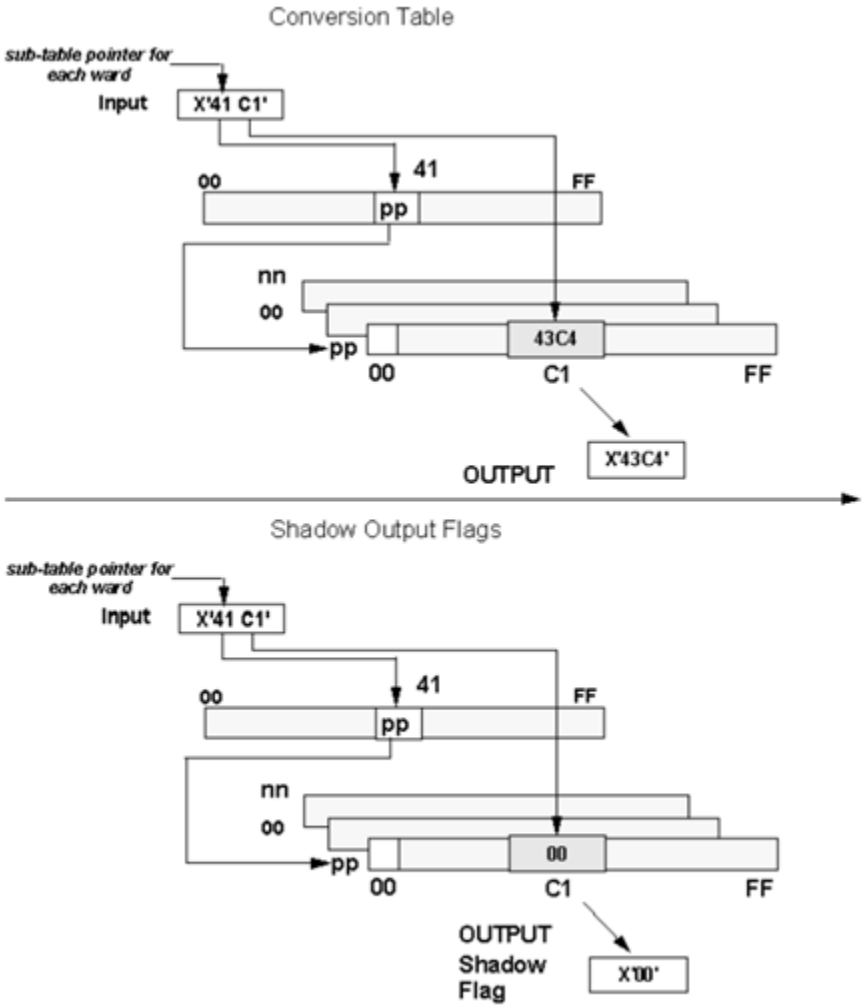


Figure 71. Method 2: Double-Byte to Double-Byte Conversion with Shadow Flags

Enhancements to support string types

Each of the methods described above can be enhanced to deal with different input and output string types.

Null-Terminated

When the input is a null-terminated string, the input parsing step will be enhanced with the necessary logic to sense the null-termination and take appropriate action.

Similarly, when the output string type is null-terminated, the output assembly step of conversion will append a null-termination character to the end of the string. The necessary error-checking logic to ensure that a null-termination character is not encountered in the string converted using the conversion table should also be in place.

See the "Null-terminated string" section in Chapter 6 for the semantics of a null-terminated string.

SPACE Padding

A SPACE padding logic appends the appropriate number of SPACE code points from the relevant CP at the end of the converted string. This enhancement must be added to the output assembly step for these output string types.

See the "Padded string" section in Chapter 6 for the semantics of a SPACE-padded string.

In addition to these two special string types, other string types have been defined in support of bidirectional text. See "Types of strings" in Chapter 6 for specific information.

Appendix C. CCSID Repository

The CCSID Repository is owned and managed by the IBM Globalization Center of Competency (GCoC). If you have questions about specific CCSID values or would like additional information, please contact the GCoC by email at: gcoc@ca.ibm.com. The repository contains detailed information about each CCSID which IBM has registered. Figure 72 shows the fields contained in each CSID record. A list of registered CCSIDs including CCSID number and CCSID name is included below for your reference.

Field	Explanation
CCSID	The value of the coded character set identifier (CCSID) assigned by CDRA. Both decimal and hexadecimal forms are shown. Note: Wherever feasible, CCSID is assigned the same value as its CPGID element, to facilitate coexistence and migration; however, the one-to-one relationship between the two values cannot be maintained in all situations.
Name	A short, descriptive name for the CCSID.
SC	This indicates whether the CCSID is designated for interoperable (I) use, or is required for coexistence and migration purposes (C).
FMS	This indicates whether the character set (CS) is full, maximal, growing or subset. Full (F) Uses all the allocated graphic character space in the ES Maximal (M) Does not use all the allocated graphic character space, but is the largest (maximal) set for the associated ES at the time of registering the value of CCSID by CDRA Growing (G) This indicates that the CCSID has a growing CS. Use the current maximal CS as the character repertoire. Subset (S) Does not use all the allocated graphic character space, and is a subset of another full or maximal registered character set. The number of individual graphic characters (excluding the character SPACE, whose GCGID is SP010000) in the character set is shown in parentheses as (Size).

Registration Date	This field contains the date on which the CCSID became officially registered. Note that this value is not available for some CCSIDs as originally registration dates were not recorded.
Description	A description of the CCSID (may contain the language, country or standard that it is used to support).
Notes	The Notes field will contain additional information about a CCSID such as CCSIDs it has superseded or if it has been replaced by a newer CCSID.
Encoding scheme	The value of the encoding scheme identifier (ES), in hexadecimal form. The number of other identifiers associated with ES depends on this value.
Size	The number of coded graphic characters represented by the CCSID.
MCCSID	The value of the Maximal CCSID. This is the CCSID with the largest character repertoire for a specific Code Page. The Maximal CCSID is always equal to or a superset of the CCSID.
ACRI	This indicates the type of Additional Coding-related Required Information. The format is T-nn, where T is the type of ACRI information, and nn references an entry in the Definition Table for that ACRI Type. See the “Additional Coding-Related Required Information” section in Chapter 3 for definitions of the formats for these various ACRI types. The specific ACRI value definitions are found in the CCSID Repository. • (P) identifies the ACRI information as PCMB (PC Mixed Byte). • (EUC) identifies the ACRI information as EUC • (TCP) identifies the ACRI information as TCP.
Group	Graphic character sets used in different countries to support different languages have been grouped into sets with common properties.
Number of code pages	The number of code page, character set pairs associated with this CCSID.
CS, CP and CCSID	The registered coded graphic character set global identifier (CGCSGID): the GCSGID is in the CS column and the CPGID is in the CP column. Decimal forms are shown. Depending on the ES value, more than one pair of CS and CP may be listed. When the character set associated with a code page is maximal, the CS value associated with a CP may be entered as X'FFFF'

	indicating a growing character set. The contents of the associated code page resource definition in that installation implicitly defines the associated maximal character set. The CCSID column contains the CCSID associated with the corresponding CS, CP pair. This is especially useful when dealing with CCSIDs that contain more than one CS, CP pair.
Control Function Definition	The default control function definitions associated with the CCSID. The possible values for each of the Control Function Definitions are found in the CCSID Repository. • SP (Space) references an entry in the Space Character Definition Table consisting of the code point value (hex), the width in bytes of the code point, and the state in which this code point of SPACE definition is used. • SUB (Substitute) references an entry in the SUB Character Definition Table consisting of the code point value (hex), the width in bytes of the code point, and the state in which this code point of SUB (Substitute) definition is used. • NL (New Line) references an entry in the New Line Character Definition Table consisting of the code point value (hex), the width in bytes of the code point, and the state in which this code point of NL (New Line) definition is used. • LF (Line Feed) references an entry in the Line Feed Character Definition Table consisting of the code point value (hex), the width in bytes of the code point, and the state in which this code point of LF (Line Feed) definition is used. • CR (Carriage Return) references an entry in the Carriage Return Character Definition Table consisting of the code point value (hex), the width in bytes of the code point, and the state in which this code point of CR (Carriage Return) definition is used. • EOF (End of File) references an entry in the End of File Character Definition Table consisting of the code point value (hex), value, the width in bytes of the code point, and the state in which this code point of EOF (End of File) definition is used.

Figure 72. CCSID element descriptions

List of registered CCSIDs

CCSID (decimal)	CCSID (hex)	Name
37	0025	COM EUROPE EBCDIC
256	0100	NETHERLAND EBCDIC
259	0103	SYMBOLS SET 7
273	0111	AUS/GERM EBCDIC
274	0112	BELGIUM EBCDIC
275	0113	BRAZIL EBCDIC
277	0115	DEN/NORWAY EBCDIC
278	0116	FIN/SWEDEN EBCDIC
280	0118	ITALIAN EBCDIC
281	0119	JAPAN EBCDIC
282	011A	PORTUGAL EBCDIC
284	011C	SPANISH EBCDIC
285	011D	UK EBCDIC
286	011E	AUS/GER 3270 EBCD
290	0122	JAPANESE EBCDIC
293	0125	APL
297	0129	FRENCH EBCDIC
300	012C	JAPAN DB EBCDIC
301	012D	JAPAN DB PC-DATA
310	0136	APL/TN
367	016F	US ANSI X3.4 ASCII
420	01A4	ARABIC EBCDIC
421	01A5	MAGHREB/FRENCH EBCDIC
423	01A7	GREEK EBCDIC
424	01A8	HEBREW EBCDIC
425	01A9	Arabic/Latin EBCDIC
437	01B5	USA PC-DATA
500	01F4	INTL EBCDIC
720	02D0	MSDOS ARABIC
737	02E1	MSDOS GREEK
775	0307	MSDOS BALTIC
803	0323	HEBREW EBCDIC
806	0326	PC-ISCII - DV
808	0328	CYRILLIC PC-DATA
813	032D	ISO 8859-7 Greek/Latin
819	0333	ISO 8859-1 ASCII
833	0341	KOREAN EBCDIC
834	0342	KOREAN DB EBCDIC
835	0343	T-CHINESE DB EBCD

CCSID (decimal)	CCSID (hex)	Name
836	0344	S-CHINESE EBCDIC
837	0345	S-CHINESE EBCDIC
838	0346	THAILAND EBCDIC
848	0350	UKRAINE PC-DATA
849	0351	BELARUS PC-DATA
850	0352	LATIN-1 PC-DATA
851	0353	GREEK PC-DATA
852	0354	LATIN-2 PC-DATA
853	0355	TURKISH PC-DATA
855	0357	CYRILLIC PC-DATA
856	0358	HEBREW PC-DATA
857	0359	TURKISH PC-DATA
858	035A	LATIN-1E PC-DATA
859	035B	LATIN-9 PC-DATA
860	035C	PORTUGESE PC-DATA
861	035D	ICELAND PC-DATA
862	035E	HEBREW PC-DATA
863	035F	CANADA PC-DATA
864	0360	ARABIC PC-DATA
865	0361	DEN/NORWAY PC-DAT
866	0362	CYRILLIC PC-DATA
867	0363	HEBREW PC-DATA
868	0364	URDU PC-DATA
869	0365	GREEK PC-DATA
870	0366	LATIN-2 EBCDIC
871	0367	ICELAND EBCDIC
872	0368	CYRILLIC PC-DATA
874	036A	THAI PC-DATA
875	036B	GREEK EBCDIC
876	036C	OCR-A
878	036E	KOI8-R CYRILLIC
880	0370	CYRILLIC EBCDIC
891	037B	KOREA SB PC-DATA
892	037C	Host OCR-A
893	037D	Host OCR-B
895	037F	JAPAN 7-BIT LATIN
896	0380	JAPAN 7-BIT KATAK
897	0381	JAPAN SB PC-DATA
899	0383	SYMBOLS - PC
901	0385	BALTIC ISO-8
902	0386	ESTONIA ISO-8

CCSID (decimal)	CCSID (hex)	Name
903	0387	S-CHINESE PC-DATA
904	0388	T-CHINESE PC-DATA
905	0389	TURKEY EBCDIC
912	0390	ISO 8859-2 ASCII
913	0391	ISO 8859-3 ASCII
914	0392	ISO 8859-4 ASCII
915	0393	ISO 8859-5 ASCII
916	0394	ISO 8859-8 ASCII
918	0396	URDU EBCDIC
920	0398	ISO 8859-9 ASCII
921	0399	ISO 8859-13
922	039A	ESTONIA ISO-8
923	039B	ISO 8859-15 ASCII
924	039C	Latin 9 EBCDIC
926	039E	KOREA DB PC-DATA
927	039F	T-CHINESE PC-DATA
928	03A0	S-CHINESE PC-DATA
930	03A2	JAPAN MIX EBCDIC
931	03A3	JAPAN MIX EBCDIC
932	03A4	JAPAN MIX PC-DATA
933	03A5	KOREA MIX EBCDIC
934	03A6	KOREA MIX PC-DATA
935	03A7	S-CHINESE MIX EBC
936	03A8	S-CHINESE PC-DATA
937	03A9	T-CHINESE MIX EBC
938	03AA	T-CHINESE MIX PC
939	03AB	JAPAN MIX EBCDIC
941	03AD	JAPAN OPEN
942	03AE	JAPAN MIX PC-DATA
943	03AF	JAPAN OPEN
944	03B0	KOREA MIX PC-DATA
946	03B2	S-CHINESE PC-DATA
947	03B3	T-CHINESE BIG-5
948	03B4	T-CHINESE PC-DATA
949	03B5	KOREA KS PC-DATA
950	03B6	T-CH MIX PC-DATA
951	03B7	IBM KS PC-DATA
952	03B8	JAPANESE EUC
953	03B9	JAPANESE EUC
954	03BA	JAPANESE EUC
955	03BB	JAPANESE TCP

CCSID (decimal)	CCSID (hex)	Name
956	03BC	JAPANESE TCP
957	03BD	JAPANESE TCP
958	03BE	JAPANESE TCP
959	03BF	JAPANESE TCP
960	03C0	T-CHINESE EUC
961	03C1	T-CHINESE EUC
963	03C3	T-CHINESE TCP
964	03C4	T-CHINESE EUC
965	03C5	T-CHINESE TCP
970	03CA	KOREAN EUC
971	03CB	KOREAN EUC
1002	03EA	DCF R2
1004	03EC	LATIN-1 PC-DATA
1006	03EE	URDU ISO-8
1008	03F0	ARABIC ISO/ASCII
1009	03F1	ISO-7 OLD IRV
1010	03F2	FRENCH ISO-7 ASCI
1011	03F3	GERM ISO-7 ASCII
1012	03F4	ITALY ISO-7 ASCII
1013	03F5	UK ISO-7 ASCII
1014	03F6	SPAIN ISO-7 ASCII
1015	03F7	PORTUGAL ISO7 ASC
1016	03F8	NOR ISO-7 ASCII
1017	03F9	DENMK ISO-7 ASCII
1018	03FA	FIN/SWE ISO-7 ASC
1019	03FB	BELG/NETH ASCII
1020	03FC	CANADA ISO-7
1021	03FD	SWISS ISO-7
1023	03FF	SPAIN ISO-7
1025	0401	CYRILLIC EBCDIC
1026	0402	TURKEY LATIN-5 EB
1027	0403	JAPAN LATIN EBCD
1040	0410	KOREA PC-DATA
1041	0411	JAPAN PC-DATA
1042	0412	S-CHINESE PC-DATA
1043	0413	T-CHINESE PC-DATA
1046	0416	ARABIC - PC
1047	0417	LATIN OPEN SYS EB
1051	041B	HP EMULATION
1088	0440	KOREA KS PC-DATA
1089	0441	ARABIC ISO 8859-6

CCSID (decimal)	CCSID (hex)	Name
1097	0449	FARSI EBCDIC
1098	044A	FARSI - PC
1100	044C	MULTI EMULATION
1101	044D	BRITISH ISO-7 NRC
1102	044E	DUTCH ISO-7 NRC
1103	044F	FINNISH ISO-7 NRC
1104	0450	FRENCH ISO-7 NRC
1105	0451	NOR/DAN ISO-7 NRC
1106	0452	SWEDISH ISO-7 NRC
1107	0453	NOR/DAN ISO-7 NRC
1112	0458	BALTIC EBCDIC
1114	045A	T-CH SB PC-DATA
1115	045B	S-CH GB PC-DATA
1122	0462	ESTONIA EBCDIC
1123	0463	UKRAINE EBCDIC
1124	0464	UKRAINE ISO-8
1125	0465	UKRAINE PC-DATA
1126	0466	Korean MS-WIN
1127	0467	ARABIC/FR PC-DATA
1129	0469	VIETNAMESE ISO8
1130	046A	VIETNAMESE EBCDIC
1131	046B	BELARUS PC-DATA
1132	046C	LAO EBCDIC
1133	046D	LAO ISO8
1137	0471	DEVANAGARI EBCDIC
1140	0474	COM EUROPE ECECP
1141	0475	AUS/GERM ECECP
1142	0476	DEN/NORWAY ECECP
1143	0477	FIN/SWEDEN ECECP
1144	0478	ITALIAN ECECP
1145	0479	SPANISH ECECP
1146	047A	UK ECECP
1147	047B	FRENCH ECECP
1148	047C	INTL ECECP
1149	047D	ICELAND ECECP
1153	0481	LATIN-2 EBCDIC
1154	0482	CYRILLIC EBCDIC
1155	0483	TURKEY LATIN-5
1156	0484	BALTIC EBCDIC
1157	0485	ESTONIA EBCDIC
1158	0486	UKRAINE EBCDIC

CCSID (decimal)	CCSID (hex)	Name
1159	0487	T-CHINESE EBCDIC
1160	0488	THAI SB EBCDIC
1161	0489	THAI SB PC-DATA
1162	048A	THAI WINDOWS
1163	048B	VIETNAMESE ISO8
1164	048C	VIETNAMESE EBCDIC
1165	048D	LATIN-2 OPEN SYS EB
1166	048E	Cyrillic Multilingual - Kazakhstan
1167	048F	KOI8-RU
1168	0490	KOI8-U
1174	0496	Windows Cyrillic (Kazakhstan)
1200	04B0	UTF-16 BE with IBM PUA
1201	04B1	UTF-16 BE
1202	04B2	UTF-16 LE with IBM PUA
1203	04B3	UTF-16 LE
1204	04B4	UTF-16 with IBM PUA
1205	04B5	UTF-16
1208	04B8	UTF-8 with IBM PUA
1209	04B9	UTF-8
1210	04BA	UTF-EBCDIC with IBM PUA
1211	04BB	UTF-EBCDIC
1212	04BC	SCSU with IBM PUA
1213	04BD	SCSU
1214	04BE	BOCU-1 with IBM PUA
1215	04BF	BOCU-1
1232	04D0	UTF-32 BE with IBM PUA
1233	04D1	UTF-32 BE
1234	04D2	UTF-32 LE with IBM PUA
1235	04D3	UTF-32 LE
1236	04D4	UTF-32 with IBM PUA
1237	04D5	UTF-32
1250	04E2	MS-WIN LATIN-2
1251	04E3	MS-WIN CYRILLIC
1252	04E4	MS-WIN LATIN-1
1253	04E5	MS-WIN GREEK
1254	04E6	MS-WIN TURKEY
1255	04E7	MS-WIN HEBREW
1256	04E8	MS-WIN ARABIC
1257	04E9	MS-WIN BALTIC
1258	04EA	MS-WIN VIETNAM
1275	04FB	APPLE LATIN-1

CCSID (decimal)	CCSID (hex)	Name
1276	04FC	ADOBE STANDARD
1277	04FD	ADOBE LATIN-1
1280	0500	APPLE GREEK
1281	0501	APPLE TURKEY
1282	0502	APPLE LATIN2
1283	0503	APPLE CYRILLIC
1284	0504	APPLE CROATIAN
1285	0505	APPLE ROMANIAN
1286	0506	APPLE ICELANDIC
1287	0507	DEC Greek 8-Bit
1288	0508	DEC Turkish 8-Bit
1350	0546	JISeucJP
1351	0547	JAPAN OPEN
1362	0552	KOREAN MS-WIN
1363	0553	KOREAN MS-WIN
1364	0554	KOREAN EBCDIC
1370	055A	T-CH MIX PC-DATA
1371	055B	T-CHINESE MIX EBC
1374	055E	DB Big-5 extension for HKSCS
1375	055F	Mixed Big-5 Ext for HKSCS
1376	0560	Host HKSCS DBCS growing
1377	0561	Mixed Host HKSCS Growing
1380	0564	S-CH GB PC-DATA
1381	0565	S-CH GB PC-DATA
1382	0566	S-CHINESE EUC
1383	0567	S-CHINESE EUC
1385	0569	S-CH GBK PC-DATA
1386	056A	S-CH GBK PC-DATA
1388	056C	S-CHINESE Mixed EBCDIC
1390	056E	JAPAN MIX EBCDIC
1391	056F	S-ch 4 byte, growing CS, for GB 18030
1392	0570	S-ch PC Data mixed (growing) GB18030
1393	0571	Shift_JISx0213 - Double-byte portion
1394	0572	Shift_JISx0213
1399	0577	JAPAN MIX EBCDIC
1400	0578	Unicode BMP
1401	0579	Unicode Plane 1
1402	057A	Unicode Plane 2
1414	0586	Unicode Plane 14

CCSID (decimal)	CCSID (hex)	Name
1446	05A6	Unicode Plane 15
1447	05A7	Unicode Plane 16
1448	05A8	Unicode, Generic PUA of BMP
1449	05A9	Unicode, PUA of BMP, IBM Default
2105	0839	LCS 3800-1 TEXT 1 & 2
4133	1025	USA EBCDIC
4369	1111	AUS/GERMAN EBCDIC
4370	1112	BELGIUM EBCDIC
4371	1113	BRAZIL EBCDIC
4372	1114	CANADA EBCDIC
4373	1115	DEN/NORWAY EBCDIC
4374	1116	FIN/SWEDEN EBCDIC
4376	1118	ITALY EBCDIC
4378	111A	PORTUGAL EBCDIC
4380	111C	LATIN EBCDIC
4381	111D	UK EBCDIC
4386	1122	JAPAN EBCDIC SB
4389	1125	APL
4393	1129	FRANCE EBCDIC
4396	112C	JAPAN EBCDIC DB
4397	112D	JAPAN DB PC-DATA
4516	11A4	ARABIC EBCDIC
4517	11A5	MAGHR/FREN EBCDIC
4519	11A7	GREEK EBCDIC 3174
4520	11A8	HEBREW EBCDIC
4533	11B5	SWISS PC-DATA
4596	11F4	LATIN AMER EBCDIC
4899	1323	HEBREW EBCDIC
4902	1326	PC-ISCII - DV
4904	1328	CYRILLIC MS-PC-DATA
4909	132D	GREEK/LATIN ASCII with euro
4929	1341	KOREA SB EBCDIC
4930	1342	KOREAN DB EBCDIC
4931	1343	T-CHINESE DB EBCD
4932	1344	S-CHINESE EBCDIC
4933	1345	S-CHINESE EBCDIC
4934	1346	THAI SB EBCDIC
4944	1350	UKRAINE MS-PC-DATA
4945	1351	BELARUS MS-PC-DATA
4946	1352	LATIN-1 PC-DATA
4947	1353	GREEK PC-DATA

CCSID (decimal)	CCSID (hex)	Name
4948	1354	LATIN-2 PC-DATA
4949	1355	TURKEY PC-DATA
4951	1357	CYRILLIC PC-DATA
4952	1358	HEBREW PC-DATA
4953	1359	TURKEY PC-DATA
4954	135A	LATIN-1E MS-PC-DATA
4955	135B	LATIN-9 MS-PC-DATA
4956	135C	PORTUGESE MS-PC-DATA
4957	135D	ICELAND MS-PC-DATA
4958	135E	HEBREW MS-PC-DATA
4959	135F	CANADA MS-PC-DATA
4960	1360	ARABIC PC-DATA
4961	1361	DEN/NORWAY PC-DAT
4962	1362	CYRILLIC MS-PC-DATA
4963	1363	HEBREW MS-PC-DATA
4964	1364	URDU PC-DATA
4965	1365	GREEK PC-DATA
4966	1366	ROECE LATIN-2 EBC
4967	1367	ICELAND EBCDIC
4968	1368	CYRILLIC PC-DATA
4970	136A	THAI SB PC-DATA
4971	136B	GREEK EBCDIC
4976	1370	CYRILLIC EBCDIC
4992	1380	JAPANESE TCP- 2022
4993	1381	JAPAN SB PC-DATA
5012	1394	ISO 8859-8 ASCII
5014	1396	URDU EBCDIC
5023	139F	T-CHINESE PC-DATA
5026	13A2	JAPAN MIX EBCDIC
5028	13A4	JAPAN MIX PC-DATA
5029	13A5	KOREA MIX EBCDIC
5031	13A7	S-CH MIXED EBCDIC
5033	13A9	T-CHINESE EBCDIC
5035	13AB	JAPAN MIX EBCDIC
5037	13AD	JAPAN OPEN
5038	13AE	JAPAN HP15-J
5039	13AF	JAPAN OPEN
5043	13B3	T-CHINESE BIG-5
5045	13B5	KOREA KS PC-DATA
5046	13B6	T-CHINESE BIG-5
5047	13B7	KOREA KS PC DATA

CCSID (decimal)	CCSID (hex)	Name
5048	13B8	JAPANESE EUC
5049	13B9	JAPANESE EUC
5050	13BA	JAPANESE EUC
5052	13BC	JAPANESE TCP
5053	13BD	JAPANESE TCP
5054	13BE	JAPANESE TCP
5055	13BF	JAPANESE TCP
5056	13C0	T-Ch TCP-2022, G1-CNS
5057	13C1	T-CHINESE EUC
5060	13C4	T-CHINESE EUC
5066	13CA	KOREAN EUC
5067	13CB	KOREAN EUC
5100	13EC	LATIN-1 PC-DATA
5104	13F0	ARABIC ISO/ASCII
5123	1403	JAPAN LATIN EBCD
5137	1411	JAPAN PC-DATA
5142	1416	ARABIC - PC
5143	1417	LATIN OPEN SYS
5210	145A	S-CH SB PC-DATA
5211	145B	S-CH GB PC-DATA
5222	1466	Korean MS-WIN
5233	1471	DEVANAGARI EBCDIC with Rupee
5255	1487	T-CHINESE EBCDIC
5304	14B8	Unicode 2.0, UTF-8 with IBM PUA
5305	14B9	Unicode 2.0, UTF-8
5328	14D0	Unicode 4.0, UTF-32 BE with IBM PUA
5346	14E2	MS-WIN LATIN-2
5347	14E3	MS-WIN CYRILLIC
5348	14E4	MS-WIN LATIN-1
5349	14E5	MS-WIN GREEK
5350	14E6	MS-WIN TURKEY
5351	14E7	MS-WIN HEBREW with euro
5352	14E8	MS-WIN ARABIC
5353	14E9	MS-WIN BALTIC
5354	14EA	MS-WIN VIETNAM
5458	1552	KOREAN MS-WIN
5459	1553	KOREAN MS-WIN
5460	1554	KOREAN EBCDIC
5470	155E	DB Big-5 ext for HKSCS-2001

CCSID (decimal)	CCSID (hex)	Name
5471	155F	Mixed Big-5 ext for HKSCS-2001
5472	1560	Host HKSCS-2001, DB
5473	1561	Host Mixed HKSCS-2001
5476	1564	S-CH GB PC-DATA
5477	1565	S-CH GB PC-DATA
5478	1566	S-CHINESE EUC
5479	1567	S-CHINESE EUC
5481	1569	S-CH GBK PC-DATA
5482	156A	S-CH GBK PC-DATA
5484	156C	S-CHINESE MIX EBC
5486	156E	JAPAN MIX EBCDIC
5487	156F	S-ch 4 byte for GB 18030
5488	1570	S-ch PC Data mixed (fixed) GB18030
5495	1577	JAPAN MIX EBCDIC
5496	1578	Unicode 2.0, BMP
5497	1579	Unicode 4.0, Plane 1
5498	157A	Unicode 4.0, Plane 2
5510	1586	Unicode 4.0, Plane 14
6201	1839	LCS 3800-1 Extended
8229	2025	INTL EBCDIC
8448	2100	INTL EBCDIC
8476	211C	SPAIN EBCDIC
8482	2122	JAPAN EBCDIC SB
8489	2129	FRANCE EBCDIC
8492	212C	JAPAN EBCDIC DB
8493	212D	JAPAN HP15-J
8612	21A4	ARABIC EBCDIC
8616	21A8	HEBREW EBCDIC
8629	21B5	AUS/GERM PC-DATA
8692	21F4	AUS/GERMAN EBCDIC
9005	232D	ISO 8859-7:2003 Greek/Latin
9025	2341	KOREA SB EBCDIC
9026	2342	KOREA DB EBCDIC
9027	2343	T-CHINESE DB EBCD
9028	2344	S-CHINESE EBCDIC
9029	2345	S-CHINESE EBCDIC
9030	2346	THAI SB EBCDIC
9042	2352	LATIN-1 MS-PC-DATA
9044	2354	LATIN-2 PC-DATA
9047	2357	CYRILLIC PC-DATA
9048	2358	HEBREW PC-DATA

CCSID (decimal)	CCSID (hex)	Name
9049	2359	TURKISH PC-DATA
9056	2360	ARABIC PC-DATA
9060	2364	URDU PC-DATA
9061	2365	GREEK PC-DATA
9064	2368	CYRILLIC MS-PC-DATA
9066	236A	THAI SB PC-DATA
9067	236B	Greek EBCDIC - 2005
9088	2380	Japanese EUC, G2-JIS
9089	2381	JAPAN PC-DATA SB
9122	23A2	JAPAN MIX EBCDIC
9124	23A4	JAPAN MIX PC-DATA
9125	23A5	KOREA MIX EBCDIC
9127	23A7	S-CH MIXED EBCDIC
9131	23AB	JAPAN MIX EBCDIC
9133	23AD	JAPAN DB PC
9135	23AF	JAPAN MIXED PC
9139	23B3	T-CHINESE BIG-5
9142	23B6	T-CHINESE BIG-5
9144	23B8	Japanese TCP-2022, G1
9145	23B9	JAPANESE EUC
9146	23BA	JAPANESE EUC
9148	23BC	JAPANESE TCP
9163	23CB	Korean EUC, G1
9219	2403	JAPAN LATIN EBCDIC
9238	2416	ARABIC - PC
9306	245A	S-CH SB MS-PC-DATA
9400	24B8	CESU-8 with IBM PUA
9424	24D0	Unicode 4.1, UTF-32 BE with IBM PUA
9444	24E4	S-Ch SB part of GB 18030
9447	24E7	MS-WIN HEBREW-2001
9448	24E8	MS-WIN ARABIC-2001
9449	24E9	MS-WIN BALTIC-2001
9554	2552	KOREAN MS-WIN
9555	2553	KOREAN MS-WIN
9563	255B	T-CHINESE MIX EBC
9566	255E	DB Big-5 ext for HKSCS-2004
9567	255F	Mixed Big-5 ext for HKSCS-2004
9568	2560	DB, Host HKSCS-2004
9569	2561	Mixed Host HKSCS-2004
9572	2564	S-CH GB PC-DATA

CCSID (decimal)	CCSID (hex)	Name
9574	2566	S-CHINESE EUC
9575	2567	S-CHINESE TCP
9577	2569	S-CH GBK PC-DATA
9580	256C	S-Ch Host mixed for GBK
9582	256E	JAPAN MIX EBCDIC - JIS X0213:2004
9591	2577	JAPAN MIX EBCDIC - JIS X0213:2004
9592	2578	Unicode 3.0, BMP
9593	2579	Unicode 4.1, Plane 1
9594	257A	Unicode 5.2, Plane 2
12325	3025	CANADA EBCDIC
12544	3100	FRANCE EBCDIC
12578	3122	JAPAN EBCDIC SB
12588	312C	JAPAN EBCDIC DB
12708	31A4	ARABIC EBCDIC
12712	31A8	HEBREW EBCDIC
12725	31B5	FRANCE PC-DATA
12788	31F4	ITALY EBCDIC
13121	3341	KOREA SB EBCDIC
13122	3342	KOREAN DB EBCDIC
13124	3344	S-CHINESE EBCDIC
13125	3345	S-Ch Host- double-byte for GBK
13140	3354	LATIN-2 MS-PC-DATA
13143	3357	CYRILLIC MS-PC-DATA
13144	3358	HEBREW PC-DATA
13145	3359	TURKISH MS-PC-DATA
13152	3360	ARABIC PC-DATA
13156	3364	URDU MS-PC-DATA
13157	3365	GREEK MS-PC-DATA
13162	336A	THAI MS-PC-DATA
13184	3380	JAPAN 7-BIT KATAK
13185	3381	JAPAN PC-DATA SB
13218	33A2	JAPAN MIX EBCDIC
13219	33A3	JAPAN MIX EBCDIC
13221	33A5	KOREA MIX EBCDIC
13223	33A7	S-CH MIXED EBCDIC
13229	33AD	JAPAN DB PC
13231	33AF	JAPAN MIXED PC
13235	33B3	T-CHINESE BIG-5
13238	33B6	T-CHINESE BIG-5

CCSID (decimal)	CCSID (hex)	Name
13240	33B8	JAPANESE TCP-2022
13241	33B9	Japanese TCP-2022, G3
13242	33BA	JAPANESE EUC
13259	33CB	KOREAN EUC
13488	34B0	Unicode 2.0, UTF-16 BE with IBM PUA
13489	34B1	Unicode 2.0, UTF-16 BE
13490	34B2	Unicode 2.0, UTF-16 LE with IBM PUA
13491	34B3	Unicode 2.0, UTF-16 LE
13496	34B8	Unicode 3.0, UTF-8 with IBM PUA
13497	34B9	Unicode 3.0, UTF-8
13520	34D0	Unicode 5.0, UTF-32 BE with IBM PUA
13650	3552	KOREAN MS-WIN
13651	3553	KOREAN MS-WIN
13662	355E	DB Big-5 ext for HKSCS-2008
13663	355F	Mixed Big-5 ext for HKSCS-2008
13664	3560	DB, Host HKSCS-2008
13665	3561	Mixed Host HKSCS-2008
13676	356C	S-Ch Host mixed for GBK
13688	3578	Unicode 4.0, BMP
13689	3579	Unicode 5.0, Plane 1
13690	357A	Unicode 6.0, Plane 2
16421	4025	CANADA EBCDIC
16684	412C	JAPAN DB EBCDIC
16804	41A4	ARABIC EBCDIC
16821	41B5	ITALY PC-DATA
16884	41F4	FIN/SWEDEN EBCDIC
17218	4342	KOREAN DB EBCDIC
17219	4343	T-CHINESE DB EBCDIC
17221	4345	S-CH Host - double-byte for GBK
17240	4358	HEBREW MS-PC-DATA
17248	4360	ARABIC PC-DATA
17314	43A2	JAPAN MIX EBCDIC
17317	43A5	KOREA MIX EBCDIC
17325	43AD	JAPAN DB PC
17331	43B3	T-CHINESE BIG-5
17336	43B8	JAPANESE TCP-2022
17337	43B9	Japanese TCP-2022 G3-JIS
17338	43BA	JAPANESE EUC

CCSID (decimal)	CCSID (hex)	Name
17354	43CA	KOREAN TCP
17584	44B0	Unicode 3.0, UTF-16 BE with IBM PUA
17585	44B1	Unicode 3.0, UTF-16 BE
17586	44B2	Unicode 3.0, UTF-16 LE with IBM PUA
17587	44B3	Unicode 3.0, UTF-16 LE
17592	44B8	Unicode 4.0, UTF-8 with IBM PUA
17593	44B9	Unicode 4.0, UTF-8
17616	44D0	Unicode 5.1, UTF-32 BE with IBM PUA
17784	4578	Unicode 4.1, BMP
17785	4579	Unicode 5.1, Plane 1
20517	5025	PORTUGAL EBCDIC
20780	512C	JAPAN DB EBCDIC
20917	51B5	UK PC-DATA
20980	51F4	DEN/NORWAY EBCDIC
21314	5342	Korean DB EBCDIC
21317	5345	S-CHINESE EBCDIC
21344	5360	ARABIC MS-PC-DATA
21427	53B3	T-CHINESE BIG-5
21432	53B8	JAPANESE EUC - growing
21433	53B9	JAPANESE EUC
21434	53BA	JAPANESE EUC - growing
21450	53CA	KOREAN TCP
21680	54B0	Unicode 4.0, UTF-16 BE with IBM PUA
21681	54B1	Unicode 4.0, UTF-16 BE
21682	54B2	Unicode 4.0, UTF-16 LE with IBM PUA
21683	54B3	Unicode 4.0, UTF-16 LE
21688	54B8	Unicode 4.1, UTF-8 with IBM PUA
21689	54B9	Unicode 4.1, UTF-8
21712	54D0	Unicode 5.2, UTF-32 BE with IBM PUA
21880	5578	Unicode 5.0, BMP
21881	5579	Unicode 5.2, Plane 1
24613	6025	INTL EBCDIC
24876	612C	Extended Japanese Host
24877	612D	JAPAN DB PC-DISPL
25013	61B5	USA PC-DISPLAY
25076	61F4	DEN/NORWAY EBCDIC

CCSID (decimal)	CCSID (hex)	Name
25313	62E1	MSDOS GREEK
25351	6307	MSDOS BALTIC
25384	6328	CYRILLIC PC-DISP
25424	6350	UKRAINE PC-DISP
25425	6351	BELARUS PC-DISP
25426	6352	LATIN-1 PC-DISP
25427	6353	GREECE PC-DISPLAY
25428	6354	LATIN-2 PC-DISP
25429	6355	TURKEY PC-DISPLAY
25431	6357	CYRILLIC PC-DISP
25432	6358	HEBREW PC-DISPLAY
25433	6359	TURKEY PC-DISPLAY
25434	635A	LATIN-1E PC-DISP
25435	635B	LATIN-9 PC-DISP
25436	635C	PORTUGAL PC-DISP
25437	635D	ICELAND PC-DISP
25438	635E	HEBREW PC-DISPLAY
25439	635F	CANADA PC-DISPLAY
25440	6360	ARABIC PC-DISPLAY
25441	6361	DEN/NOR PC-DISP
25442	6362	CYRILLIC PC-DISP
25443	6363	HEBREW PC-DISPLAY
25444	6364	URDU PC-DISPLAY
25445	6365	GREECE PC-DISPLAY
25448	6368	CYRILLIC PC-DISP
25450	636A	THAILAND PC-DISP
25467	637B	KOREA SB PC-DISP
25473	6381	JAPAN SB PC-DISP
25477	6385	BALTIC ISO-8
25478	6386	ESTONIA ISO-8
25479	6387	S-CHIN SB PC-DISP
25480	6388	T-ChINESE PC-DISP
25488	6390	ISO 8859-2 ASCII
25491	6393	ISO 8859-5 ASCII
25497	6399	BALTIC ISO-8
25498	639A	ESTONIA ISO-8
25502	639E	KOREA DB PC-DISP
25503	639F	T-CHINESE PC-DISP
25504	63A0	S-CHINESE PC-DISP
25505	63A1	THAILAND PC-DISP
25508	63A4	JAPAN PC-DISPLAY

CCSID (decimal)	CCSID (hex)	Name
25510	63A6	KOREA PC-DISPLAY
25512	63A8	S-CHINESE PC-DISP
25514	63AA	T-CHINESE PC-DISP
25518	63AE	JAPAN PC-DISPLAY
25520	63B0	KOREA PC-DISPLAY
25522	63B2	S-CHINESE PC-DISP
25524	63B4	T-CHINESE PC-DISP
25525	63B5	KOREA KS PC-DISP
25527	63B7	KOREA KS PC-DISP
25528	63B8	JAPANESE EUC
25530	63BA	JAPANESE EUC
25546	63CA	KOREAN TCP
25580	63EC	LATIN-1 PC-DISPLAY
25616	6410	KOREA SB PC-DISP
25617	6411	JAPAN PC-DISPLAY
25618	6412	S-CHINESE PC-DISP
25619	6413	T-CHINESE PC-DISP
25664	6440	KOREA KS PC-DISP
25690	645A	T-CH SB PC-DISP
25691	645B	S-CH GB PC-DATA
25702	6466	Korean MS-WIN
25703	6467	ARABIC/FR PC-DISP
25776	64B0	Unicode 4.1, UTF-16 BE with IBM PUA
25777	64B1	Unicode 4.1, UTF-16 BE
25778	64B2	Unicode 4.1, UTF-16 LE with IBM PUA
25779	64B3	Unicode 4.1, UTF-16 LE
25784	64B8	Unicode 5.0, UTF-8 with IBM PUA
25785	64B9	Unicode 5.0, UTF-8
25808	64D0	Unicode 6.0, UTF-32 BE with IBM PUA
25976	6578	Unicode 5.1, BMP
25977	6579	Unicode 6.0, Plane 1
28709	7025	T-CHINESE EBCDIC
29109	71B5	USA PC-DISPLAY
29172	71F4	BRAZIL EBCDIC
29522	7352	LATIN-1 PC-DISP
29523	7353	GREECE PC-DISPLAY
29524	7354	LATIN-2 PC-DISP
29525	7355	TURKEY PC-DISPLAY

CCSID (decimal)	CCSID (hex)	Name
29527	7357	CYRILLIC PC-DISP
29528	7358	HEBREW PC-DISPLAY
29529	7359	TURKEY PC-DISPLAY
29532	735C	PORTUGAL PC-DISP
29533	735D	ICELAND PC-DISP
29534	735E	HEBREW PC-DISPLAY
29535	735F	CANADA PC-DISPLAY
29536	7360	ARABIC PC-DISPLAY
29537	7361	DEN/NOR PC-DISP
29540	7364	URDU PC-DISPLAY
29541	7365	GREECE PC-DISPLAY
29546	736A	THAILAND PC-DISP
29614	73AE	JAPAN PC-DISPLAY
29616	73B0	KOREA PC-DISPLAY
29618	73B2	S-CHINESE PC-DISP
29620	73B4	T-CHINESE PC-DISP
29621	73B5	KOREA KS MIX PC
29623	73B7	KOREA KS PC-DISP
29626	73BA	JAPANESE EUC
29712	7410	KOREA PC-DISPLAY
29713	7411	JAPAN PC-DISPLAY
29714	7412	S-CHINESE PC-DISP
29715	7413	T-CHINESE PC-DISP
29760	7440	KOREA KS PC-DISP
29786	745A	S-CH SB PC-DISP
29872	74B0	Unicode 5.0, UTF-16 BE with IBM PUA
29873	74B1	Unicode 5.0, UTF-16 BE
29874	74B2	Unicode 5.0, UTF-16 LE with IBM PUA
29875	74B3	Unicode 5.0, UTF-16 LE
29880	74B8	Unicode 5.1, UTF-8 with IBM PUA
29881	74B9	Unicode 5.1, UTF-8
30072	7578	Unicode 5.2, BMP
32805	8025	JAPAN LATIN EBCDC
33058	8122	JAPAN EBCDIC
33205	81B5	SWISS PC-DISPLAY
33268	81F4	UK/PORTUGAL EBCDC
33618	8352	LATIN-1 PC-DISP
33619	8353	GREECE PC-DISPLAY
33620	8354	ROECE PC-DISPLAY

CCSID (decimal)	CCSID (hex)	Name
33621	8355	TURKEY PC-DISPLAY
33623	8357	CYRILLIC PC-DISP
33624	8358	HEBREW PC-DISPLAY
33625	8359	TURKEY PC-DISPLAY
33632	8360	ARABIC PC-DISPLAY
33636	8364	URDU PC-DISPLAY
33637	8365	GREECE PC-DISPLAY
33665	8381	JAPAN PC-DISPLAY
33698	83A2	JAPAN KAT/KAN EBC
33699	83A3	JAPAN LAT/KAN EBC
33700	83A4	JAPAN PC-DISPLAY
33717	83B5	KOREA KS PC-DISP
33722	83BA	IBMeucJP
33968	84B0	Unicode 5.1, UTF-16 BE with IBM PUA
33969	84B1	Unicode 5.1, UTF-16 BE
33970	84B2	Unicode 5.1, UTF-16 LE with IBM PUA
33971	84B3	Unicode 5.1, UTF-16 LE
33976	84B8	Unicode 5.2, UTF-8 with IBM PUA
33977	84B9	Unicode 5.2, UTF-8
34168	8578	Unicode 6.0, BMP
37301	91B5	AUS/GERM PC-DISP
37364	91F4	BELGIUM EBCDIC
37716	9354	LATIN-2 PC-DISP
37719	9357	CYRILLIC PC-DISP
37720	9358	HEBREW PC-DISPLAY
37728	9360	ARABIC PC-DISPLAY
37732	9364	URDU PC-DISPLAY
37733	9365	GREECE PC-DISPLAY
37761	9381	JAPAN SB PC-DISP
37796	93A4	JAPAN PC-DISPLAY
37813	93B5	KOREA KS PC-DISP
37818	93BA	JAPANESE EUC
38064	94B0	Unicode 5.2, UTF-16 BE with IBM PUA
38065	94B1	Unicode 5.2, UTF-16 BE
38066	94B2	Unicode 5.2, UTF-16 LE with IBM PUA
38067	94B3	Unicode 5.2, UTF-16 LE
38072	94B8	Unicode 6.0, UTF-8 with IBM PUA
38073	94B9	Unicode 6.0, UTF-8

CCSID (decimal)	CCSID (hex)	Name
41397	A1B5	FRANCE PC-DISPLAY
41460	A1F4	SWISS EBCDIC
41824	A360	ARABIC PC-DISPLAY
41828	A364	URDU PC-DISPLAY
42160	A4B0	Unicode 6.0, UTF-16 BE with IBM PUA
42161	A4B1	Unicode 6.0, UTF-16 BE
42162	A4B2	Unicode 6.0, UTF-16 LE with IBM PUA
42163	A4B3	Unicode 6.0, UTF-16 LE
45493	B1B5	ITALY PC-DISPLAY
45556	B1F4	SWISS EBCDIC
45920	B360	ARABIC PC-DISPLAY
49589	C1B5	UK PC-DISPLAY
49652	C1F4	BELGIUM EBCDIC
50016	C360	ARABIC PC-DISPLAY
53668	D1A4	Arabic EBCDIC - special
53685	D1B5	USA MS-PC-DATA
53748	D1F4	INTL EBCDIC
54189	D3AD	Special - JAPAN DB PC
54191	D3AF	Special - Japan Open
54289	D411	Special - JAPAN SB PC-DATA
61696	F100	GLOBAL SB EBCDIC
61697	F101	GLOBAL SB PC-DATA
61698	F102	GLOBAL PC-DISPLAY
61699	F103	GLBL ISO-8 ASCII
61700	F104	GLBL ISO-7 ASCII
61710	F10E	GLOBAL USE ASCII
61711	F10F	GLOBAL USE EBCDIC
61712	F110	GLOBAL USE EBCDIC
62251	F32B	Arabic/Latin EBCDIC for OS/390 OE
62337	F381	Special - JAPAN SB PC-DATA
62381	F3AD	Special - JAPAN DB PC
62383	F3AF	Special - JAPAN OPEN
65520	FFF0	Unicode, empty plane

Appendix D. Platform Support of CDRA

This appendix summarizes the CDRA support available on three IBM platforms. The table lists the CDRA defined APIs and indicates on which platforms support is provided. At the bottom of the table, the mechanism providing the support for the platform is given.

	Platform		
API	IBM i	z/OS	AIX
CDRGESP	*	*	*
CDRSCSP	*	*	*
CDRGESE		*	*
CDRGCTL	*	*	*
CDRSMXC	*	*	*
CDRGRDC	*		
CDRGCCN	*		
CDRCVRT	*		
CDRMSCI		*	*
CDRMSCP		*	*
CDRMSCC		*	*
CDRXSRF		*	*

Figure 73. CDRA Implementation by Platform

Support Mechanism:

IBM i – with the operating system
z/OS – with DFSMS/MVS 1.3.0
AIX – with IBM COBOL Set for AIX, IBM PL/1 Set for AIX

A large bullet (*) indicates that support is provided on the platform for the corresponding API. The APIs are documented in “Chapter 5. CDRA Interface Definitions”. “Appendix H. CDRA and IBM I” details restrictions and deviations in the IBM i implementation of the CDRA defined APIs. Likewise “Appendix I. DFSMS/MVS Considerations” details the variances for the DFSMS support of CDRA which is provided on z/OS

Appendix E. Graphic character identification

IBM has an established system to uniquely and uniformly identify and name graphic characters. This system provides for character identification using one of the following graphic character identifiers:

- Graphic Character Global Identifier (GCGID).

  The GCGID identifies graphic characters defined by IBM. The GCGID definition uniformly associates an arbitrary graphic character shape with an eight-character identifier GCGID.

- Graphic Character UCS Identifier (GCUID).

  The GCUID format is used for defining additional characters and sets of characters that (mostly) exist in the Universal Coded Character Set (UCS) defined in ISO/IEC 10646 and Unicode standards and need to be used in IBM resource definitions such as IBM code pages. The format allows all current and future characters from UCS planes 0 through 17 to be described. It also allows for identifying characters and glyphs that are not defined in UCS as well as glyph variants of the unified Han area of UCS.

The GCGID Registry contains the actual GCGIDs, GCUIDs, character glyphs and character names. Direct access to the registry is restricted to IBM employees. Requests for registry information from non-IBMers can be sent to the IBM Globalization Center of Competency (GCoC) at gcoc@ca.ibm.com.

Appendix F: Character sets and code pages

IBM maintains character set and code page repositories. These repositories contain the definitions of the character set and code page resources used and supported by IBM products. Following are two lists. The first contains registered character set number and name while the second contains registers code page number and name. To obtain a copy of any character set or code page resource please contact your IBM representative or email the IBM Globalization Center of Competency (GCoC) at gcoc@ca.ibm.com.

List of Graphic Character Set Global Identifiers

GCSGID	Name
00001	USA WP 96
00025	Latin America, Puerto Rico, Costa Rica WP 96
00029	Germany, Austria WP 96
00041	Italy WP 96
00053	Sweden, Finland WP 96
00055	Norway WP 96
00057	Denmark WP 96
00061	Brazil WP 96
00067	United Kingdom WP 96
00101	USA DP 94
00103	International DP 94 (ASCII)
00218	Greece
00235	Arabic Bilingual - 181
00236	Maghreb/French
00237	Maghreb/French
00265	Austria/Germany DP 94
00269	Belgium DP 94
00273	Brazil DP 94
00277	Canada (French) DP 94
00281	Denmark, Norway DP 94
00285	Finland, Sweden DP 94
00288	France DP 94
00289	France DP 94
00293	Italy DP 94
00301	Portugal DP 94
00309	Latin America (Spanish Speaking) DP 94
00313	United Kingdom DP 94
00317	Austria/Germany F.R., Alternate (3270)
00332	Japan (Katakana)

GCSGID	Name
00337	MLP # 1
00340	Symbols, Set 7
00370	Japanese, Kanji DBCS including 1,880 UDCs
00380	APL/EBCDIC
00640	Syntactic Character Set
00650	Spain DP/TP - 118
00682	France 108
00684	Canada (Bilingual)
00687	Israel (Hebrew)
00695	Euro Country Extended Code Page (ECECP)
00697	Country Extended Code Page (CECP)
00700	Belgium - 167
00810	MS-DOS Greek (Max)
00811	MS-DOS Baltic Rim (Max)
00812	MS-DOS Greek (PC-Data)
00813	MS-DOS Baltic Rim (PC-Data)
00814	MS-DOS Arabic (Transparent ASMO)
00904	Switzerland, French/German - 116
00905	Canadian Bilingual - 124
00908	Switzerland - 131
00919	Personal Computer
00921	France, PC - 103
00922	Germany, PC - 104
00923	Italy, PC - 98
00924	United Kingdom, PC - 97
00925	Greece
00933	Korea
00934	Korean DBCS including 1,880 UDCs
00935	Traditional Chinese DBCS including 6,204 UDCs
00936	People's Republic of China (PRC)
00937	Simplified Chinese DBCS with UDCs
00938	Thailand
00941	Israel (Hebrew)
00948	Russian internet koi8-r
00959	Latin 2 - Multilingual
00960	Cyrillic, Multilingual
00963	Graphic Escape APL2/TN - 138
00965	Latin 3, Multilingual - 182
00966	Thailand - Personal Computer
00968	OCR A
00969	OCR B
00980	MLP - PC

GCSGID	Name
00981	Greece - Personal Computer
00982	Latin 2 - Personal Computer
00983	Latin 3 - Personal Computer
00985	Cyrillic - Personal Computer
00986	Hebrew - PC
00988	PC-display multilingual with euro
00989	PC-data multilingual with euro
00990	Portugal - Personal Computer
00991	Iceland - Personal Computer
00992	Israel - Personal Computer
00993	Canadian French - Personal Computer
00994	Arabic-Personal Computer, Output Imaging-249
00995	Nordic - Personal Computer
00996	Cyrillic, Russian
00997	Urdu - Personal Computer, Output Imaging-253
00998	Greece - Personal Computer
01000	Japanese DBCS without UDCs
01001	Japanese, Kanji DBCS including 4,370 UDCs
01030	Traditional Chinese DBCS without UDCs
01050	Korean DBCS-PC including 1,880 UDCs
01051	DBCS-PC excluding 1880 UDCs, Wards A1-C8, CA-FD
01056	Korean DBCS, KSC 5601 Set including 188 UDCs, Wards A1-FE
01058	Japanese DBCS-EUC, JIS X 0208 Set including 940 UDCs
01059	Japanese DBCS-EUC, JIS X 0212 Set including IBM Selected + 940 UDCs
01060	Japanese DBCS-EUC IBM selected including 940 UDCs
01061	Japanese DBCS-EUC JIS X0208
01062	Japanese DBCS-EUC JIS X0212
01063	Japanese JIS X 0208 - 1978 set (6802 char)
01064	Japanese DBCS-TCP JIS X0208-1983
01066	Japanese DBCS PC for Open Environment
01070	Traditional Chinese DBCS-EUC CNS 11643 Primary Set
01071	Traditional Chinese TBCS-EUC CNS 11643 Remainder Set including 7650 CNS, 11643 secondary set, 325 IBM selected characters + 6,204 UDCs
01073	Traditional Chinese DBCS CNS 11643 Plane 2
01075	Traditional Chinese DBCS-PC equivalent to CNS 11643-1984
01080	Simplified Chinese DBCS equivalent to GB2312-80 (excluding IBM selected & UDC)
01081	Simplified Chinese DBCS-EUC GB 2312-80
01084	GB 18030 Two-byte Code Set excluding Euro sign without UDCs
01085	GB 18030 DBCS excluding Euro sign, 206 Uygur characters, 193 Tibetan characters, 155 Mongolian characters, 1165 Yi Syllables, 50 Yi Radical, 6530 CJK Unified Ideograph Extension-A,

GCSGID	Name
01093	Korean DBCS excluding 623 unique characters and 1880 UDCs and Euro currency sign(Ward 49) and Registered sign(Ward 49) (Common Set with DBCS-PC) Wards 40 to D3
01094	Korean DBCS excluding 1880 UDCs and Euro currency sign(Ward 49) and Registered sign(Ward 49) Wards 40 to D3
01101	Arabic - PC, Data Storage and Interchange - 153
01106	MLP - 222, PC
01114	Belgium - 160
01116	Portugal - 94
01117	Norway - 94
01118	United Kingdom - 94
01119	Spain - 94
01120	Japan 7-Bit Latin
01121	Japan 7-Bit Katakana
01122	Japan Alphanumeric and Katakana
01124	Greece - 180 (3174)
01125	Cyrillic - 180 (3174)
01126	Iceland - 181 (3174)
01128	Latin 2, ROECE - 181 (3174)
01129	France - 105
01132	DCF Release 2 Compatibility
01134	SNA Character Set, Type AR
01135	Denmark - 94
01136	Finland/Sweden - 94
01137	Netherlands - 94
01141	Arabic Bilingual - 136
01142	Arabic Bilingual - 133
01146	Latin-1 Extended, Desk Top Publishing/Windows
01147	Hebrew Character Set A
01150	Cyrillic, Multilingual
01151	Urdu-PC, Data Storage & Interchange - 217
01152	Latin #5, Turkey
01160	Urdu, Output Imaging - 190
01162	Arabic - 189
01164	Japan PC #1 - 181
01165	Korea - Personal Computer - 170
01166	People's Republic of China (PRC)-PC-118
01167	Taiwan - 118
01168	Urdu, Data Storage & Interchange - 154
01169	International Alphabet 5
01170	Japan Intersection (US English & PC)
01171	Japan Intersection (Katakana & PC)

GCSGID	Name
01172	Japanese Extended (EBCDIC/PC Common)
01173	Korean Extended (EBCDIC/PC Common)
01174	Simplified Chinese Ext (EBCDIC/PC Common)
01175	Traditional Chinese Ext (EBCDIC/PC Common)
01176	Thai Extended (EBCDIC/PC Common)
01177	Arabic Extended, Output Imaging
01178	Arabic Extended, Data Storage & Interchange
01185	Simplified Chinese PC Data
01186	Korean PC Display Extended
01187	Japanese PC Display Extended
01188	Simplified Chinese PC Display Extended
01189	Traditional Chinese PC Display Extended
01190	Cyrillic, Russian
01192	Canadian (French) Variant
01193	Switzerland Variant
01195	Spain Variant
01201	H-P Emulation, Roman 8
01212	US - PC Data Character Set
01213	Portugese - PC Data Character Set
01214	Icelandic - PC Data Character Set
01215	Canadian French - PC Data Character Set
01216	Nordic - PC Data Character Set
01217	Hebrew - PC Data Characters Subset #1
01218	Hebrew - PC Data Characters Subset #2
01219	Farsi Bilingual - 190
01220	Country Extended Code Page (CECP)
01224	Korea - Personal Computer
01231	Greece - PC Data Character Set
01232	Latin 2 - PC Data Character Set
01233	Latin 3 - PC Data Character Set
01235	Cyrillic - PC Data Character Set
01237	Latin 5, Turkey - PC Data Character Set
01238	US PC Display Subset - 126
01239	Simplified Chinese Ext (EBCDIC/PC Common)
01240	US PC Display Subset - 129
01244	Arabic - PC Data Character Set
01248	Urdu - PC Data Character Set
01249	Greece - PC Data Character Set
01256	Latin 4
01267	Revised Korean, PC Display
01271	Arabic Character Set, Data Stor & Interchange
01274	Dualcase Printable Graphics of ASN.1

GCSGID	Name
01278	Revised Korean, PC Data
01279	Thai with Low Tone Marks & Ancient Characters
01284	Japan Katakana Extended
01285	Farsi, PC Display
01286	Latin 3
01288	Farsi, PC Data
01290	Multinational Emulation
01291	British NRC Set
01292	Dutch NRC Set
01293	Finnish NRC Set
01295	Norwegian/Danish NRC Set
01296	Swedish NRC Set
01297	Norwegian/Danish NRC Alternate
01302	Latin 3 - Personal Computer
01303	Latin 3 - PC Data Character Set
01305	Baltic - Multilingual
01307	Estonia
01310	Symbols - Personal Computer
01326	Cyrillic, Ukraine
01331	Cyrillic, Ukrainian PC-display
01332	Cyrillic, Ukrainian PC data
01334	Arabic/French PC data
01336	Vietnamese
01337	Arabic/French PC display
01338	Cyrillic, Belorussian, PC display
01339	Cyrillic, Belorussian, PC data
01340	APL (USA)
01341	Lao
01344	Latin 2 ISO8 display
01345	Cyrillic 8-bit, ISO8 display
01346	Baltic ISO8 display
01347	Estonia ISO8 display
01349	Israel (Hebrew)
01350	Arabic Bilingual - 146 (incl Hindi numerics)
01351	Indian Script Code (ISCII-91)
01353	Latin 9
01354	PC Indian Script Code (ISCII-91)
01356	Israel (Hebrew)
01357	Hebrew Character Set A
01358	Hebrew - PC
01359	Hebrew - PC Data Characters Subset #2
01360	Israel - Personal Computer

GCSGID	Name
01361	Hebrew - PC Data Characters Subset #1
01369	PC-display Latin 9
01370	PC-data Latin 9
01371	Greece with euro
01372	PC Data, Greece with euro
01373	PC display, Greece with euro
01375	Latin 2 - Multilingual with euro
01376	PC-data Latin 2 Multilingual with euro
01377	PC-display Latin 2 Multilingual with euro
01378	Turkey Latin 5 with euro
01379	PC-data, Latin 5, Turkey with euro
01380	PC-display, Latin #5, Turkey with euro
01381	Cyrillic, Multilingual with euro
01382	PC-data Cyrillic Multilingual with euro
01383	PC-display Cyrillic Multilingual with euro
01384	PC-data Cyrillic, Russian with euro
01385	PC-display Cyrillic, Russian with euro
01386	PC-data Cyrillic, Belorussian with euro
01387	PC-display Cyrillic, Belorussian with euro
01388	Cyrillic, Ukraine with euro
01389	PC-data, Cyrillic, Ukraine with euro
01390	PC-display Cyrillic, Ukrainian with euro
01391	Estonia with euro
01392	ISO-8 display Estonia with euro
01393	Baltic - Multilingual with euro
01394	ISO-8 display Baltic multilingual with euro
01395	Thai with Low Tone Marks & Ancient Characters
01396	Thai MS Windows
01397	Vietnamese
01398	Japanese Extended (EBCDIC/PC Common)
01399	Traditional Chinese Ext
01400	Windows, Latin 2
01401	Windows, Cyrillic
01402	Windows, Latin 1
01403	Windows, Greek
01404	Windows, Turkish
01405	Windows, Hebrew
01406	Windows, Arabic
01407	Windows, Baltic Rim
01408	Windows, Vietnamese
01410	Windows, Latin 2 + euro
01411	Windows, Cyrillic + euro

GCSGID	Name
01412	Windows, Latin 1 + euro
01413	Windows, Greek + euro
01414	Windows, Turkish + euro
01415	Windows, Hebrew + euro
01416	Windows, Arabic + euro
01417	Windows, Baltic Rim + euro
01418	Windows, Vietnamese + euro
01419	Windows, Hebrew + Euro
01420	Windows, Arabic extended including euro
01421	Windows, Baltic Rim + Euro
01425	Apple Latin 1
01426	Postscript Standard Encoding
01427	Postscript Latin 1
01430	Apple Greek
01431	Apple Turkish
01432	Apple Central European (Latin-2)
01433	Apple Cyrillic
01434	Apple Croatian
01435	Apple Romanian
01436	Apple Icelandic
01437	DEC Greek
01438	DEC Turkish
01441	Cyrillic, Multilingual with euro
01461	Arabic Bilingual
01462	Arabic - PC Data
01463	Arabic-Personal Computer PC-display
01464	Arabic
01465	Arabic extended
01466	Israel (Hebrew)
01467	Devanagari EBCDIC
01468	Arabic - Data Storage & Interchange (Extended)
01469	Devanagari EBCDIC with Rupee Sign
01500	T-Ch/S-Ch Latin 1 PC-display subset
01502	OCR-B plus alt-m, euro, and vertical line
01508	Belarusian/Ukrainian KOI8-RU
01509	Ukrainian KOI8-U
01514	Latin/Greek - ISO
02059	Extended Japanese DBCS, including 12,237 Kanji characters, 2,585 Non-Kanji characters, 6,205 UDCs
02081	GBK Host DBCS incl. 1894UDCs
02084	GB 18030 Two-byte Code Set
02085	GB 18030 Four-byte Code Set

GCSGID	Name
02086	GB 18030 DBCS-Host including 206 Uygur characters and others
02087	GB 18030 DBCS-Host
02092	Extended Japanese DBCS-Host for JIS X0213
02096	Japanese DBCS-EUC, JIS X 0208 Set including 83 NEC selected chars and 940 UDCs
02110	DB Big-5 ext for HKSCS-2001
02111	Host HKSCS-2001, DB
02112	DB Big-5 ext for HKSCS-2004
02113	DB, Host HKSCS-2004
02114	DBCS PC for HKSCS-2008
02115	Host for HKSCS-2008
02131	T-Ch DBCS Fixed CS for additional CNS 11643 subset including 6395 UDCs
03001	ISO 10646 (Unicode 2.0)
03004	ISO 10646 (Unicode 3.0)
03005	Repertoire of Unicode V4.0 BMP (Plane 00)
03006	Repertoire of Unicode V4.0 SMP (Plane 01)
03007	Repertoire of Unicode V4.0 SIP (Plane 02)
03008	Repertoire of Unicode V4.0 SSP (Plane 0E)
03009	Repertoire of Unicode V 4.1 BMP (Plane 00)
03010	Repertoire of Unicode V 4.1 SMP (Plane 01)
03011	Repertoire of Unicode V 5.0 BMP (Plane 00)
03012	Repertoire of Unicode V 5.0 SMP (Plane 01)
03013	Repertoire of Unicode V 5.1 BMP (Plane 00)
03014	Repertoire of Unicode V 5.1 SMP (Plane 01)
03015	Repertoire of Unicode V 5.2 BMP (Plane 00)
03016	Repertoire of Unicode V 5.2 SMP (Plane 01)
03017	Repertoire of Unicode V5.2 SIP (Plane 02)
03018	Repertoire of Unicode V 6.0 BMP (Plane 00)
03019	Repertoire of Unicode V 6.0 SMP (Plane 01)
03020	Repertoire of Unicode V6.0 SIP (Plane 02)
03095	IBM Advanced Function Printing (AFP) PUA No. 1
03096	Repertoire of Unicode V4.0 PUP-15 (Plane 0F)
03097	Repertoire of Unicode V4.0 PUP-16 (Plane 10)
03098	Repertoire of Unicode V4.0 PUA of UCS-BMP (Generic UDC)
03099	IBM default PUA
65520	Empty Character Set
65535	Growing Character Set

List of Code Page Global Identifiers

CPGID	Name
00037	USA/Canada - CECP
00256	International #1
00259	Symbols, Set 7
00273	Germany F.R./Austria - CECP
00274	Old Belgium Code Page
00275	Brazil - CECP
00276	Canada (French) - 94
00277	Denmark, Norway - CECP
00278	Finland, Sweden - CECP
00280	Italy - CECP
00281	Japan (Latin) - CECP
00282	Portugal - CECP
00284	Spain/Latin America - CECP
00285	United Kingdom - CECP
00286	Austria/Germany F.R., Alternate (3270)
00290	Japanese (Katakana) Extended
00293	APL (USA)
00297	France - CECP
00300	Japan (Kanji) - Host, DBCS
00301	Japan (Kanji) - PC, DBCS
00310	Graphic Escape APL/TN
00367	ASCII
00420	Arabic Bilingual
00421	Maghreb/French
00423	Greece - 183
00424	Israel (Hebrew)
00425	Arabic/Latin for OS/390 Open Edition
00437	Personal Computer
00500	International #5
00720	MS DOS Arabic (Transparent ASMO)
00737	MS DOS Greek
00775	MS DOS Baltic Rim
00803	Hebrew Character Set A
00806	PC Indian Script Code (ISCII-91)
00808	PC Data, Cyrillic, Russian with euro
00813	Greece ISO 8859-7
00819	ISO/ANSI Multilingual
00833	Korean Extended
00834	Korean Hangul - Host, DBCS with UDCs
00835	Traditional Chinese DBCS - Host

CPGID	Name
00836	Simplified Chinese Extended
00837	Simplified Chinese DBCS-HOST
00838	Thai with Low Tone Marks & Ancient Characters
00848	PC, Cyrillic, Ukrainian with euro
00849	PC Data, Cyrillic, Belorussian with euro
00850	Personal Computer - Multilingual Page
00851	Greece - Personal Computer
00852	Latin 2 - Personal Computer
00853	Latin 3 - Personal Computer
00855	Cyrillic - Personal Computer
00856	Hebrew - Personal Computer
00857	Latin #5, Turkey - Personal Computer
00858	Personal Computer - Multilingual with euro
00859	PC Latin 9
00860	Portugal - Personal Computer
00861	Iceland - Personal Computer
00862	Israel - Personal Computer
00863	Canadian French - Personal Computer
00864	Arabic - Personal Computer
00865	Nordic - Personal Computer
00866	PC Data, Cyrillic, Russian
00867	Israel - Personal Computer
00868	Urdu - Personal Computer
00869	Greece - Personal Computer
00870	Latin 2 - EBCDIC Multilingual
00871	Iceland - CECP
00872	Cyrillic - PC with euro
00874	Thai with Low Tone Marks & Ancient Chars - PC
00875	Greece
00878	Russian internet koi8-r
00880	Cyrillic, Multilingual
00891	Korea - Personal Computer
00892	EBCDIC, OCR A
00893	EBCDIC, OCR B
00895	Japan 7-Bit Latin
00896	Japan 7-Bit Katakana Extended
00897	Japan PC #1
00899	Symbol - Personal Computer
00901	PC Baltic Multi with Euro
00902	8-bit Estonia with Euro
00903	People's Republic of China (PRC)-PC
00904	Taiwan - Personal Computer

CPGID	Name
00905	Latin 3 - EBCDIC
00912	Latin 2 - ISO
00913	Latin 3 - ISO
00914	Latin 4
00915	Cyrillic, 8-Bit
00916	Hebrew (Latin)
00918	Urdu Bilingual
00920	Latin #5 - Turkey
00921	Baltic - Multilingual, superset of ISO 8859-13
00922	Estonia, similar to ISO 8859-1
00923	Latin 9
00924	Latin 9 EBCDIC
00926	Korean PC Data Double-Byte incl. 1880 UDC
00927	T-Ch PC Data Double-Byte incl. 6204 UDC
00928	S-Ch PC Data Double-Byte incl. 1880 UDC
00941	Japanese DBCS PC for Open environment
00947	Pure DBCS for Big-5
00951	Korean DBCS-PC (10104 characters)
00952	Japanese EUC for JIS X 0208 including 83 NEC selected chars + 940 UDC
00953	Japanese EUC for JIS X 0212 + IBM Select + UDC
00955	Japanese TCP, JIS X0208-1978
00960	Traditional Chinese DBCS-EUC SICGCC Primary set (1st plane)
00961	Traditional Chinese TBCS-EUC SICGCC Full set + IBM Select + UDC
00963	T-Ch TCP, CNS 11643 plane 2 only
00971	Korean EUC, DBCS EUC (G1, KSC 5601)
01002	DCF Release 2 Compatibility
01004	Latin-1 Extended, Desk Top Publishing/Windows
01006	Urdu, 8-Bit
01008	Arabic 8-Bit ISO/ASCII
01009	ISO IRV
01010	7-Bit France
01011	7-Bit Germany F.R.
01012	7-Bit Italy
01013	7-Bit United Kingdom
01014	7-Bit Spain
01015	7-Bit Portugal
01016	7-Bit Norway
01017	7-Bit Denmark
01018	7-Bit Finland/Sweden
01019	7-Bit Netherlands
01020	Canadian (French) Variant
01021	Switzerland Variant

CPGID	Name
01023	Spain Variant
01025	Cyrillic, Multilingual
01026	Latin #5 - Turkey
01027	Japanese (Latin) Extended
01040	Korean Extended - Personal Computer
01041	Japanese Extended - Personal Computer
01042	Simplified Chinese Extended - PC
01043	Traditional Chinese Extended - PC
01046	Arabic Extended-Euro
01047	Latin 1/Open Systems
01051	H-P Emulation, Roman 8
01070	USA/Canada - CECP
01079	Spain/Latin America - CECP
01081	France - CECP
01084	International #5
01088	Revised Korean - Personal Computer
01089	Arabic Code Page, Data Storage & Interchange
01097	Farsi Bilingual - EBCDIC
01098	Farsi - Personal Computer
01100	Multinational Emulation
01101	British NRC Set
01102	Dutch NRC Set
01103	Finnish NRC Set
01104	French NRC Set
01105	Norwegian/Danish NRC Set
01106	Swedish NRC Set
01107	Norwegian/Danish NRC Alternate
01112	Baltic - Multilingual, EBCDIC
01114	Taiwan - Personal Computer
01115	People's Republic of China (PRC)-PC
01122	Estonia, EBCDIC
01123	Cyrillic, Ukraine
01124	Cyrillic, Ukraine
01125	PC, Cyrillic, Ukrainian
01126	Korean - Personal Computer for Windows
01127	Arabic/French - Personal Computer
01129	Vietnamese ISO-8
01130	Vietnamese EBCDIC
01131	PC Data, Cyrillic, Belorussian
01132	Lao EBCDIC
01133	Lao ISO-8
01137	Devanagari EBCDIC

CPGID	Name
01140	USA, Canada, etc. ECECP
01141	Austria, Germany ECECP
01142	Denmark, Norway ECECP
01143	Finland, Sweden ECECP
01144	Italy ECECP
01145	Spain, Latin America (Spanish) ECECP
01146	UK ECECP
01147	France ECECP
01148	International ECECP
01149	Iceland ECECP
01153	EBCDIC Latin 2 Multilingual with euro
01154	EBCDIC Cyrillic, Multilingual with euro
01155	EBCDIC Turkey with euro
01156	EBCDIC Baltic Multi with euro
01157	EBCDIC Estonia with euro
01158	EBCDIC Cyrillic, Ukraine with euro
01159	T-Chinese EBCDIC
01160	Thai with Low Tone Marks & Ancient Characters
01161	Thai with Low Tone Marks & Ancient Chars - PC
01162	Thai MS Windows
01163	Vietnamese ISO-8 with euro
01164	Vietnamese EBCDIC with euro
01165	Latin 2 EBCDIC/Open Systems
01166	EBCDIC Cyrillic, Multilingual with euro
01167	Belarusian/Ukrainian KOI8-RU
01168	Ukrainian KOI8-U
01250	Windows, Latin 2
01251	Windows, Cyrillic
01252	Windows, Latin 1
01253	Windows, Greek
01254	Windows, Turkish
01255	Windows, Hebrew
01256	Windows, Arabic
01257	Windows, Baltic Rim
01258	Windows, Vietnamese
01275	Apple, Latin 1
01276	Adobe (PostScript) Standard Encoding
01277	Adobe (PostScript) Latin 1
01280	Apple Greek
01281	Apple Turkish
01282	Apple Central European
01283	Apple Cyrillic

CPGID	Name
01284	Apple, Croatian
01285	Apple, Romanian
01286	Apple, Icelandic
01287	DEC Greek 8-Bit
01288	DEC Turkish 8-Bit
01351	DBCS-PC, including 940 HP UDCs, Japanese
01362	Korean Hangul - PC, DBCS with UDCs
01372	MS T-Chinese Big-5 (Special for DB2)
01374	DB Big-5 extension for HKSCS
01376	Traditional Chinese DBCS-Host extension for HKSCS
01380	Simplified Chinese GB PC-DATA
01382	Simplified Chinese EUC
01385	Simplified Chinese 2 Byte, growing CS for GB18030, also used for GBK PC-DATA
01391	Simplified Chinese 4 Byte, growing CS for GB18030
01393	Shift_JISX0213 DBCS
01400	ISO 10646 UCS-BMP (Based on Unicode V6.0)
01401	ISO 10646 UCS-SMP (Based on Unicode V6.0)
01402	ISO 10646 UCS-SIP (Based on Unicode V6.0)
01414	ISO 10646 UCS-SSP (Based on Unicode 4.0)
01445	IBM AFP PUA No. 1
01446	ISO 10646 UCS-PUP15 (Based on Unicode 4.0)
01447	ISO 10646 UCS-PUP16 (Based on Unicode 4.0)
01448	UCS-BMP (Generic UDC)
01449	IBM default PUA
65520	Empty Unicode Plane

Appendix G. Control character mappings

This appendix contains the predefined default control character mappings used by CDRA when creating single-byte to single-byte round trip conversion tables.

It is important to note that these mappings are applied only after the matched graphic characters and matched control mnemonics have been mapped. If an output code point found in these tables has already been mapped, the corresponding input code point is added to the list of unmapped code points and mapped accordingly.

The EBCDIC control codes are defined in the IBM Corporate Standard, C-S 3-3220-002. An excerpt from the standard can be found in Appendix L: EBCDIC control definitions.

- IFS is X'1A' instead of X'1C' in ASCII.
- DEL is X'1C' instead of X'7F' in ASCII.
- SUB is X'7F' instead of X'1A' in ASCII.

These exceptions arise from the use of X'1A' as the End of File in the IBM-PC operating systems, making X'1A' unsuitable as SUB.

Also note the use of X'14' and X'15' as graphic characters under exception conditions as described in the “Exceptions” section of Chapter 6.

EBCDIC to IBM-PC

EBCDIC			IBM-PC		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	EXT	End of Text
04	SEL	Select	DC	graphic (1)	
05	HT	Horizontal Tab	09	HT	Horizontal/ Character Tabulation

06	RNL	Required New Line	C3	graphic (1)	
07	DEL	Delete	1C	DEL	Delete (2)
08	GE	Graphic Escape	CA	graphic (1)	
09	SPS	Superscript	B2	graphic (1)	
0A	RPT	Repeat	D5	graphic (1)	
0B	VT	Vertical Tab	0B	VT	Vertical/Line Tabulation
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO	Shift Out	0E	SO/LS1 (3)	Shift Out/Locking Shift 1
0F	SI	Shift In	0F	SI/LS0 (3)	Shift In/Locking Shift 0
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1	Device Control 1	11	DC1/XON (3)	Device Control 1/XON
12	DC2	Device Control 2	12	DC2	Device Control 2
13	DC3	Device Control 3	13	DC3/XOFF (3)	Device Control 3/XOFF
14	RES/ENP	Restore/Enable Presentation	DB	graphic (1)	
15	NL	New Line	DA	graphic (1)	
16	BS	Backspace	08	BS	Backspace
17	POC	Program Operator Communication	C1	graphic (1)	
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	UBS	Unit Backspace Hex	C8	graphic (1)	
1B	CU1	Customer Use 1	F2	graphic (1)	

1C	IFS	Interchange File Separator	1A	IS4/FS (3)	Information Separator Four (2) / File Separator
1D	IGS	Interchange Group Separator	1D	IS3/GS (3)	Information Separator Three / Group Separator
1E	IRS	Interchange Record Separator	1E	IS2/RS (3)	Information Separator Two / Record Separator
1F	IUS/ITB (3)	Interchange Unit Separator/ Intermediate Transmission Block	1F	IS1/US (3)	Information Separator One / Unit Separator
20	DS	Digit Select	C4	graphic (1)	
21	SOS. (4)	Start of Significance	B3	graphic (1)	
22	FS	Field Separator	C0	graphic (1)	
23	WUS	Word Underscore	D9	graphic (1)	
24	BYP/INP	Bypass or Inhibit Presentation	BF	graphic (1)	
25	LF	Line Feed	0A	LF	Line Feed
26	ETB	End of Transmission Block	17	ETB	End of Transmission Block
27	ESC	Escape	1B	ESC	Escape
28	SA	Set Attribute	B4	graphic (1)	
29	SFE	Start Field Extended	C2	graphic (1)	
2A	SM/SW (3)	Set Mode/Switch	C5	graphic (1)	
2B	CSP	Control Sequence Prefix	B0	graphic (1)	
2C	MFA	Modify Field Attribute	B1	graphic (1)	

2D	ENQ	Enquiry	05	ENQ	Enquiry
2E	ACK	Acknowledge	06	ACK	Acknowledge
2F	BEL	Bell	07	BEL	Bell
30	xxx	Reserved	CD	graphic (1)	
31	xxx	Reserved	BA	graphic (1)	
32	SYN	Synchronous Idle	16	SYN	Synchronous Idle
33	IR	Index Return	BC	graphic (1)	
34	PP	Presentation Position	BB	graphic (1)	
35	TRN	Transparent	C9	graphic (1)	
36	NBS	Numeric Backspace	CC	graphic (1)	
37	EOT	End of Transmission	04	EOT	End of Transmission
38	SBS	Subscript	B9	graphic (1)	
39	IT	Indent Tab	CB	graphic (1)	
3A	RFF	Required Form Feed	CE	graphic (1)	
3B	CU3	Customer Use 3	DF	graphic (1)	
3C	DC4	Device Control 4	14	DC4	Device Control 4
3D	NAK	Negative Acknowledge	15	NAK	Negative Acknowledge
3E	xxx	Reserved	FE	graphic (1)	
3F	SUB	Substitute	7F	SUB	Substitute (2)
FF	EO	Eight Ones	9F	graphic (1)	

Figure 74. Control Character Mapping - SBCS EBCDIC to IBM-PC

IBM-PC to EBCDIC

IBM-PC			EBCDIC		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	EXT	End of Text
04	EOT	End of Transmission	37	EOT	End of Transmission
05	ENQ	Enquiry	2D	ENQ	Enquiry
06	ACK	Acknowledge	2E	ACK	Acknowledge
07	BEL	Bell	2F	BEL	Bell
08	BS	Backspace	16	BS	Backspace
09	HT	Horizontal/ Character Tabulation	05	HT	Horizontal Tab
0A	LF	Line Feed	25	LF	Line Feed
0B	VT	Vertical/Line Tabulation	0B	VT	Vertical Tab
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO/LS1 (3)	Shift Out/Locking Shift 1	0E	SO	Shift Out
0F	SI/LS0 (3)	Shift In/Locking Shift 0	0F	SI	Shift In
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1/XON (3)	Device Control 1/XON	11	DC1	Device Control 1
12	DC2	Device Control 2	12	DC2	Device Control 2
13	DC3/XOFF (3)	Device Control 3/XOFF	13	DC3	Device Control 3
14	DC4	Device Control 4	3C	DC4	Device Control 4
15	NAK	Negative Acknowledge	3D	NAK	Negative Acknowledge
16	SYN	Synchronous Idle	32	SYN	Synchronous Idle
17	ETB	End of Transmission Block	26	ETB	End of Transmission Block
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	IS4/FS (3)	Information Separator Four (2)/ File Separator	1C	IFS	Interchange File Separator
1B	ESC	Escape	27	ESC	Escape
1C	DEL	Delete (2)	07	DEL	Delete
1D	IS3/GS (3)	Information Separator Three / Group Separator	1D	IGS	Interchange Group Separator
1E	IS2/RS (3)	Information Separator Two / Record Separator	1E	IRS	Interchange Record Separator

IBM-PC			EBCDIC		
1F	IS1/US (3)	Information Separator One / Unit Separator	1F	IUS/ITB (3)	Interchange Unit Separator/ Intermediate Transmission Block
7F	SUB	Substitute (2)	3F	SUB	Substitute

Figure 75. Control Character Mapping - SBCS EBCDIC to IBM-PC

ISO-8 to IBM-PC

ISO-8			IBM-PC		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	ETX	End of Text
04	EOT	End of Transmission	04	EOT	End of Transmission
05	ENQ	Enquiry	05	ENQ	Enquiry
06	ACK	Acknowledge	06	ACK	Acknowledge
07	BEL	Bell	07	BEL	Bell
08	BS	Backspace	08	BS	Backspace
09	HT	Horizontal / Character Tabulation	09	HT	Horizontal / Character Tabulation
0A	LF	Line Feed	0A	LF	Line Feed
0B	VT	Vertical/ Line Tabulation	0B	VT	Vertical/ Line Tabulation
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO/LS1 (3)	Shift Out/Locking Shift 1	0E	SO/LS1 (3)	Shift Out/Locking Shift 0
0F	SI/LS0 (3)	Shift In/Locking Shift 0	0F	SI/LS0 (3)	Shift In/Locking Shift 0
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1/XON (3)	Device Control 1/XON	11	DC1/XON (3)	Device Control 1/XON
12	DC2	Device Control 2	12	DC2	Device Control 2
13	DC3/XOFF (3)	Device Control 3/XOFF	13	DC3/XOFF (3)	Device Control 3/XOFF
14	DC4	Device Control 4	14	DC4	Device Control 4
15	NAK	Negative Acknowledge	15	NAK	Negative Acknowledge
16	SYN	Synchronous Idle	16	SYN	Synchronous Idle
17	ETB	End of Transmission Block	17	ETB	End of Transmission Block
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	SUB	Substitute	7F	SUB	Substitute (2)
1B	ESC	Escape	1B	ESC	Escape
1C	IS4/FS (3)	Information Separator Four / File Separator	1A	IS4/FS (3)	Information Separator Four (2)

ISO-8			IBM-PC		
1D	IS3/GS (3)	Information Separator Three / Group Separator	1D	IS3/GS (3)	Information Separator Three / Group Separator
1E	IS2/RS (3)	Information Separator Two / Record Separator	1E	IS2/RS (3)	Information Separator Two / Record Separator
1F	IS1/US (3)	Information Separator One / Unit Separator	1F	IS1/US (3)	Information Separator One / Unit Separator
7F	DEL	Delete	1C	DEL	Delete (2)
80	xxx	Reserved	BA	graphic (1)	
81	xxx	Reserved	CD	graphic (1)	
82	BPH	Break Permitted Here	C9	graphic (1)	
83	NBH	No Break Here	BB	graphic (1)	
84	IND	Index	C8	graphic (1)	
85	NEL	Next Line	BC	graphic (1)	
86	SSA	Start of Selected Area	CC	graphic (1)	
87	ESA	End of Selected Area	B9	graphic (1)	
88	HTS	Character Tabulation Set	CB	graphic (1)	
89	HTJ	Character Tabulation with Justification	CA	graphic (1)	
8A	VTS	Line Tabulation Set	CE	graphic (1)	
8B	PLD	Partial Line Forward	DF	graphic (1)	
8C	PLU	Partial Line Backward	DC	graphic (1)	
8D	RI	Reverse Line Feed	DB	graphic (1)	
8E	SS2	Single Shift Two	FE	graphic (1)	
8F	SS3	Single Shift Three	F2	graphic (1)	
90	DCS	Device Control String	B3	graphic (1)	
91	PU1	Private Use One	C4	graphic (1)	
92	PU2	Private Use Two	DA	graphic (1)	
93	STS	Set Transmit State	BF	graphic (1)	
94	CCH	Cancel Character	C0	graphic (1)	
95	MW	Message Waiting	D9	graphic (1)	
96	SPA	Start of Guarded Area	C3	graphic (1)	
97	EPA	End of Guarded Area	B4	graphic (1)	
98	SOS	Start of String	C2	graphic (1)	
99	xxx	Reserved	C1	graphic (1)	
9A	SCI	Single Character Introducer	C5	graphic (1)	

ISO-8			IBM-PC		
9B	CSI	Control Sequence Introducer	B0	graphic (1)	
9C	ST	String Terminator	B1	graphic (1)	
9D	OSC	Operating System Command	B2	graphic (1)	
9E	PM	Privacy Message	D5	graphic (1)	
9F	APC	Application Program Command	9F	graphic (1)	

Figure 76. Control Character Mapping – SBCS ISO-8 to IBM-PC

IBM-PC to ISO-8

IBM-PC			ISO-8		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	ETX	End of Text
04	EOT	End of Transmission	04	EOT	End of Transmission
05	ENQ	Enquiry	05	ENQ	Enquiry
06	ACK	Acknowledge	06	ACK	Acknowledge
07	BEL	Bell	07	BEL	Bell
08	BS	Backspace	08	BS	Backspace
09	HT	Horizontal/ Character Tabulation	09	HT	Horizontal/ Character Tabulation
0A	LF	Line Feed	0A	LF	Line Feed
0B	VT	Vertical/Line Tabulation	0B	VT	Vertical/ Line Tabulation
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO/LS1 (3)	Shift Out/Locking Shift 1	0E	SO/LS0 (3)	Shift Out/Locking Shift 0
0F	SI/LS0 (3)	Shift In/Locking Shift 0	0F	SI/LS0 (3)	Shift In/Locking Shift 0
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1/XON (3)	Device Control 1/XON	11	DC1/XON (3)	Device Control 1
12	DC2	Device Control Two	12	DC2	Device Control Two
13	DC3/XOFF (3)	Device Control 3/XOFF	13	DC3/XOFF (3)	Device Control 3/XOFF
14	DC4	Device Control Four	14	DC4	Device Control Four
15	NAK	Negative Acknowledge	15	NAK	Negative Acknowledge
16	SYN	Synchronous Idle	16	SYN	Synchronous Idle
17	ETB	End of Transmission Block	17	ETB	End of Transmission Block
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	IS4/FS (3)	Information Separator Four / File Separator (2)	1C	IS4/FS (3)	Information Separator Four / File Separator
1B	ESC	Escape	1B	ESC	Escape
1C	DEL	Delete (2)	7F	DEL	Delete

IBM-PC			ISO-8		
1D	IS3/GS (3)	Information Separator Three / Group Separator	1D	IS3/GS (3)	Information Separator Three / Group Separator
1E	IS2/RS (3)	Information Separator Two / Record Separator	1E	IS2/RS (3)	Information Separator Two / Record Separator
1F	IS1/US (3)	Information Separator One / Unit Separator	1F	IS1/US (3)	Information Separator One / Unit Separator
7F	SUB	Substitute (2)	1A	SUB	Substitute

Figure 77. Control Character Mapping - SBCS ISO-8 to IBM-PC

EBCDIC to ISO-8

EBCDIC			ISO-8		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	ETX	End of Text
04	SEL	Select	9C	ST	String Terminator
05	HT	Horizontal Tab	09	HT	Horizontal/ Character Tabulation
06	RNL	Required New Line	86	SSA	Start of Selected Area
07	DEL	Delete	7F	DEL	Delete
08	GE	Graphic Escape	97	EPA	End of Guarded Area
09	SPS	Superscript	8D	RI	Reverse Line Feed
0A	RPT	Repeat	8E	SS2	Single Shift Two
0B	VT	Vertical Tab	0B	VT	Vertical/Line Tabulation
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO	Shift Out	0F	SO/LS1 (3)	Shift Out/Locking Shift 1
0F	SI	Shift In	0F	SI/LS0 (3)	Shift In/Locking Shift 0
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1	Device Control 1	11	DC1/XON (3)	Device Control 1/XON
12	DC2	Device Control 2	12	DC2	Device Control 2
13	DC3	Device Control 3	13	DC3/XOFF (3)	Device Control 3/XOFF
14	RES/ENP	Restore/Enable Presentation	9D	OSC	Operating System Command
15	NL	New Line	85	NEL	Next Line
16	BS	Backspace	08	BS	Backspace
17	POC	Program Operator Communication	87	ESA	End of Selected Area
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	UBS	Unit Backspace Hex	92	PU2	Private Use Two
1B	CU1	Customer Use 1	8F	SS3	Single Shift Three

EBCDIC			ISO-8		
1C	IFS	Interchange File Separator	1C	IS4/FS (3)	Information Separator Four / File Separator
1D	IGS	Interchange Group Separator	1D	IS3/GS (3)	Information Separator Three / Group Separator
1E	IRS	Interchange Record Separator	1E	IS2/RS (3)	Information Separator Two / Record Separator
1F	IUS/ITB	Interchange Unit Separator/ Intermediate Transmission Block	1F	IS1/US (3)	Information Separator One / Unit Separator
20	DS	Digit Select	80	xxx	Reserved
21	SOS. (4)	Start of Significance	81	xxx	Reserved
22	FS	Field Separator	82	BPH	Break Permitted Here
23	WUS	Word Underscore	83	NBH	No Break Here
24	BYP/INP	Bypass or Inhibit Presentation	84	IND	Index
25	LF	Line Feed	0A	LF	Line Feed
26	ETB	End of Transmission Block	17	ETB	End of Transmission Block
27	ESC	Escape	1B	ESC	Escape
28	SA	Set Attribute	88	HTS	Horizontal/ Character Tabulation Set
29	SFE	Start Field Extended	89	HTJ	Character Tabulation with Justification
2A	SM/SW (3)	Set Mode/Switch	8A	VTS	Vertical/ Line Tabulation Set
2B	CSP	Control Sequence Prefix	8B	PLD	Partial Line Forward
2C	MFA	Modify Field Attribute	8C	PLU	Partial Line Backward
2D	ENQ	Enquiry	05	ENQ	Enquiry
2E	ACK	Acknowledge	06	ACK	Acknowledge
2F	BEL	Bell	07	BEL	Bell
30	xxx	Reserved	90	DCS	Device Control String
31	xxx	Reserved	91	PU1	Private Use One
32	SYN	Synchronous Idle	16	SYN	Synchronous Idle
33	IR	Index Return	93	STS	Set Transmit State
34	PP	Presentation Position	94	CCH	Cancel Character
35	TRN	Transparent	95	MW	Message Waiting

EBCDIC			ISO-8		
36	NBS	Numeric Backspace	96	SPA	Start of Guarded Area
37	EOT	End of Transmission	04	EOT	End of Transmission
38	SBS	Subscript	98	SOS	Start of String
39	IT	Indent Tab	99	xxx	Reserved
3A	RFF	Required Form Feed	9A	SCI	Single Character Introducer
3B	CU3	Customer Use 3	9B	CSI	Control Sequence Introducer
3C	DC4	Device Control 4	14	DC4	Device Control 4
3D	NAK	Negative Acknowledge	15	NAK	Negative Acknowledge
3E	xxx	Reserved	9E	PM	Privacy Message
3F	SUB	Substitute	1A	SUB	Substitute
FF	EO	Eight Ones	9F	APC	Application Program Command (5)

Figure 78. Control Character Mapping - SBCS EBCDIC to ISO-8

ISO-8 to EBCDIC

ISO-8			EBCDIC		
Hex	Abbreviation	Character Name	Hex	Abbreviation	Character Name
00	NUL	Null	00	NUL	Null
01	SOH	Start of Heading	01	SOH	Start of Heading
02	STX	Start of Text	02	STX	Start of Text
03	ETX	End of Text	03	ETX	End of Text
04	EOT	End of Transmission	37	EOT	End of Transmission
05	ENQ	Enquiry	2D	ENQ	Enquiry
06	ACK	Acknowledge	2E	ACK	Acknowledge
07	BEL	Bell	2F	BEL	Bell
08	BS	Backspace	16	BS	Backspace
09	HT	Horizontal/ Character Tabulation	05	HT	Horizontal Tab
0A	LF	Line Feed	25	LF	Line Feed
0B	VT	Vertical/Line Tabulation	0B	VT	Vertical Tab
0C	FF	Form Feed	0C	FF	Form Feed
0D	CR	Carriage Return	0D	CR	Carriage Return
0E	SO/SL1 (3)	Shift Out/Locking Shift 1	0E	SO	Shift Out

ISO-8			EBCDIC		
0F	SI/LS0 (3)	Shift In/Locking Shift 0	0F	SI	Shift In
10	DLE	Data Link Escape	10	DLE	Data Link Escape
11	DC1/XON (3)	Device Control 1/XON	11	DC1	Device Control 1
12	DC2	Device Control 2	12	DC2	Device Control 2
13	DC3/XOFF (3)	Device Control 3/XOFF	13	DC3	Device Control 3
14	DC4	Device Control 4	3C	DC4	Device Control 4
15	NAK	Negative Acknowledge	3D	NAK	Negative Acknowledge
16	SYN	Synchronous Idle	32	SYN	Synchronous Idle
17	ETB	End of Transmission Block	26	ETB	End of Transmission Block
18	CAN	Cancel	18	CAN	Cancel
19	EM	End of Medium	19	EM	End of Medium
1A	SUB	Substitute	3F	SUB	Substitute
1B	ESC	Escape	27	ESC	Escape
1C	IS4/FS (3)	Information Separator Four / File Separator	1C	IFS	Interchange File Separator
1D	IS3/GS (3)	Information Separator Three / Group Separator	1D	IGS	Interchange Group Separator
1E	IS2/RS (3)	Information Separator Two / Record Separator	1E	IRS	Interchange Record Separator
1F	IS1/US (3)	Information Separator One / Unit Separator	1F	IUS/ITB (3)	Interchange Unit Separator/ Intermediate Transmission Block
7F	DEL	Delete	07	DEL	Delete
80	xxx	Reserved	20	DS	Digit Select
81	xxx	Reserved	21	SOS. (4)	Start of Significance
82	BPH	Break Permitted Here	22	FS	Field Separator
83	NBH	No Break Here	23	WUS	Word Underscore
25	IND	Index	24	BYP/INP	Bypass or Inhibit Presentation
85	NEL	Next Line	15	NL	New Line
86	SSA	Start of Selected Area	06	RNL	Required New Line
87	ESA	End of Selected Area	17	POC	Program Operator Communication

ISO-8			EBCDIC		
88	HTS	Horizontal/ Character Tabulation Set	28	SA	Set Attribute
89	HTJ	Character Tabulation with Justification	29	SFE	Start Field Extended
8A	VTS	Vertical/Line Tabulation Set	2A	SM/SW	Set Mode/Switch
8B	PLD	Partial Line Forward	2B	CSP	Control Sequence Prefix
8C	PLU	Partial Line Backward	2C	MFA	Modify Field Attribute
8D	RI	Reverse Line Feed	09	SPS	Superscript
8E	SS2	Single Shift Two	0A	RPT	Repeat
8F	SS3	Single Shift Three	1B	CU1	Customer Use One
90	DCS	Device Control String	30	xxx	Reserved
91	PU1	Private Use One	31	xxx	Reserved
92	PU2	Private Use Two	1A	UBS	Unit Backspace Hex
93	STS	Set Transmit State	33	IR	Index Return
94	CCH	Cancel Character	34	PP	Presentation Position
95	MW	Message Waiting	35	TRN	Transparent
96	SPA	Start of Guarded Area	36	NBS	Numeric Backspace
97	EPA	End of Guarded Area	08	GE	Graphic Escape
98	SOS	Start of String	38	SBS	Subscript
99	xxx	Reserved	39	IT	Indent Tab
9A	SCI	Single Character Introducer	3A	RFF	Required Form Feed
9B	CSI	Control Sequence Introducer	3B	CU3	Customer Use 3
9C	ST	String Terminator	04	SEL	Select
9D	OSC	Operating System Command	14	RES/ENP	Restore / Enable Presentation
9E	PM	Privacy Message	3E	xxx	Reserved
9F	APC	Application Program Command	FF	EO	Eight Ones

Figure 79. Control Character Mapping - SBCS ISO-8 to EBCDIC

Footnotes:

(1) These code points are in the graphic character space for IBM-PC code pages. The actual graphic characters vary from code page to code page. These code points are used for

mapping control code points for consistency. (Note that a graphic character match will override the control character mapping.)

(2) These control points do not follow the definitions of ASCII in ANSI X3.4.

(3) Two mnemonics are specified when the standard has changed over time or the control code may be used for different purposes depending on the context of use. Both mnemonics are acceptable abbreviations.

(4) The mnemonic for the Start of Significance control character in EBCDIC has been modified to include a dot (.) at the end (SOS.). This has been done to distinguish it from the SOS mnemonic used in ISO-8 for the Start of String control character. The dot does not alter the property of the control in any way.

(5) Prior to 1986, ISO-8 X'9F' (APC) mapped to EBCDIC X'E1'. This control code point is a graphic code point. It was previously used as numeric space character in many EBCDIC SBCS coded character sets, and with the latest revised CECPs, the numeric space character has been replaced with DIVISION SYMBOL. The map shown in here is to EBCDIC Eight Ones control, which is used as a filler character.

Appendix H. CDRA and IBM i (formerly OS/400)

IBM i, formerly OS/400, has supported CDRA for many years. Information on system specific implementation of APIs can be found in the IBM Knowledge Center.

Appendix I. DFSMS/MVS Considerations

CDRA API Considerations for DFSMS/MVS

This appendix provides additional information necessary to install and use the CDRA APIs included in the DFSMS/MVS product. APIs are found in Chapter 5. CDRA Interface Definitions; they are not duplicated in this appendix.

The CDRA support provided will function in the following environments:

- DFSMS/MVS 1.3.0
- All levels of MVS/SP that are consistent with DFSMS 1.3.0, which are currently MVS/ESA 4.3, 5.1 and 5.2.
- The Language Environment/370 (LE/370) must be available for both the installation and execution of the CDRA services code.

With MVS/ESA SP 5.2 or higher, these requirements are met if the C/C++ Language support feature of MVS/ESA SP or the LE for MVS and VM program product is used.

With MVS/ESA SP 5.1, these requirements are met if the C/C++ Language support feature of MVS/ESA SP or the LE/370 program product is used.

With MVS/ESA SP 4.3, these requirements are met if the LE/370 program product is used.

Completing the Character Conversion Services Installation

Tailor the CDRAINIT member of SYS1.SAMPliB as required by your installation, then run it to define the work data sets used by the various conversion routines.

The following data sets will be created as a result of running the CDRAINIT member:

SYS1.CDRASRVT
  The code page resource tables
SYS1.CDRSRVCI
  An index over the SYS1.CDRSRVCR data set
SYS1.CDRSRVCR
  The CCSID resource table
SYS1.CDRSRVGR
  The GCCST resource table

These data sets will occupy approximately 350 tracks of 33xx-type direct access storage.

API Tracing

API tracing can be performed by coding a DD statement for CDRATRC in the job control language for job steps invoking CDRA services. The trace statement can be entered as follows:

//CDRATRC DD SYSOUT=*

Application Programming Considerations

The application program must ensure that the appropriate Language Environment run time environment is enabled prior to calling any of the CDRA based services.

If CDRA is invoked without having performed the recommended installation process that linkedits the CDRA component with SYS1.SCEELKED, the result will be an error with a status code of 2048 and a reason code of 2, when the CDRA component attempts to call a C runtime function. For more information on the Language Environment, please see the Language Environment Programming Guide, SC26-4818.

The calling application program may or may not be APF authorized and can be running in any non-zero protect key. The calling program must be AMODE(31) and must be in TCB mode.

The CDRA services have been implemented using the C programming language, but the functions themselves can be called in a language independent manner from any high-level programming language.

Two programming examples are provided to show how an application program can call the CDRA services. For applications written using AD/CYCLE C, the CDRA services can be accessed in the same manner as other C functions are called. Click on ‘Sample C Routine’ below to view the code. COBOL based applications can invoke the CDRA services as shown in the COBOL programming example. Click on ‘Sample COBOL Routine’ below to view the code. PL/I based applications can similarly invoke CDRA services.

Sample C Routine

This routine demonstrates how to call the DFSMS/MVS CDRA services to perform the multistep character data conversion using the C programming language.

Sample C Routine

```
#pragma runopts(STACK(16K,4K,BELOW),HEAP(64K,4K,BELOW))


#pragma title("CDRATEST")
#pragma options(RENT)
#pragma strings(readonly)
#pragma csect(code, "CDRATEST")
#pragma csect(static, "#CDRATST")


/*#pragma string(WRITABLE)*/
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <ctype.h>
#include <string.h>
/* Definitions required to call CDRA APIs */
typedef long CDRASRV_CCSID_T;


typedef struct { /* structure used to return a return code */
short Status;
short Reason;
long reserved1;
long reserved2;
} CDRASRV_FeedBack_T;


void CDRMSCI(const CDRASRV_CCSID_T * ccsid1,
const long * StringType1,
const CDRASRV_CCSID_T * ccsid2,
const long * StringType2,
const long * gccasn,
long * Token,
CDRASRV_FeedBack_T * FeedBack);


void CDRMSCP(const long * Token,
const char * String1,
const long * Length1,
const long * Length2,
char * String2,
long * OutLength,
long * ErrorByteNr,
CDRASRV_FeedBack_T * FeedBack);


void CDRMSCC( long * Token,
CDRASRV_FeedBack_T * FeedBack);
```

```
main(int argc, char* argv[])
{
const long inputST = 0; /* type of input string */
const long outputST = 1; /* type of input string */
const long cnvGCCASN = 0; /* conversion alternative */
const long lL1 = 26;
const long lL2 = 160;
int RecDataLen, /* Record Length */
NumRecs, /* Number of Records */
RC1,RC2; /* Return Code */


int RecRtnCnt, /* record return count */
RecordLen; /* record length */
long int Token[8]; /* CDRA token */
char instring [80];
char outstring [160];
long inputCCSID, /* input CCSID */
outputCCSID; /* output CCSID*/
CDRASRV_FeedBack_T FeedBack; /* feed back area */
int long lL3,lL4; /* string length */
printf(" program starting \n");


memcpy(instring,"ABCEDFGHIJKLMNOPQRSTUVWXYZ",26);
inputCCSID = 500;
outputCCSID = 437;


CDRMSCI(&inputCCSID, /* EBCDIC codepage */
&inputST, /* graphic char with length specify */
&outputCCSID, /* ASCII codepage */
&outputST, /* graphic char with length specify */
&cnvGCCASN, /* installation default */
(long int *)&Token, /* handle */
&FeedBack); /* feed back */
if(!FeedBack.Status && !FeedBack.Reason)
printf("Sucessfully initialize multiple-step conversion.\n");
else {
printf("Unsucessfully initialize multiple-step conversion.\n");
printf("Status: %d, Reason: %d\n",FeedBack.Status,FeedBack.Reason);
return(1);
}
CDRMSCP((long int *)&Token, /* handle */
instring, /* string to be converted */
&lL1, /* input string length */
&lL2, /* output string length */
outstring, /* result string */
&lL3, /* result string length */
&lL4, /* result string error */
&FeedBack); /* feed back */
if(!FeedBack.Status && !FeedBack.Reason) {
printf("Sucessfully perfom multiple-step conversion.\n");
printf("OUTSTREAM=%s\n",outstring);
}
```

```
CDRMSCC((long int *)&Token, /* handle */
&FeedBack); /* feed back */
if(!FeedBack.Status && !FeedBack.Reason)
printf("Sucessfully clean up multiple-step conversion.\n");
else
printf("Unsucessfully clean up multiple-step conversion.\n");


}
```

Sample COBOL Routine

This example shows how to call the DFSMS/MVS CDRA services to initialize, perform data conversion on a character string, and cleanup, using the COBOL programming language.

Sample Cobol Routine

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CDRA.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.


01 CCSID1 PIC 9(9) COMP VALUE 500.
01 STRTYP1 PIC 9(9) COMP VALUE 0.
01 CCSID2 PIC 9(9) COMP VALUE 437.
01 STRTYP2 PIC 9(9) COMP VALUE 0.
01 GCCASN PIC 9(9) COMP VALUE 0.
01 INSTR PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
01 OUTSTR PIC X(26) VALUE " ".
01 STRLL1 PIC 9(9) COMP VALUE 26.
01 STRLL2 PIC 9(9) COMP VALUE 26.
01 STRLL3 PIC 9(9) COMP VALUE 0 .
01 STRLL4 PIC 9(9) COMP VALUE 0 .
01 TOKEN.
05 V OCCURS 8 TIMES.
10 V1 PIC 9(4) .
01 FB1 PIC 9(16) COMP.
01 FB2.
02 STAT PIC 9(4) COMP.
02 REASON PIC 9(4) COMP.
02 RES1 PIC 9(4) COMP.
02 RES2 PIC 9(4) COMP.
PROCEDURE DIVISION.
CALL "CDRMSCI" USING CCSID1, STRTYP1, CCSID2, STRTYP2,
GCCASN, TOKEN, FB2.
```

```
DISPLAY FB2 .
DISPLAY "RESULT" STAT, REASON.
CALL "CDRMSCP" USING TOKEN, INSTR, STRLL1, STRLL2,
OUTSTR, STRLL3, STRLL4, FB2.
DISPLAY "RESULT" STAT, REASON.
DISPLAY OUTSTR .
CALL "CDRMSCC" USING TOKEN, FB2.
DISPLAY "RESULT" STAT, REASON.
STOP RUN.
```

CDRA APIs in the DFSMS/MVS Product

The following CDRA APIs are included in the DFSMS/MVS product library:

- Get Encoding Scheme, Character Set, and Code Page Elements (CDRGESP)
- Get Short Form (CCSID) from Specified ES (CS, CP) (CDRSCSP)
- Get Encoding Scheme Element and its Sub-elements (CDRGESE)
- Get Control Function Definition (CDRGCTL)
- Get Short Form (CCSID) with Maximal CS for Specified ES, CP (CDRSMXC)
- Multiple-Step Convert Initialize (CDRMSCI)
- Multiple-Step Convert Perform (CDRMSCP)
- Multiple-Step Convert Clean Up (CDRMSCC)
- Extract Status and Reason Codes from Feedback Code (CDRXSRF)

CDRGESP - Get Encoding Scheme, Character Set, and Code Page Elements API

The following parameter ranges and functional differences apply when using the CDRGESP API (as described in section “CDRGESP – Get Encoding Scheme, Character Set, and Code Page Elements” of Chapter 5) with the DFSMS/MVS product.

GESP parameter difference
none

CDRSCSP - Get Short Form (CCSID) from Specified ES (CS, CP) API

The following parameter ranges and functional differences apply when using the CDRSCSP API (as described in section “CDRSCSP – Get Short Form (CCSID) from Specified ES (CS,CP)” of Chapter 5) with the DFSMS/MVS product.

SCSP parameter difference
none

CDRGESE - Get Encoding Scheme Element and its Sub-elements API

The following parameter ranges and functional differences apply when using the CDRGESE API (as described in section “CDRGESE – Get Encoding Scheme Element and its Sub-elements” of Chapter 5) with the DFSMS/MVS product.

GESE parameter difference
none

CDRGCTL - Get Control Function Definition API

The following parameter ranges and functional differences apply when using the CDRGCTL API (as described in section “CDRGCTL – Get Control Function Definition” of Chapter 5) with the DFSMS/MVS product.

GCTL parameter difference
none

CDRSMXC - Get Short Form (CCSID) with Maximal CS for Specified ES, CP API

The following parameter ranges and functional differences apply when using the CDRSMXC API (as described in section “CDRSMXC – Get short Form (CCSID) with Maximal CS for Specified ES, CP”) with the DFSMS/MVS product.

SMXC parameter difference
none

CDRMSCI - Multiple-Step Convert Initialize API

The following parameter ranges and functional differences apply when using the CDRMSCI API (as described in section “CDRMSCI – Multiple-Step convert Initialize” of Chapter 5) with the DFSMS/MVS product.

MSCI parameter difference
only values of 0 or 1 are supported and BOTH of these values will provide the IBM defined defaults. The value 0 is used to select the designated installation default conversion method and tables. The value of 1 is used to select the CDRA-defined default method and conversion tables.

CDRMSCP - Multiple-Step Convert Perform API

The following parameter ranges and functional differences apply when using the CDRMSCP API (as described in section “CDRMSCP – Multiple-Step Convert Perform” of Chapter 5) with the DFSMS/MVS product.

MSCP parameter difference
none

CDRMSCC - Multiple-Step Convert Clean Up API

The following parameter ranges and functional differences apply when using the CDRMSCC API (as described in section “CDRMSCC – Multiple-Step convert Clean Up” of Chapter 5) with the DFSMS/MVS product.

MSCC parameter difference
none

CDRXSRF - Extract Status and Reason Codes from Feedback Code API

The following parameter ranges and functional differences apply when using the CDRXSRF API (as described in section “CDRXSRF – Extract Status and Reason Codes from Feedback Code” of Chapter 5) with the DFSMS/MVS product.

XSRF parameter difference
none

Appendix J. CDRA Conversion Resources

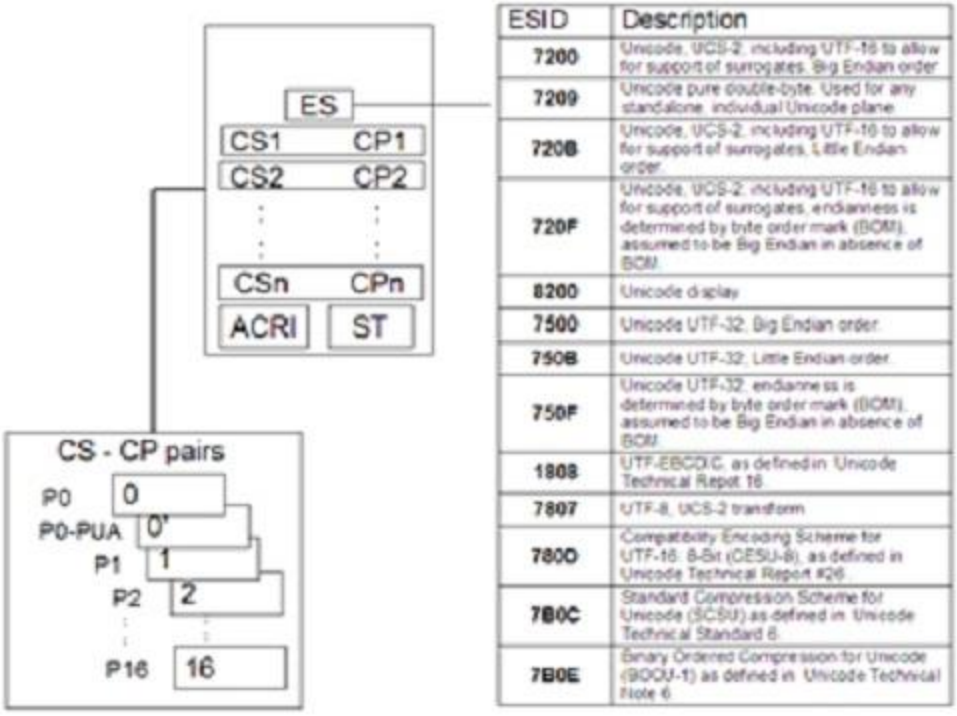
CDRA has defined many graphic character conversion tables to enable users to convert data between various encodings. Tables are defined for single-byte, double-byte, and mixed-byte encodings. There are many tables that convert legacy encodings into Unicode. The conversion table resources and supporting documentation are available from the internet via IBM developerWorks. They can be downloaded from the IBM developerWorks site. The tables are found in the Downloads & products view, under the heading Character Data Conversion Tables. Tables are also available by contacting the GCoC at gcoc@ca.ibm.com.

Appendix K. CDRA and Unicode

Character Data Representation Architecture (CDRA) defines a set of identifiers which are used to uniquely identify graphic character data. The Coded Character Set Identifier (CCSID) is a 16-bit integer that can be expanded to a long form identifier which contains the following information:

- Encoding Scheme
- Code Page Identifier(s)
- Character Set Identifier(s)
- Additional Coding-Related Required Information (ACRI) (optional)

In the case of Unicode, several encoding schemes are defined and for each Unicode CCSID there are multiple Code Page and Character Set pairs. The following figure shows how the Unicode CCSIDs are formulated.



ESID	Description
7200	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, Big Endian order
7209	Unicode pure double-byte. Used for any standalone individual Unicode plane
720B	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, Little Endian order
720F	Unicode, UCS-2, including UTF-16 to allow for support of surrogates, endianness is determined by byte order mark (BOM), assumed to be Big Endian in absence of BOM
8200	Unicode display
7500	Unicode UTF-32, Big Endian order
750B	Unicode UTF-32, Little Endian order
750F	Unicode UTF-32, endianness is determined by byte order mark (BOM), assumed to be Big Endian in absence of BOM
1808	UTF-EBCDIC, as defined in Unicode Technical Report 16
7807	UTF-8, UCS-2 transform
780D	Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8), as defined in Unicode Technical Report #26
7B0C	Standard Compression Scheme for Unicode (SCSU) as defined in Unicode Technical Standard 6
7B0E	Binary Ordered Compression for Unicode (BOCU-1) as defined in Unicode Technical Note 6

Figure 80. Unicode CCSID Structure

Unicode Identifiers

While Unicode is a very unique code in terms of how it is defined and the several formats that it can be used in, it can still be well defined using the standard CDRA identifiers listed above. The following sections describe how the IBM and CDRA identifiers have been applied to handle Unicode.

Unicode Encoding Schemes

The Unicode encoding space is well defined within the standard as are the various formats that Unicode data may be encoded using. Refer to the section in Appendix A on the Unicode Code Structure for detailed information on the encoding structure and related CDRA encoding schemes.

IBM Code Page Identifiers in Support of Unicode

Within the IBM corporate code page registry, the values in the range 01400 through 01499 have all been reserved for assignment to the individual components of Unicode (ISO 10646). The following table shows which values have been assigned or reserved for a specific purpose. The Unicode Standard deals with the Unicode character repertoire as a single entity. For managing such a large set of characters, CDRA defines a unique code page for each plane of Unicode.

Code Page	Plane	Comments
1400	0 - BMP	Basic Multilingual Plane (does not include PUA area)
1401	1 - SMP	Supplementary Multilingual Plane
1402	2 - SIP	Supplementary Ideographic Plane
1403-1413	3 - 13	Currently unassigned - reserved for planes 3 through 13
1414	14 - SSP	Supplementary Special-Purpose Plane
1415-1444	-	Not currently assigned, reserved for future assignment for Unicode code page components
1445	IBM advanced function print PUA	Registered for private use area 1 of plane 15. See code page definition for details.

Code Page	Plane	Comments
1446	15	Private Use Plane 15
1447	16	Private Use Plane 16
1448	PUA	Reserved for PUA area of BMP including corporate zone
1449	IBM default PUA	Registered IBM default for the PUA area of BMP
65520	Special Value Empty Plane	Registered IBM Special value used to indicate an empty Unicode Plane

Figure 81. Code Page Identifiers in Support of Unicode

Additionally, the values 1200 through 1249 in the code page registry have been marked as reserved. This is to prevent these values from being used for code page assignments as the corresponding values are used for the Unicode CCSIDs.

IBM Character Set Identifiers in Support of Unicode

The IBM corporate character set registry contains the definition for all graphic character sets used within IBM as well as the definition of some special purpose values. One of these special purpose values is X'FFFF' or 65535. When used in a CCSID definition, in conjunction with a valid code page value, this value indicates that the character set (CS) for this CCSID is growing. This means that from time to time more characters will be added to the set and the character set to be used with this CCSID is the current maximal set associated with the code page. When dealing with Unicode identifiers this is a very useful value since the Unicode character set is still growing at regular intervals. What this means is that a product that supports Unicode can use a CCSID with a growing character set and not have to change the CCSID value every time more characters are added to Unicode. There is also a fixed character set that corresponds to the growing set at a given point in time; usually a specific version of Unicode. This allows products that are concerned with precise definitions to use exact identifiers while others can use the less specific growing values. The following character set identifiers are used with the various code pages assigned for use in Unicode CCSID definitions.

Character Set	Plane Number	Comments
3001	0 - BMP	Unicode 2.0 character repertoire
3002-3003	-	Reserved for future Unicode definitions
3004	0 - BMP	Unicode 3.0 character repertoire
3005	0 - BMP	Unicode 4.0 character repertoire
3006	1	Unicode 4.0 character repertoire for Plane 1
3007	2	Unicode 4.0 character repertoire for Plane 2

Character Set	Plane Number	Comments
3008	14	Unicode 4.0 character repertoire for Plane 14
3009	0 - BMP	Unicode 4.1 character repertoire
3010	1	Unicode 4.1 character repertoire for Plane 1
3011	0 – BMP	Unicode 5.0 character repertoire
3012	1	Unicode 5.0 character repertoire for Plane 1
3013	0 - BMP	Unicode 5.1 character repertoire for Plane 0
3014	1	Unicode 5.1 character repertoire for Plane 1
3015	0 - BMP	Unicode 5.2 character repertoire for Plane 0
3016	1	Unicode 5.2 character repertoire for Plane 1
3017	2	Unicode 5.2 character repertoire for Plane 2
3018	0 - BMP	Unicode 6.0 character repertoire for Plane 0
3019	1	Unicode 6.0 character repertoire for Plane 1
3020	2	Unicode 6.0 character repertoire for Plane 2
3021	0 - BMP	Unicode 6.2 character repertoire for Plane 0
3022	1	Unicode 6.1 character repertoire for Plane 1
3023	0 - BMP	Unicode 8.0 character repertoire for Plane 0
3024	1	Unicode 8.0 character repertoire for Plane 1
3025	2	Unicode 8.0 character repertoire for Plane 2
3026	0 - BMP	Unicode 9.0 character repertoire for Plane 0
3027	1	Unicode 9.0 character repertoire for Plane 1
3028 – 3094	-	Reserved for future Unicode definitions
3095	15*	IBM Advanced Function Printing private use area no. 1 (*for use in row FF of PUA plane 15)
3096	15	Unicode 4.0 generic PUA definition for Plane 15
3097	16	Unicode 4.0 generic PUA definition for Plane 16
3098	PUA of BMP	Reserved for BMP PUA full character set of CP 1448
3099	PUA of BMP	IBM Default PUA definition
65535	any	Growing character set, use the current maximal set

Figure 82. Character Set Identifiers in Support of Unicode

CCSIDs Defined in Support of Unicode

The basic principle of CDRA is to be able to unambiguously identify data based on a unique, well defined identifier. The CDRA Coded Character Set IDentifier (CCSID) can be used to do this for Unicode data. Figure 80 above shows how each Unicode CCSID is composed. Each CCSID, can be expanded to a long form consisting of an Encoding Scheme (ES), and a list of character set, code page pairs (CSn, CPn) and optionally

ACRI (Additional Coding-Related Required Information). Each CCSID also has a string type (ST) characteristic associated with it which may be specified. In the case of Unicode CCSIDs, if the ST is not specified it defaults to ST 10. These string types cannot be enforced on incoming data, however any data originating within IBM should comply to the string type properties. For more information on String types see “Types of Strings” in chapter 6. In the case of Unicode, each full CCSID definition has 18 CS, CP pairs. The first pair is for the basic multilingual plane (BMP or plane 0) not including the private use area (PUA). The second CS, CP pair is for the PUA of the BMP. The subsequent sets represent the character sets and code pages associated with each of planes 1 through 16. Special CS and CP values of 65520 have been defined to represent an 'empty' Unicode plane and are used for all planes that are unpopulated. Empty planes may be omitted from any CCSID definition so long as the implementation has a well-defined means of determining which planes are included in the definitions and which ones have been omitted because they are unpopulated.

CDRA has used a combination of 'growing' and 'fixed' CCSIDs for Unicode. CCSID 1200 was the first Unicode CCSID defined. It is a growing CCSID with an encoding scheme of 7200 and was initially defined using code page 1400 with a growing character set (CS 65535) for the BMP (without the PUA) and code page 1449 with the fixed set character set 3099. This character set has the IBM defined default PUA characters in the last 256 positions of the PUA area and generic characters in all other PUA positions. Planes 1 through 16 were all 'empty'. As this is a growing CCSID, over time, as the definition of Unicode expanded so too did the definition of CCSID 1200. Today CCSID 1200 includes the initial two code page and character set pairs but has been expanded to include code pages 1401, 1402 and 1414 with growing character sets for planes 1, 2 and 14 respectively. It also includes code pages 1446 and 1447 for planes 15 and 16 with default character set definitions of 3096 and 3097. Planes 3 through 13 inclusive remain undefined using the special 65520 code page and character set in the full definition.

The following table presents a list of the Unicode CCSIDs currently defined. The full definition for each of these CCSIDs can be found in the CDRA CCSID Repository.

CCSID Decimal	Description	CCSID Decimal	Description
1200	UTF-16 BE with IBM PUA	21681	Unicode 4.0, UTF-16
1201	UTF-16 BE	21682	Unicode 4.0, UTF-16 LE with IBM PUA
1202	UTF-16 LE with IBM PUA	21683	Unicode 4.0, UTF-16 LE
1203	UTF-16 LE	21688	Unicode 4.1, UTF-8 with IBM PUA

CCSID Decimal	Description	CCSID Decimal	Description
1204	UTF-16 with IBM PUA	21689	Unicode 4.1, UTF-8
1205	UTF-16	21712	Unicode 5.2, UTF-32 BE with IBM PUA
1208	UTF-8 with IBM PUA	21880	Unicode 5.0 BMP
1209	UTF-8	21881	Unicode 5.2, Plane 1
1210	UTF-EBCDIC with IBM PUA	25776	Unicode 4.1, UTF-16 with IBM PUA
1211	UTF-EBCDIC	25777	Unicode 4.1, UTF-16
1212	SCSU with IBM PUA	25778	Unicode 4.1, UTF-16 LE with IBM PUA
1213	SCSU	25779	Unicode 4.1, UTF-16 LE
1214	BOCU-1 with IBM PUA	25784	Unicode 5.0, UTF-8 with IBM PUA
1215	BOCU-1	25785	Unicode 5.0 UTF-8
1232	UTF-32 BE with IBM PUA	25808	Unicode 6.0, UTF-32 BE with IBM PUA
1233	UTF-32 BE	25976	Unicode 5.1, BMP
1234	UTF-32 LE with IBM PUA	25977	Unicode 6.0, Plane 1
1235	UTF-32 LE	29872	Unicode 5.0, UTF-16 with IBM PUA
1236	UTF-32 with IBM PUA	29873	Unicode 5.0, UTF-16
1237	UTF-32	29874	Unicode 5.0, UTF-16 LE with IBM PUA
1400	Unicode BMP	29875	Unicode 5.0, UTF-16 LE
1401	Unicode Plane 1	29880	Unicode 5.1, UTF-8 with IBM PUA
1402	Unicode Plane 2	29881	Unicode 5.1, UTF-8
1414	Unicode Plane 14	29904	Unicode 6.2 UTF-32 BE with IBM PUA
1446	Unicode Plane15	30072	Unicode 5.2, BMP
1447	Unicode Plane 16	30073	Unicode 6.1, Plane 1
1448	Unicode, Generic PUA of BMP	33968	Unicode 5.1, UTF-16 BE with IBM PUA
1449	Unicode, PUA of BMP, IBM Default	33969	Unicode 5.1, UTF-16 BE
5304	Unicode 2.0, UTF-8 with IBM PUA	33970	Unicode 5.1, UTF-16 LE with IBM PUA

CCSID Decimal	Description	CCSID Decimal	Description
5305	Unicode 2.0, UTF-8	33971	Unicode 5.1, UTF-16 LE
5328	Uncode 4.0, UTF-32 BE with IBM PUA	33976	Unicode 5.2, UTF-8 with IBM PUA
5329	Unicode 8.0, UTF0-32 BE	33977	Unicode 5.2, UTF-8
5496	Unicode 2.0 BMP	34000	Unicode 8.0, UTF-32 BE with IBM PUA
5497	Unicode 4.0, Plane 1	34168	Unicode 6.0, BMP
5498	Unicode 4.0, Plane 2	34169	Unicode 8.0, Plane 1
5510	Unicode 4.0, Plane 14	38064	Unicode 5.2, UTF-16 BE with IBM PUA
9400	CESU-8 with IBM PUA	38065	Unicode 5.2, UTF-16
9424	Unicode 4.1, UTF-32 BE with IBM PUA	38066	Unicode 5.2, UTF-16 LE with IBM PUA
9592	Unicode 3.0, BMP	38067	Unicode 5.2, UTF-16 LE
9593	Unicode 4.1, Plane 1	38072	Unicode 6.0, UTF-8 with IBM PUA
9594	Unicode 5.2, Plane 2	38073	Unicode 6.0, UTF-8
13488	Unicode 2.0, UTF-16 IBM PUA	38264	Unicode 6.2, BMP
13489	Unicode 2.0, UTF-16	38265	Unicode 9.0, Plane 1
13490	Unicode 2.0, UTF-16 LE with IBM PUA	42160	Unicode 6.0, UTF-16 BE with IBM PUA
13491	Unicode 2.0, UTF-16 LE	42161	Unicode 6.0, UTF-16 BE
13496	Unicode 3.0, UTF-8 with IBM PUA	42162	Unicode 6.0, UTF-16 LE with IBM PUA
		42163	Unicode 6.0, UTF-16 LE
13497	Unicode 3.0, UTF-8	42168	Unicode 6.2, UTF-8 with IBM PUA
13520	Unicode 5.0, UTF-32 with IBM PUA	42169	Unicode 6.2, UTF-8
13688	Unicode 4.0, BMP	42360	Unicode 8.0. BMP
13689	Unicode 5.0, Plane 1	46256	Unicode 6.2, UTF-16 BE with IBM PUA
13690	Unicode 6.0, Plane 2	46257	Unicode 6.2, UTF-16 BE
17584	Unicode 3.0, UTF-16 with IBM PUA	46258	Unicode 6.2, UTF-16 LE with IBM PUA
17585	Unicode 3.0, UTF-16	46259	Unicode 6.2, UTF-16 LE

CCSID Decimal	Description	CCSID Decimal	Description
17586	Unicode 3.0, UTF-16 LE with IBM PUA	46264	Unicode 8.0, UTF-8 with IBM PUA
17587	Unicode 3.0, UTF-16 LE	46265	Unicode 8.0, UFT-8
17592	Unicode 4.0, UTF-8 with IBM PUA	46456	Unicode 9.0, BMP
17593	Unicode 4.0, UTF-8	50352	Unicode 8.0, UTF-16 BE with IBM PUA
17616	Unicode 5.1, Utf-32 BE with IBM PUA	50353	Unicode 8.0, UTF-16 BE
17784	Unicode 4.1, BMP	50354	Unicode 8.0, UTF-16 LE with IBM PUA
17785	Unicode 5.1, Plane 1	50355	Unicode 8.0, UTF-16 LE
17786	Unicode 8.0, Plane 2	54448	Unicode 9.0, UTF-16 BE with IBM PUA
21680	Unicode 4.0, UTF-16 with IBM PUA	65520	Unicode, empty plane

Figure 83. Unicode CCSIDs

In addition to the above CCSIDs, several 'special' CCSIDs have been defined for exclusive use by several IBM customers. These CCSID values have been assigned from the customer use range and are not intended for general use. The special CCSIDs are used to allow customers to define their own character assignments for the private use area (PUA).

Appendix L. EBCDIC control character definitions.

The EBCDIC control codes are defined in the IBM Corporate Standard, C-S 3-3220-002. Appendix L contains an excerpt from the standard which includes the definition of each control character as well as supporting documentation which provides the user with additional information related to the definition and use of the control characters.

The EBCDIC control characters have been divided into eight major classifications based on their function. Each classification is described in section G-1.1 below and includes a list of those controls contained within the classification. Additionally, there are three unassigned and unclassified control codes which have been reserved for future use by IBM products.

Control character classification

Customer Use Control Characters

As the name suggests, these characters are used to designate customer assigned functions. These control characters must not be implemented on IBM products. (CU1, CU3)

Device Control Characters

These characters are used to control devices, or to control major functions of devices (CSP, DC1, DC2, DC3, DC4, MFA, POC, SA, SEL, SFE, WUS).

Error Control Characters

These characters are used for error control, for indicating alarms, or for identifying or requesting identification of stations in a communications system (ACK, BEL, CAN, DEL, ENQ, EO, NAK, SUB).

Formatting or Editing Control Characters

Characters in this classification group are used for formatting, or for editing data (BS, CR, DS, FF, FS, HT, IR, IT, LF, NBS, NL, PP, RFF, RNL, SBS, SOS, SPS, UBS, VT). Characters SP, NSP, RSP and SHY are special characters within this group. Additional

information is provided with the specific control character definitions in section G-1.2 below.

Grouping Control Characters

These characters are used for grouping data or information. Depending on the actual control character, they may be at the start of the data, at the end of a data-block, at the start and end of a data-block (data-framing), at the end of communications control block or procedure, etc., (EM, EOT, ETB, ETX, IFS, IGS, IR, IRS, ITB/IUS, SOH, STX, TRN).

Mode Control Characters

These characters are used to set modes of operation, to change a mode of operation or to restore previous mode of operation (BYP/INP, DLE, ESC, GE, IT, RES/ENP, RPT, SI, SO, SM/SW).

Synchronization Control Characters

Characters in this classification group are used for synchronization of communications systems, or for synchronization of data within a format or for synchronization of data streams with certain timing characteristics of some device functions (EO, NUL, SYN).

Communication Control Characters

These characters (which also fall into the major function classifications above) are reserved exclusively for communications control (ACK, DLE, ENQ, EOT, ETB, ETX, IUS/ITB, NAK, SOH, STX, SYN). These controls must not be used for device control.

Control character definitions

Specific definitions for the control characters are given below. Each definition includes the control mnemonic, control name, hexadecimal code assignment and a brief description. Note that these are generalized descriptions. Specific architectures may give more complete implementation details.

NUL - Null (Hex 00)

A synchronization control character, with an all-zeros bit-pattern, which may serve to accomplish time and media fill.

SOH - Start of Heading (Hex 01)

A communication grouping control character which is used at the beginning of a sequence of characters which constitute a machine-sensible address or routing information. Such a sequence is referred to as the heading.

STX - Start of Text (Hex 02)

A communication grouping control character which precedes a sequence of characters that is to be treated as an entity and entirely transmitted to the ultimate destination. Such a sequence is referred to as the text. STX may be used to terminate a sequence of characters started by SOH.

ETX - End of Text (Hex 03)

A communication grouping control character which is used to terminate a sequence of characters started with STX and transmitted as an entity.

SEL - Select (Hex 04)

A device control character that is used with a one-byte parameter to control a function within a device.

HT - Horizontal Tab (Hex 05)

A formatting control character that moves the active position horizontally to the next tab stop setting.

RNL - Required New Line (Hex 06)

A formatting control character that causes a mandatory move of the active position to the starting margin on the next line. Required New Line resets Indent Tab mode.

DEL - Delete (Hex 07)

An error control character which erases characters on perforated tape. Delete may also apply to other devices such as displays.

GE - Graphic Escape (Hex 08)

A mode control character used to extend the standard graphic set of the code table. It is a non-locking shift character which changes the graphic meaning of the next single following bit-pattern.

SPS - Superscript (Hex 09)

A formatting control character that causes a fractional line feed vertically, bottom to top. The value of the fraction is less than one line and is equal to the corresponding Subscript movement.

RPT - Repeat (Hex 0A)

A mode control character that sets a mode of operation such as managing a printer buffer to allow a device to print repeatedly the character string contained in that buffer.

VT - Vertical Tab (Hex 0B)

A formatting control character that moves the active position vertically, to the next in a series of predetermined lines.

FF - Form Feed (Hex 0C)

A formatting control character that moves the active position to the starting margin on the first predetermined printing line on the next form or page.

CR - Carriage Return (Hex 0D)

A formatting control character that moves the active position to the starting margin on the same line.

SO - Shift Out (Hex 0E)

A mode control character that indicates the bit-patterns which follow shall be interpreted according to the most recently designated Shift Out set.

SI - Shift In (Hex 0F)

A mode control character that indicates the bit-patterns which follow shall be interpreted according to the most recently designated Shift In set.

DLE - Data Link Escape (Hex 10)

A communication mode control character which will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communication networks.

DC1 - Device Control 1 (Hex 11)

A device control character that controls ancillary devices associated with data processing or telecommunications systems.

DC2 - Device Control 2 (Hex 12)

A device control character that controls ancillary devices associated with data processing or telecommunications systems.

DC3 - Device Control 3 (Hex 13)

A device control character that controls ancillary devices associated with data processing or telecommunications systems.

RES/ENP - Restore/Enable Presentation (Hex 14)

A mode control character that terminates the Bypass/Inhibit Presentation mode of operation and activates associated printers or displays.

NL - New Line (Hex 15)

A formatting control character that moves the active position to the starting margin on the next line.

BS - Backspace (Hex 16)

A formatting control character that moves the active position horizontally to the next position in the reverse direction.

POC - Program Operator Communication (Hex 17)

A device control character that is used with two one-byte parameters to provide a communication protocol between end users. The first parameter defines the function and the second defines a unit associated with the function (for example, indicator light or function key).

CAN - Cancel (Hex 18)

An error control character that is used to indicate the data with which it is sent is in error or is to be disregarded.

EM - End of Medium (Hex 19)

A grouping control character associated with the transmitted data that may be used to identify physical end of medium, or the end of the used, or wanted, portion of information recorded on a medium. (The position of this character does not necessarily correspond to the physical end of the medium.)

UBS - Unit Backspace (Hex 1A)

A formatting control character that moves the active position horizontally in the reverse direction a fraction of the space used for a graphic. It is used, for example, for vertical alignment of line endings on a proportional spacing device.

CU1 - Customer Use 1 (Hex 1B)

A customer use control character sent into a system to designate a customer assigned function. This control character must not be implemented by IBM products.

IFS - Interchange File Separator (Hex 1C)

A grouping control character that terminates an information block called a FILE. (See definitions of IGS, IRS, IUS.)
When used in hierarchical order, the hierarchical order is ascending, IUS, IRS, IGS, IFS.
An information block must not be split by a higher order separator, for example, a RECORD may contain a whole number of UNITS, but may not contain a part of a UNIT.

IGS - Interchange Group Separator (Hex 1D)

A grouping control character that terminates an information block called a GROUP. (See definition of IFS, Interchange File Separator.)

IRS - Interchange Record Separator (Hex 1E)

A grouping control character that terminates an information block called a RECORD. (See definition of IFS, Interchange File Separator.)

IUS/ITB - Interchange Unit Separator, Intermediate Transmission Block (Hex 1F)

A grouping control character that terminates an information block called a UNIT. (See definition of IFS, Interchange File Separator). In Binary Synchronous Communications line control, this character is used to indicate the end of an intermediate block of data.

DS - Digit Select (Hex 20)

An editing control character that causes either a digit from the source field or a fill character to be inserted in the result field, in CPU editing operations.

SOS - Start of Significance (Hex 21)

An editing control character that causes either a digit from the source field or a fill character to be inserted in the result field, and also indicates, by setting a status indicator, that the following digits are significant in CPU editing operations.

FS - Field Separator (Hex 22)

An editing control character that identifies individual fields in a multiple field CPU editing operation.

WUS - Word Underscore (Hex 23)

A device control character that causes the entire word immediately preceding it to be underscored.

BYP/INP - Bypass/Inhibit Presentation (Hex 24)

A mode control character that deactivates the associated printers or displays and causes the succeeding control characters except the communication control characters and Restore/Enable Presentation to be ignored. Restore/Enable Presentation resets this mode.

LF - Line Feed (Hex 25)

A formatting control character that moves the active position vertically to the next line.

ETB - End of Transmission Block (Hex 26)

A communication grouping control character that indicates the end of a block of data for transmission purposes.

ESC - Escape (Hex 27)

A mode control character used to provide code extension in general information exchange. The Escape character itself is a prefix affecting the interpretation of a limited number of contiguously following characters.

Escape sequences are used to obtain additional control functions. Such control functions must not be used as additional communication controls.

SA - Set Attribute (Hex 28)

A device control character that indicates the beginning of a fixed length control sequence.
It is recommended that this control character not be implemented on future *products*, unless such *products* are required to provide compatibility with a previously announced *product*.
This control function can be provided by a CSP (hex 2B) control sequence.

SFE - Start Field Extended (Hex 29)

A device control character that indicates the beginning of a variable length control sequence.
It is recommended that this control character not be implemented on future *products*, unless such *products* are required to provide compatibility with a previously announced *product*. This control function can be provided by a CSP (hex 2B) control sequence.

SM/SW - Set Mode/Switch (Hex 2A)

A mode control character that sets a mode of operation such as switching between two print buffers during a print operation.

CSP - Control Sequence Prefix (Hex 2B)

A control character that indicates the beginning of a variable length control sequence. CSP sequences appear as:

```
CSP,CLASS,COUNT,TYPE,P1,P2,. . .,Pn.
```

where:

Class specifies a class or set of control functions which have a common purpose or attribute.
The binary *count* indicates the number of bytes to the end of the CSP sequence, including the count byte.
Minimum value of the count field is 2. If the count field is 2, then there is a type field and no parameter field. If the count field is 3 or larger, then the sequence must contain a type field and one or more parameter fields.

Type is a one-byte field; it specifies one control function within a class of control functions.
P1 . . . *Pn* are parameters. The length of each parameter depends upon the class and type fields.
The aggregate length of all the parameters may be in the range of 0 to 253.

MFA - Modify Field Attribute (Hex 2C)

A device control character that indicates the beginning of a variable length control sequence.
It is recommended that this control character not be implemented on future *products*, unless such *products* are required to provide compatibility with a previously announced *product*.
This control function can be provided by a CSP (hex 2B) control sequence.

ENQ - Enquiry (Hex 2D)

A communication control character that is used in data communication systems as a request for a response from a remote station. It may be used as a *Who Are You* (WRU) to obtain identification, or may be used to obtain station status, or both.

ACK - Acknowledge (Hex 2E)

A communication control character that is transmitted by a receiver as an affirmative response to a sender.

BEL - Bell (Hex 2F)

A control character that is used when there is a need to call for human attention. It may control alarm or attention devices or cause a device to stop.

Reserved (Hex 30)

A control location reserved for assignment of future functions.

Reserved (Hex 31)

A control location reserved for assignment of future functions.

SYN - Synchronous Idle (Hex 32)

A communication synchronization control character that is used by a synchronous transmission system in the absence of any other character to provide a signal from which synchronism may be achieved or retained.

IR - Index Return (Hex 33)

As a formatting control character, it moves the active position to the starting margin on the next line. As a grouping control character, it terminates an information block called a unit.

PP - Presentation Position (Hex 34)

A formatting control character that is used with two one-byte parameters to move the active position. The first parameter defines the function and the second parameter is a binary number that denotes either a column or line number.

TRN - Transparent (Hex 35)

A grouping control character that is used with a one-byte parameter to denote the start of a transparent data stream. The parameter is a binary count of the number of bytes of transparent data not including the count byte.

NBS - Numeric Backspace (Hex 36)

A formatting control character that moves the active position horizontally, in the reverse direction, a distance equal to the space used for digits (0-9) in the pitch being used.

EOT - End of Transmission (Hex 37)

A communication grouping control character that is used to indicate the conclusion of a transmission.

SBS - Subscript (Hex 38)

A formatting control character that causes a fractional line feed vertically, top to bottom. The value of the fraction is less than one line and is equal to the corresponding Superscript movement.

IT - Indent Tab (Hex 39)

A mode control character that causes a Horizontal Tab to be executed immediately and following every subsequent New Line control character. This mode has the effect of indenting the starting margin. Multiple Indent Tabs may be used for greater indentation. Indent Tab mode is reset by Required New Line or Required Form Feed.

RFF - Required Form Feed (Hex 3A)

A formatting control character that causes a mandatory move of the active position to the starting margin of the first line of the next page. Required Form Feed resets Indent Tab mode.

CU3 - Customer Use 3 (Hex 3B)

A customer use control character sent into a system to designate a customer assigned function. This control character must not be implemented by IBM products.

DC4 - Device Control 4 (Hex 3C)

A device control character that controls ancillary devices associated with data processing or telecommunications systems, more especially switching devices on or off. (If a single stop control is required to interrupt or turn off ancillary devices, DC4 is the preferred assignment.)

NAK - Negative Acknowledge (Hex 3D)

A communication control character that is transmitted by a receiver as a negative response to the sender.

Reserved (Hex 3E)

A control location reserved for assignment of future functions.

SUB - Substitute (Hex 3F)

A control character that replaces a character that is determined to be invalid or in error, or, for graphic display devices is inserted at the end of a message to signify that some character or characters in the message are invalid.

EO - Eight Ones (Hex FF)

A synchronization control character, with an all-ones bit-pattern, which may serve to accomplish time and media fill.

G-1.3 Special graphic characters

Code points with hexadecimal reference numbers from 40 through FE inclusive are reserved for assignment of graphic characters. Four of these code points are often used for special characters which have the characteristics of control characters. Code Point hexadecimal reference number 40 is always assigned to the Space Character which is required in all EBCDIC code pages. The Required Space (RSP), Numeric Space (NSP), and Syllable Hyphen (SHY) coded graphic characters are optional, but when used, the preferred hexadecimal code points are: RSP (hexadecimal 41), NSP (hexadecimal E1), and SHY (hexadecimal CA). Definitions of these graphic characters are included below.

SP - Space (Hex 40)

A normally non-printing graphic character used to separate words. It is also a formatting character that moves the active position horizontally one position in the forward direction. This character can be eliminated by products that adjust text.

RSP - Required Space (preferred Hex 41)

A normally non-printing graphic character used to separate words. It is also a formatting character that moves the active position horizontally one position in the forward direction. This character cannot be eliminated by products that adjust text.

NSP - Numeric Space (preferred Hex E1)

A normally non-printing graphic character, on products with proportional spacing. It is also a formatting character that moves the active position horizontally a distance in the forward direction equal to the width of a numeric character.

SHY - Syllable Hyphen (preferred Hex CA)

A hyphen graphic character used at the end of a line after the syllable of a word that must be split onto the next line. This character can be eliminated by products that adjust text.

Glossary

This glossary includes definitions of terms and acronyms found in this document.

A

ACRI
See *additional coding-related required information*.

ACRI-PCMB
See *additional coding-related required information – PC Mixed byte*.

additional coding-related required information (ACRI)
The information, in addition to encoding scheme identifier, code page, and character set global identifiers, that is required to complete the definition associated with using particular encoding schemes. An example is the ranges of valid first bytes of double-byte code points in a PC Mixed single-byte and double-byte code.

additional coding-related required information - PC mixed byte (ACRI-PCMB) A CDRA identifier that defines the ranges of valid first bytes of double byte code points in a PC Mixed SB/DB encoding scheme.

American Standard Code for Information Interchange (ASCII)
A standard code used for information exchange among data processing systems, data communication systems, and associated equipment. ASCII uses a coded character set consisting of 7-bit coded characters.

API
See *application programming interface*.

APL
See *A programming language*.

application programming interface (API)
An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

A programming language (APL)
A programming language based on mathematical notation that is used to develop application programs. A is particularly useful for commercial data processing, system

design, mathematical and scientific computation, database applications, and teaching mathematics.

Arabic numeral
One of the 10 numerals used in decimal notation: the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. See also *Hindi numeral*.

ASCII
See *American Standard Code for Information Interchange*.

C

CCS
See *coded character set*.

CCSID
See *coded character set identifier*.

CCSID resource
A representation of the various elements associated with a CCSID in a system in a machine-readable form.

CCSID resource repository
An organized collection of CCSID resources that are maintained by a service provider in a system.

CDRA
See *Character Data Representation Architecture*.

CECP
See *country extended code page*.

CGCSGID
See *coded graphic character set global identifier*.

Character Data Representation Architecture (CDRA)
An IBM architecture that defines a set of identifiers, resources, services, and conventions to achieve consistent representation, processing, and interchange of graphic character data in heterogeneous environments.

coded character set (CCS)
A set of unambiguous rules that establishes a character set and the one-to-one

relationships between the characters of the set and their coded representations. See also *invariant character set*.

coded character set identifier (CCSID)
A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

coded graphic character set
A set of graphic characters with their assigned code points.

coded graphic character set global identifier (CGCSGID)
A 4-byte binary or a 10-digit decimal identifier consisting of the concatenation of a GCSGID and a CPGID. The CGCSGID identifies the code point assignments in the code page for a specific graphic character set, from among all the graphic characters that are assigned in the code page.

code extension method
A method prescribed in an encoding scheme for representing characters that cannot be accommodated within the limits of the basic structure of the code. It prescribes a method to alter the interpretation of one or more code points that follow a prescribed single control character or a control sequence.

code page
A specification of code points from a defined encoding structure for each graphic character in a set or in a collection of graphic character sets. Within a code page, a code point can have only one specific meaning. See also *invariant character set*.

code page global identifier (CPGID) A 5-digit decimal or 2-byte binary identifier that is assigned to a code page. The range of values is 00001 to 65534 (X'0001' to X'FFFE').

code point
A unique bit pattern defined in a code. Depending on the code, a code point can be 7-bits, 8-bits, 16-bits, or other. Code points are assigned graphic characters in a code page.

component
A hardware or software entity forming part of a system, or a piece of logic that controls the operation of a device, modifies, or stops a control function.

control function
An element of a character set that affects the recording, processing, transmission, or

interpretation of data, and that has a coded representation of, one or more, bit combinations (see ISO/IEC 6429).

conversion
The process of replacing a code point that is assigned to a character in one code with its corresponding code point assigned in another code.

conversion method
An algorithm used during conversion. It includes the necessary logic to separate the input code point string into appropriate substrings, converting the substrings and assembling the resultant substrings, for a set of criteria to be used during conversion. A conversion method may use associated conversion tables as resources during the conversion.

conversion table
A resource used with a conversion method to perform conversion. Typically, a conversion table contains a set of input code point values corresponding to a given set of output code point values. Its structure and contents are designed to suit the conversion algorithm with which it is to be used.

country extended code page (CECP)
A single-byte EBCDIC code page in the IBM corporate registry that contains the 190 characters found in character set 00697. While each CECP contains the same set of characters (allowing for conversion of data without loss), the code point allocation of the characters is not identical. For example, all CECPs contain the character backwards slash, however in code page 500 it is located at code point x'E0' and in code page 280 it is located at code point x'48'.

CPGID
See *code page global identifier*.

D

database (DB)
A collection of interrelated or independent data items that are stored together to serve one or more applications.

data stream
The commands, control codes, data, or structured fields that are transmitted between an application program and a device such as printer or nonprogrammable display station.

DB
See *database*.

DBCS
See *double-byte character set*.

DCF
See *Document Composition Facility*.

Distributed Relational Database Architecture (DRDA)
The architecture that defines formats and protocols for providing transparent access to remote data. DRDA defines two types of functions: the application requester function and the application server function.

Document Composition Facility (DCF)
An IBM licensed program used to format input to a printer.

double-byte character set (DBCS)
A set of characters in which each character is represented by 2 bytes. These character sets are commonly used by national languages, such as Japanese and Chinese, that have more symbols than can be represented by a single byte.

double-wide character
A character, such as a Kanji ideogram, that requires twice the nominal width of other characters, such as the letter A, for the character to be legible on a display screen or a printer.

DRDA
See *Distributed Relational Database Architecture*.

E

even parity bit
A check bit that is usually generated or included in a parity-checking algorithm to make the total number of bits in a bit pattern an even number. See also odd parity bit.

F

folding
The substitution of one graphic character for another. Folding generally maps a larger character set into a subset, and may result in loss of information. Folding allows the

presentation of uppercase graphic characters when lowercase characters are not available. See also *mono-casing*.

full character set
The maximal character set of a code page such that there are no more unassigned graphic code points remaining in the associated encoding scheme. No other larger character set can be represented in that code page. For example, CS 697 (the maximal character set of CP 500 in encoding scheme ES 1100), contains 190 graphic characters and is assigned all the 190 available graphic code points in ES 1100. See also *maximal character set* and *subset character set*.

G

GCCASN
See *graphic character conversion alternative selection number*.

GCCST
See *graphic character conversion selection table*.

GCSGID
See *graphic character set global identifier*.

graphic character
A graphic symbol, such as a numeric, alphabetic, or special character (see C-S 3-3220-019 Corporate Standard).

graphic character conversion alternative selection number (GCCASN) A parameter of a function call to a graphic character data conversion process that facilitates selecting a specific conversion method and associated conversion tables from different alternatives.

graphic character conversion selection table (GCCST)
A table used in the graphic character data conversion process to manage the access to the various conversion methods and associated conversion tables under its sphere of control.

graphic character set
A defined set of graphic characters treated as an entity. No coded representation is assumed.

graphic character set global identifier (GCSGID)
A unique five-digit decimal number assigned to a graphic character set in IBM

standards. The range of GCSGID values is 00001 to 65534 or x'0001' to x'FFFE' (see C-S 3-3220-019 Corporate Standard).

H

hardcoded
Pertaining to software instructions that are statically encoded and not intended to be altered.

high-level language (HLL)
A programming language that provides some level of abstraction from assembler language and independence from a machine.

Hindi numeral
Any of the set of numerals used in many Arabic countries instead of, or in addition to, the Arabic numerals. Hindi numeral shapes are ١٢٣٤٥٦٧٨٩, which correspond to the Arabic numeral shapes of 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. See also *Arabic numeral*.

HLL
See *high-level language*.

identity map
A special case of code point conversion in which all input code points are equal to the output code points, thus eliminating the need for a conversion. When converting data from one CCSID to another using the round-trip criterion, if the CCSIDs share the same CPGID, an identity mapping condition exists.

I

IEC
See *International Electrotechnical Commission*.

International Electrotechnical Commission (IEC)
The international standards-setting organization responsible for electrical and electrotechnical issues. IEC often cooperates with ISO via technical committees on the definition of standards.

International Organization for Standardization (ISO)
An international body charged with creating standards to facilitate the exchange of

goods and services as well as cooperation in intellectual, scientific, technological, and economic activity.

International Telegraphic Alphabet Number 2 (ITA-2)
A CCITT-defined coded character set used in the international Telex communication services, worldwide.

invariant character set
A set of characters, such as the syntactic character set, having the same code point assignments in all coded character sets or code pages using a given encoding scheme. See also code page, coded character set and syntactic character set.

ISO
See *International Organization for Standardization*.

ISO environment
A coding structure defined in ISO 2022 that uses single (or multiple) septet(s) (7-bit) or octet(s) (8-bit) per code point, with or without code extension controls.

ITA-2
See *International Telegraphic Alphabet Number 2*.

K

Katakana
A Japanese phonetic syllabary used primarily for foreign names and place names and words of foreign origin.

L

Latin alphabet
An alphabet composed of the letters a - z and A - Z with or without accents and ligatures. See also non-Latin-based alphabet.

Latin alphabet number 1
The 190 characters used in most of Western Europe, North America, Central and South America. There are other Latin alphabets such as Latin-2 and Latin-3 that correspond to some of the other ISO/IEC 8859 character sets. The numbering scheme is neither rational nor orderly.

lowercase
Pertaining to the small alphabetic characters, whether accented or not, as distinguished from the capital alphabetic characters. The concept of case also applies to alphabets such as Cyrillic and Greek, but not to Arabic, Hebrew, Thai, Japanese, Chinese, Korean, and many other scripts. Examples of lowercase letters are a, b, and c.

M

machine-readable information (MRI)
All textual information contained in a program such as a system control program, an application program, or microcode. MRI includes all information that is presented to or received from a user interacting with a system. This includes messages, dialog boxes, online manuals, audio output, animations, windows, help text, tutorials, diagnostics, clip art, icons, and any presentation control that is necessary to convey information to users.

maximal character set
The largest registered character set that is assigned to a registered code page following an encoding scheme. See also *full character set*.

mono-casing
The translation of alphabetic characters from one case (usually the lowercase) to their equivalents in another case (usually the uppercase). See also *folding*.

MRI
See *machine-readable information*.

N

national use graphics
Graphic characters on a coded character set that are not part of the invariant character set.

nibble
A bit-pattern consisting of four bits.

non-Latin-based alphabet
An alphabet comprising letters other than the Latin-based ones, such as those used in Greek and Arabic.

normalization support CCSID table (NSCT)
A table containing a default CCSID value associated with a pair of CCSIDs, which will be used to normalize two strings (that are coded in two different CCSIDs), before a string operation such as concatenation, comparison, or others is performed with the two strings.

NSCT
See *normalization support CCSID table*.

O

octet
A byte composed of eight binary elements.

odd parity bit
A check bit that is usually generated or included in a parity-checking algorithm to make the total number of bits in a bit pattern an odd number. See also *even parity bit*.

R

related default CCSID table
A table containing a default CCSID associated with another CCSID and an ESID. This default CCSID is the nearest equivalent of its associated CCSID based on some relationship between the two.

Revisable-Form-Text Document Content Architecture (RFTDCA)
The architectural specification for the information interchange of documents whose text is in a revisable format. A Revisable-Form Text Document Content Architecture document consists of structured fields, controls, and graphic characters that represent the format and meaning of the document.

RFTDCA
See *Revisable-Form-Text Document Content Architecture*.

S

septet
A 7-bit byte.

session
A logical or virtual connection between two stations, software programs, or devices on

a network that allows the two elements to communicate and exchange data for the duration of the session.

special character
A graphic character that is not a letter, a digit, or a space character and not an ideogram.

subset character set
A set of characters that is completely contained in another larger set of characters. See also *full character set*.

syntactic character set
A set of 81 graphic characters that are registered in the IBM registry as character set 00640. This set is used for syntactic purposes maximizing portability and interchangeability across systems and country or region boundaries. It is contained in most of the primary registered character sets, with a few exceptions. See also *invariant character set*.

system
A set of individual components, such as people, machines, or methods, that work together to perform a function.

T

tag
A mechanism used to identify certain attributes having some bearing on handling of character data. Some examples are character set identifier, code page identifier, language identifier, country identifier, and encoding scheme identifier.

U

UDC
See *user-defined character*.

uppercase
Pertaining to the capital alphabetic characters, as distinguished from the small alphabetic characters. The concept of case also applies to alphabets such as Cyrillic and Greek, but not to Arabic, Hebrew, Thai, Japanese, Chinese, Korean, and many other scripts. Examples of capital letters are A, B, and C. See also *lowercase*.

user
Any individual, organization, process, device, program, protocol, or system that uses the services of a computing system.

user-defined character (UDC)
A character which is defined by an individual user or organization for assignment in one or more code pages. These characters are often ideographic characters, symbols or logos. Some standards, including Unicode, reserve coding space for user defined characters. The meaning of the user defined character can only be assured within the closed environment of the defining organization or by private agreement among cooperating users.

W

ward
A section of a double-byte character set (DBCS) where the first byte of each DBCS code point belonging to that section is the same value.