[MS-UCODEREF]:
Windows Protocols Unicode Reference

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/14/2008	2.0.1	Editorial	Revised and edited the technical content.
03/14/2008	2.0.2	Editorial	Revised and edited the technical content.
05/16/2008	2.0.3	Editorial	Revised and edited the technical content.
06/20/2008	3.0	Major	Updated and revised the technical content.
07/25/2008	3.0.1	Editorial	Revised and edited the technical content.
08/29/2008	3.0.2	Editorial	Revised and edited the technical content.
10/24/2008	3.0.3	Editorial	Revised and edited the technical content.
12/05/2008	3.1	Minor	Updated the technical content.
01/16/2009	3.1.1	Editorial	Revised and edited the technical content.
02/27/2009	3.1.2	Editorial	Revised and edited the technical content.
04/10/2009	3.1.3	Editorial	Revised and edited the technical content.
05/22/2009	3.1.4	Editorial	Revised and edited the technical content.
07/02/2009	4.0	Major	Updated and revised the technical content.
08/14/2009	4.0.1	Editorial	Revised and edited the technical content.
09/25/2009	4.1	Minor	Updated the technical content.
11/06/2009	5.0	Major	Updated and revised the technical content.
12/18/2009	6.0	Major	Updated and revised the technical content.
01/29/2010	7.0	Major	Updated and revised the technical content.
03/12/2010	7.0.1	Editorial	Revised and edited the technical content.
04/23/2010	7.0.2	Editorial	Revised and edited the technical content.
06/04/2010	7.0.3	Editorial	Revised and edited the technical content.
07/16/2010	7.0.3	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	7.0.3	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	7.0.3	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	7.0.3	No change	No changes to the meaning, language, or formatting of

Date	Revision History	Revision Class	Comments
			the technical content.
01/07/2011	7.0.3	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	7.0.3	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This document is a companion reference to the protocol specifications. It describes how **Unicode** strings are compared in Microsoft Windows® protocols and how Windows supports Unicode conversion to earlier **codepages**. For example:

- **UTF-16** string comparison: Provides linguistic-specific comparisons between two Unicode strings and provides the comparison result based on the language and region for a specific user.
- Mapping of UTF-16 strings to earlier ANSI codepages: Converts Unicode strings to strings in the earlier codepages that are used in older versions of Windows and the applications that are written for these earlier codepages.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Unicode
UTF-16

The following terms are specific to this document:

codepage: An ordered set of characters of a specific script in which a numerical index (code-point value) is associated with each character. In this document, the term **codepage** is used in the context of **codepages** defined by Windows; **codepages** can also be called character sets or charsets.

double-byte character set (DBCS): A character encoding in which the code-points can be either one or two bytes. For example, the **DBCS** is used to encode Chinese, Japanese, and Korean languages.

single-byte character set (SBCS): A character encoding in which each character is represented by one byte. **Single-byte character sets** are limited to 256 characters.

sort keys: Numerical representations of a sort element based on locale-specific sorting rules. A sort key consists of several weighted components that represent a character's script, diacritics, case, and additional treatment based on locale.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[CODEPAGEFILES] Microsoft Corporation, "Windows Supported Code Page Data Files.zip", 2009, http://www.microsoft.com/downloads/details.aspx?FamilyID=5fdc09fb-afec-4c2a-9394-6d046841eace&displaylang=en

If you have any trouble finding [CODEPAGEFILES], please check here.

[ECMA-035] ECMA International, "Character Code Structure and Extension Techniques", 6th Edition, ECMA-035, December 1994, http://www.ecma-international.org/publications/standards/Ecma-035.htm

[GB18030] Chinese IT Standardization Technical Committee, "Chinese National Standard GB 18030-2005: Information technology — Chinese coded character set", Published in print by the China Standard Press, http://www.sj.cesi.cn/View.asp?ISBN=GB 18030-2005

[ISCII] Bureau of Indian Standards, "Indian Script Code for Information Exchange - ISCII", http://www.bis.org.in/dir/sales.htm

If you have any trouble finding [ISCII], please check here.

[MSDN-SWT/Vista] Microsoft Corporation, "Windows Vista Sorting Weight Table.txt", 2008, http://www.microsoft.com/downloads/details.aspx?FamilyID=5fdc09fb-afec-4c2a-9394-6d046841eace&displaylang=en

[MSDN-SWT/W2K3] Microsoft Corporation, "Windows NT 4.0 through Windows Server 2003 Sorting Weight Table.txt", 2008, http://www.microsoft.com/downloads/details.aspx?FamilyID=5fdc09fb-afec-4c2a-9394-6d046841eace&displaylang=en

[MSDN-SWT/W2K8] Microsoft Corporation, "Windows Server 2008 Sorting Weight Table.txt", 2008, http://www.microsoft.com/downloads/details.aspx?FamilyID=5fdc09fb-afec-4c2a-9394-6d046841eace&displaylang=en

[MSDN-SWT/Win7] Microsoft Corporation, "Windows 7 through Server 2008 R2 Sorting Weight Table.txt", 2009, http://www.microsoft.com/downloads/details.aspx?FamilyID=5fdc09fb-afec-4c2a-9394-6d046841eace&displaylang=en

If you have any trouble finding [MSDN-SWT/Win7], please check here.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2152] Goldsmith, D., and David, M., "UTF-7 A Mail-Safe Transformation Format of Unicode", RFC 2152, May 1997, http://www.ietf.org/rfc/rfc2152.txt

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

[UNICODE-BESTFIT] The Unicode Consortium, "WindowsBestFit", 2006, http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WindowsBestFit/

[UNICODE-COLLATION] The Unicode Consortium, "Unicode Technical Standard #10 Unicode Collation Algorithm", March 2008, http://www.unicode.org/reports/tr10/

[UNICODE-README] The Unicode Consortium, "Readme.txt", 2006, http://unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WindowsBestFit/readme.txt

[UNICODE5.0.0/CH3] The Unicode Consortium, "Unicode Encoding Forms", 2006, http://www.unicode.org/versions/Unicode5.0.0/ch03.pdf#G7404

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference", July 2007.

1.3 Overview

This document describes the following protocols when dealing with Unicode strings on the Microsoft Windows® platform:

- UTF-16 string comparison: This string comparison is used to provide a linguistic-specific comparison between two Unicode strings. This scenario provides a string comparison result based on the expectations of users from different languages and different regions.
- The mapping of UTF-16 strings to earlier codepages: This scenario is used to convert between Unicode strings and strings in the earlier codepage, which are used by older versions of Windows and applications written for these earlier codepages.

1.4 Applicability Statement

This reference document is applicable as follows:

- To perform UTF-16 character comparisons in the same manner as Microsoft Windows®. This document only specifies a subset of Windows behaviors that are used by other protocols. It does not document those Windows behaviors that are not used by other protocols.
- To provide the capability to map between UTF-16 strings and earlier codepages in the same manner as Windows.

1.5 Standards Assignments

The following standards assignments are used by the Windows Protocols Unicode Reference.

Parameter	Value	Reference
Codepage Data File (section 2.2.2)	Various	[UNICODE-BESTFIT]

2 Messages

The following sections specify how Windows Protocols Unicode Reference messages are transported and Windows Protocols Unicode Reference message syntax.

2.1 Transport

2.2 Message Syntax

2.2.1 Supported Codepage in Windows

Microsoft Windows® assigns an integer, called code page ID, to every supported codepage.

Based on the usage, the codepage supported in Windows can be categorized in the following:

- ANSI codepage

  ANSI codepages are codepages for which non-ASCII values (values greater than 127) represent international characters. These code pages are used natively in Microsoft Windows® 95 operating system, Microsoft Windows® 98 operating system, and Microsoft Windows® Millennium Edition operating system and are also available on Microsoft Windows NT® 4.0 operating system, Microsoft Windows® 2000 operating system, Windows® XP operating system, Windows Server® 2003 operating system, Windows Vista® operating system, Windows Server® 2008 operating system, Windows® 7 operating system, and Windows Server® 2008 R2 operating system.

  Windows code pages are also sometimes referred to as "active code pages" or "system active code pages". Windows always has one currently active Windows code page. All ANSI Windows functions use the currently active code page.

  The usual ANSI code page ID for US English is code page 1252.

  Windows code page 1252, the code page commonly used for English and other Western European languages, was based on an American National Standards Institute (ANSI) draft. That draft eventually became ISO 8859-1, but Windows code page 1252 was implemented before the standard became final, and is not exactly the same as ISO 8859-1.

- OEM codepage

  Original equipment manufacturer (OEM) code pages are code pages for which non-ASCII values represent line drawing and punctuation characters. These code pages were originally used for MS-DOS and are still used for console applications. They are also used for the non-extended file names in the FAT12, FAT16, and FAT32 file systems. The usual OEM code page ID for US English is code page 437.

- Extended codepage

  These codepages cannot be used as ANSI codepages, or OEM codepages. Windows is able to support conversions between Unicode and these codepages. These codepages are generally used for information exchange purpose with international/national standard or legacy systems. Examples are UTF-8, UTF-7, EBCDIC, and Macintosh code pages.

The following table shows all the supported code pages by Windows. The Codepage ID lists the integer number assigned to a codepage. ANSI/OEM codepages are in bold face. The Codepage

Description column describes the codepage. The Codepage Notes column lists the category of a codepage and the section in this document to find protocol information.

Codepage ID	Codepage Descriptions	Codepage Notes
37	IBM EBCDIC US-Canada	Extended codepage; for processing rules, see section 3.1.5.1.1.
437	OEM United States	OEM codepage; for processing rules, see section 3.1.5.1.1.
500	IBM EBCDIC International	Extended codepage; for processing rules, see section 3.1.5.1.1.
708	Arabic (ASMO 708)	Extended codepage; for processing rules, see section 3.1.5.1.1.
720	Arabic (Transparent ASMO); Arabic (DOS)	Extended codepage; for processing rules, see section 3.1.5.1.1.
737	OEM Greek (formerly 437G); Greek (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
775	OEM Baltic; Baltic (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
850	OEM Multilingual Latin 1; Western European (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
852	OEM Latin 2; Central European (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
855	OEM Cyrillic (primarily Russian)	OEM codepage; for processing rules, see section 3.1.5.1.1.
857	OEM Turkish; Turkish (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
858	OEM Multilingual Latin 1 + Euro symbol	OEM codepage; for processing rules, see section 3.1.5.1.1.
860	OEM Portuguese; Portuguese (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
861	OEM Icelandic; Icelandic (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
862	OEM Hebrew; Hebrew (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
863	OEM French Canadian; French Canadian (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
864	OEM Arabic; Arabic (864)	OEM codepage; for processing rules, see section 3.1.5.1.1.
865	OEM Nordic; Nordic (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.

Codepage ID	Codepage Descriptions	Codepage Notes
866	OEM Russian; Cyrillic (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
869	OEM Modern Greek; Greek, Modern (DOS)	OEM codepage; for processing rules, see section 3.1.5.1.1.
870	IBM EBCDIC Multilingual/ROECE (Latin 2); IBM EBCDIC Multilingual Latin 2	Extended codepage; for processing rules, see section 3.1.5.1.1.
874	ANSI/OEM Thai (same as 28605, ISO 8859-15); Thai (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
875	IBM EBCDIC Greek Modern	Extended codepage; for processing rules, see section 3.1.5.1.1.
932	ANSI/OEM Japanese; Japanese (Shift-JIS)	ANSI/OEM codepage; for processing rules, see section 3.1.5.1.1.
936	ANSI/OEM Simplified Chinese (PRC, Singapore); Chinese Simplified (GB2312)	ANSI/OEM codepage; for processing rules, see section 3.1.5.1.1.
949	ANSI/OEM Korean (Unified Hangul Code)	ANSI/OEM codepage; for processing rules, see section 3.1.5.1.1.
950	ANSI/OEM Traditional Chinese (Taiwan; Hong Kong SAR, PRC); Chinese Traditional (Big5)	ANSI/OEM codepage; for processing rules, see section 3.1.5.1.1.
1026	IBM EBCDIC Turkish (Latin 5)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1047	IBM EBCDIC Latin 1/Open System	Extended codepage; for processing rules, see section 3.1.5.1.1.
1140	IBM EBCDIC US-Canada (037 + Euro symbol); IBM EBCDIC (US-Canada-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1141	IBM EBCDIC Germany (20273 + Euro symbol); IBM EBCDIC (Germany-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1142	IBM EBCDIC Denmark-Norway (20277 + Euro symbol); IBM EBCDIC (Denmark-Norway-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1143	IBM EBCDIC Finland-Sweden (20278 + Euro symbol); IBM EBCDIC (Finland-Sweden-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1144	IBM EBCDIC Italy (20280 + Euro symbol); IBM EBCDIC (Italy-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1145	IBM EBCDIC Latin America-Spain (20284 + Euro symbol); IBM EBCDIC (Spain-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.

Codepage ID	Codepage Descriptions	Codepage Notes
1146	IBM EBCDIC United Kingdom (20285 + Euro symbol); IBM EBCDIC (UK-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1147	IBM EBCDIC France (20297 + Euro symbol); IBM EBCDIC (France-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1148	IBM EBCDIC International (500 + Euro symbol); IBM EBCDIC (International-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1149	IBM EBCDIC Icelandic (20871 + Euro symbol); IBM EBCDIC (Icelandic-Euro)	Extended codepage; for processing rules, see section 3.1.5.1.1.
1200	Unicode UTF-16, little-endian byte order (BMP of ISO 10646); available only to managed applications	Not used in Windows.
1201	Unicode UTF-16, big-endian byte order; available only to managed applications	Not used in Windows.
1250	ANSI Central European; Central European (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1251	ANSI Cyrillic; Cyrillic (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1252	ANSI Latin 1; Western European (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1253	ANSI Greek; Greek (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1254	ANSI Turkish; Turkish (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1255	ANSI Hebrew; Hebrew (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1256	ANSI Arabic; Arabic (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1257	ANSI Baltic; Baltic (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1258	ANSI/OEM Vietnamese; Vietnamese (Windows)	ANSI codepage; for processing rules, see section 3.1.5.1.1.
1361	Korean (Johab)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10000	MAC Roman; Western European (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10001	Japanese (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10002	MAC Traditional Chinese (Big5); Chinese	Extended codepage; for processing rules, see

Codepage ID	Codepage Descriptions	Codepage Notes
	Traditional (Mac)	section 3.1.5.1.1.
10003	Korean (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10004	Arabic (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10005	Hebrew (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10006	Greek (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10007	Cyrillic (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10008	MAC Simplified Chinese (GB 2312); Chinese Simplified (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10010	Romanian (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10017	Ukrainian (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10021	Thai (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10029	MAC Latin 2; Central European (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10079	Icelandic (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10081	Turkish (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
10082	Croatian (Mac)	Extended codepage; for processing rules, see section 3.1.5.1.1.
12000	Unicode UTF-32, little-endian byte order; available only to managed applications	Not used in Windows.
12001	Unicode UTF-32, big-endian byte order; available only to managed applications	Not used in Windows.
20000	CNS Taiwan; Chinese Traditional (CNS)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20001	TCA Taiwan	Extended codepage; for processing rules, see section 3.1.5.1.1.
20002	Eten Taiwan; Chinese Traditional (Eten)	Extended codepage; for processing rules, see section 3.1.5.1.1.

Codepage ID	Codepage Descriptions	Codepage Notes
20003	IBM5550 Taiwan	Extended codepage; for processing rules, see section 3.1.5.1.1.
20004	TeleText Taiwan	Extended codepage; for processing rules, see section 3.1.5.1.1.
20005	Wang Taiwan	Extended codepage; for processing rules, see section 3.1.5.1.1.
20105	IA5 (IRV International Alphabet No. 5, 7-bit); Western European (IA5)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20106	IA5 German (7-bit)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20107	IA5 Swedish (7-bit)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20108	IA5 Norwegian (7-bit)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20127	US-ASCII (7-bit)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20261	T.61	Extended codepage; for processing rules, see section 3.1.5.1.1.
20269	ISO 6937 Non-Spacing Accent	Extended codepage; for processing rules, see section 3.1.5.1.1.
20273	IBM EBCDIC Germany	Extended codepage; for processing rules, see section 3.1.5.1.1.
20277	IBM EBCDIC Denmark-Norway	Extended codepage; for processing rules, see section 3.1.5.1.1.
20278	IBM EBCDIC Finland-Sweden	Extended codepage; for processing rules, see section 3.1.5.1.1.
20280	IBM EBCDIC Italy	Extended codepage; for processing rules, see section 3.1.5.1.1.
20284	IBM EBCDIC Latin America-Spain	Extended codepage; for processing rules, see section 3.1.5.1.1.
20285	IBM EBCDIC United Kingdom	Extended codepage; for processing rules, see section 3.1.5.1.1.
20290	IBM EBCDIC Japanese Katakana Extended	Extended codepage; for processing rules, see section 3.1.5.1.1.
20297	IBM EBCDIC France	Extended codepage; for processing rules, see section 3.1.5.1.1.
20420	IBM EBCDIC Arabic	Extended codepage; for processing rules, see section 3.1.5.1.1.

Codepage ID	Codepage Descriptions	Codepage Notes
20423	IBM EBCDIC Greek	Extended codepage; for processing rules, see section 3.1.5.1.1.
20424	IBM EBCDIC Hebrew	Extended codepage; for processing rules, see section 3.1.5.1.1.
20833	IBM EBCDIC Korean Extended	Extended codepage; for processing rules, see section 3.1.5.1.1.
20838	IBM EBCDIC Thai	Extended codepage; for processing rules, see section 3.1.5.1.1.
20866	Russian (KOI8-R); Cyrillic (KOI8-R)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20871	IBM EBCDIC Icelandic	Extended codepage; for processing rules, see section 3.1.5.1.1.
20880	IBM EBCDIC Cyrillic Russian	Extended codepage; for processing rules, see section 3.1.5.1.1.
20905	IBM EBCDIC Turkish	Extended codepage; for processing rules, see section 3.1.5.1.1.
20924	IBM EBCDIC Latin 1/Open System (1047 + Euro symbol)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20932	Japanese (JIS 0208-1990 and 0121-1990)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20936	Simplified Chinese (GB2312); Chinese Simplified (GB2312-80)	Extended codepage; for processing rules, see section 3.1.5.1.1.
20949	Korean Wansung	Extended codepage; for processing rules, see section 3.1.5.1.1.
21025	IBM EBCDIC Cyrillic Serbian-Bulgarian	Extended codepage; for processing rules, see section 3.1.5.1.1.
21027	Ext Alpha Lowercase	Extended codepage; for processing rules, see section 3.1.5.1.1. NOTE: Although this code page is supported, it has no known use.
21866	Ukrainian (KOI8-U); Cyrillic (KOI8-U)	Extended codepage; for processing rules, see section 3.1.5.1.1.
28591	ISO 8859-1 Latin 1; Western European (ISO)	Extended codepage; for processing rules, see section 3.1.5.1.1.
28592	ISO 8859-2 Central European; Central European (ISO)	Extended codepage; for processing rules, see section 3.1.5.1.1.
28593	ISO 8859-3 Latin 3	Extended codepage; for processing rules, see section 3.1.5.1.1.
28594	ISO 8859-4 Baltic	Extended codepage; for processing rules, see section 3.1.5.1.1.

Codepage ID	Codepage Descriptions	Codepage Notes
28595	ISO 8859-5 Cyrillic	Extended codepage; for processing rules, see section 3.1.5.1.1.
28596	ISO 8859-6 Arabic	Extended codepage; for processing rules, see section 3.1.5.1.1.
28597	ISO 8859-7 Greek	Extended codepage; for processing rules, see section 3.1.5.1.1.
28598	ISO 8859-8 Hebrew; Hebrew (ISO-Visual)	Extended codepage; for processing rules, see section 3.1.5.1.1.
28599	ISO 8859-9 Turkish	Extended codepage; for processing rules, see section 3.1.5.1.1.
28603	ISO 8859-13 Estonian	Extended codepage; for processing rules, see section 3.1.5.1.1.
28605	ISO 8859-15 Latin 9	Extended codepage; for processing rules, see section 3.1.5.1.1.
38598	ISO 8859-8 Hebrew; Hebrew (ISO-Logical)	Extended codepage; for processing rules, see section 3.1.5.1.1. Use [CODEPAGEFILES] 28598.txt.
50220	ISO 2022 Japanese with no halfwidth Katakana; Japanese (JIS)	Extended codepage; for processing rules, see section 3.1.5.1.1.
50221	ISO 2022 Japanese with halfwidth Katakana; Japanese (JIS-Allow 1 byte Kana)	Extended codepage; for processing rules, see section 3.1.5.1.2.
50222	ISO 2022 Japanese JIS X 0201-1989; Japanese (JIS-Allow 1 byte Kana - SO/SI)	Extended codepage; for processing rules, see section 3.1.5.1.2.
50225	ISO 2022 Korean	Extended codepage; for processing rules, see section 3.1.5.1.2.
50227	ISO 2022 Simplified Chinese; Chinese Simplified (ISO 2022)	Extended codepage; for processing rules, see section 3.1.5.1.2.
50229	ISO 2022 Traditional Chinese	Extended codepage; for processing rules, see section 3.1.5.1.2.
51949	EUC Korean	Extended codepage; for processing rules, see section 3.1.5.1.2. Use [CODEPAGEFILES] 20949.txt.
52936	HZ-GB2312 Simplified Chinese; Chinese Simplified (HZ)	Extended codepage; for processing rules, see section 3.1.5.1.2.
54936	GB18030 Simplified Chinese (4 byte); Chinese Simplified (GB18030)	Extended codepage; for processing rules, see section 3.1.5.1.3.
57002	ISCII Devanagari	Extended codepage; for processing rules, see section 3.1.5.1.4.

Codepage ID	Codepage Descriptions	Codepage Notes
57003	ISCII Bengali	Extended codepage; for processing rules, see section 3.1.5.1.4.
57004	ISCII Tamil	Extended codepage; for processing rules, see section 3.1.5.1.4.
57005	ISCII Telugu	Extended codepage; for processing rules, see section 3.1.5.1.4.
57006	ISCII Assamese	Extended codepage; for processing rules, see section 3.1.5.1.4.
57007	ISCII Oriya	Extended codepage; for processing rules, see section 3.1.5.1.4.
57008	ISCII Kannada	Extended codepage; for processing rules, see section 3.1.5.1.4.
57009	ISCII Malayalam	Extended codepage; for processing rules, see section 3.1.5.1.4.
57010	ISCII Gujarati	Extended codepage; for processing rules, see section 3.1.5.1.4.
57011	ISCII Punjabi	Extended codepage; for processing rules, see section 3.1.5.1.4.
65000	Unicode (UTF-7)	Extended codepage; for processing rules, see section 3.1.5.1.5.
65001	Unicode (UTF-8)	Extended codepage; for processing rules, see section 3.1.5.1.6.

2.2.2 Supported Codepage Data Files

The mapping of UTF-16 strings to earlier codepages relies on codepage data files to provide conversion data. These codepage data files map Unicode characters to characters in a **single-byte character set (SBCS)** or **double-byte character set (DBCS)**.

The data files of supported system codepages are published as specified in [CODEPAGEFILES], [UNICODE], and [UNICODE-BESTFIT]. The location identification uses a simple file-naming convention, which is bestfitxxxx.txt, where xxxx is the codepage number. For example, bestfit950.txt contains the data for codepage 950, and bestfit1252.txt contains the data for codepage 1252.

The pseudocode assumes all these codepage files are available.

2.2.2.1 Codepage Data File Format

The Readme.txt (as specified in [UNICODE-README]) provides details about the codepages files and the file format. This section specifies information about the pseudocode of mapping UTF-16 strings to earlier codepages by taking the content from the Readme.txt.

Each file has sections of keyword tags and records. Any text after ";" is ignored as blank lines. Fields are delimited by one or more space or tab characters. Each section begins with one of the following tags:

- CODEPAGE
- CPINFO
- MBTABLE
- WCTABLE
- DBCSRANGE (DBCS codepages only)
- DBSCTABLE (DBCS codepages only)

2.2.2.1.1 WCTABLE

The WCTABLE tag marks the start of the mapping from Unicode UTF-16 to MultiByte bytes. It has one field.

Field 1: The number of records of Unicode to byte mappings. Note that this field is often more than the number of roundtrip mappings that are supported by the codepage due to Microsoft Windows® best-fit behavior.

An example of the WCTABLE tag is:

```
WCTABLE 698
```

The Unicode UTF-16 mapping records follow the WCTABLE section. These mapping records are in two forms: single-byte or double-byte codepages. Both forms have two fields.

Field 1: The Unicode UTF-16 code point for the character being converted.

Field 2: The single byte that this UTF-16 code point maps to. This can be a best-fit mapping.

The following example shows Unicode to byte-mapping records for SBCSs.

```
0x0000 0x00; Null
0x0001 0x01; Start Of Heading
...
0x0061 0x61; Latin Small Letter A
0x0062 0x62; Latin Small Letter B
0x0063 0x63; Latin Small Letter C
...
0x221e 0x38; Infinity                        << Best Fit Mapping
...
0xff41 0x61; Fullwidth Latin Small Letter A  << Best Fit Mapping
0xff42 0x62; Fullwidth Latin Small Letter B  << Best Fit Mapping
0xff43 0x63; Fullwidth Latin Small Letter C  << Best Fit Mapping
...
```

Field 1: The Unicode UTF-16 code point for the character being converted.

Field 2: The byte or bytes that this code point maps to as a 16-bit value. The high byte is the lead byte, and the low byte is the trail byte. If the high byte is 0, this is a single-byte code point with the value of the low byte and no lead byte is emitted.

The following example shows Unicode to byte-mapping records for DBCSs.

```
0x0000 0x0000; Null
0x0001 0x0001; Start Of Heading
...
0x0061 0x0061; a
0x0062 0x0062; b
0x0063 0x0063; c
...
0x221e 0x8187; Infinity
...
0xff41 0x8281; Fullwidth a
0xff42 0x8282; Fullwidth b
0xff43 0x8283; Fullwidth c
...
```

2.2.2.1.2 MBTABLE

The MBTABLE tag marks the start of the mapping from Single-byte bytes to Unicode UTF-16. It has one field.

Field 1: The number of records of single-byte to Unicode mappings.

An example of the MBTABLE tag is:

```
MBTABLE 196
```

The Unicode UTF-16 mapping records follow the MBTABLE section. These mapping records have two fields.

Field 1: The single byte character of the codepage.

Field 2: The Unicode UTF-16 code point that the codepage char maps to.

The following example shows mapping records for codepage 932.

```
0x00 0x0000; Null
0x01 0x0001; Start Of Heading
0x02 0x0002; Start Of Text
0x03 0x0003; End Of Text
0x04 0x0004; End Of Transmission
0x05 0x0005; Enquiry
0x06 0x0006; Acknowledge
0x07 0x0007; Bell
0x08 0x0008; Backspace
…
0xa1 0xff61; Halfwidth Ideographic Period
0xa2 0xff62; Halfwidth Opening Corner Bracket
0xa3 0xff63; Halfwidth Closing Corner Bracket
0xa4 0xff64; Halfwidth Ideographic Comma
```

```
0xa5 0xff65; Halfwidth Katakana Middle Dot
0xa6 0xff66; Halfwidth Katakana Wo
0xa7 0xff67; Halfwidth Katakana Small A
0xa8 0xff68; Halfwidth Katakana Small I
0xa9 0xff69; Halfwidth Katakana Small U
0xaa 0xff6a; Halfwidth Katakana Small E
0xab 0xff6b; Halfwidth Katakana Small O
0xac 0xff6c; Halfwidth Katakana Small Ya
```

2.2.2.1.3 DBCSRANGE

The DBCSRANGE tag marks the start of the mapping from double-byte bytes to Unicode UTF-16. It has one field.

Field 1: The number of records of lead byte ranges.

An example of the DBCSRANGE tag is:

```
DBCSRANGE 2
```

The Lead Byte Range records follow the DBCSRANGE section. These mapping records have two fields.

Field 1: The start of lead byte range.

Field 2: The end of lead byte range.

The following example shows one of the Lead Byte Range records for codepage 932. In this codepage, it has one range of lead byte, starting from 0x81 (decimal 129) to 0x9f (decimal 159). So there are 31 lead bytes in this example (159 – 129 + 1). Each lead byte will have a corresponding DBCSRANGE.

```
0x81 0x9f; Lead Byte Range
```

A group of DBCSTABLE sections follows the lead-byte range record. Each lead byte will have a corresponding DBCSTABLE section. In each DBCSTABLE section, it has one field.

Field 1: This field is the number of trail byte mappings following.

The lead byte of the first DBCSTABLE is the first lead byte of the previous Lead Byte Range record. Each subsequent DBCSTABLE is for the next consecutive lead byte value.

The following example shows the first DBCSTABLE for codepage 932. This is for lead byte 0x81.

```
DBCSTABLE 147; LeadByte = 0x81
```

The DBCSTABLE record describes the mappings available for a particular lead byte. The comment is ignored but descriptive.

Field 1: This field is the trail byte to map from.

Field 2: This field is the Unicode UTF-16 code point that this lead byte/trail byte combination map to.

The following example shows DBCSTABLE records for codepage 932 for lead byte 0x81.

```
0x40 0x3000; Ideographic Space
0x41 0x3001; Ideographic Comma
...
```

3 Protocol Details

The following sections specify details of the Windows Protocols Unicode Reference, including abstract data models and message processing rules.

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this document.

No abstract data model is needed.

3.1.2 Timers

There are no timers.

3.1.3 Initialization

There is no initialization.

3.1.4 Higher-Layer Triggered Events

There are no higher-layer triggered events.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Mapping Between UTF-16 Strings and Legacy Codepages

3.1.5.1.1 Mapping Between UTF-16 Strings and Legacy Codepages Using CodePage Data File

This process maps between a Unicode string that is encoded in UTF-16 and a string in a specified codepage using a codepage data file described in 2.2.2.1.

3.1.5.1.1.1 Pseudocode for Accessing a Record in the Codepage Data File

This section contains the pseudocode that is used to read information from the codepage file. The following example is taken from codepage data file 950.txt.

"OPEN SECTION" indicates that queries for records in a specific section are made. To open the following section with the WCTABLE label, the following syntax is used. The "OPEN SECTION" is accessible by using the "WideCharMapping" name.

```
OPEN SECTION WideCharMapping
     where section name is WCTABLE from bestfit950.txt
```

"SELECT RECORD" assigns a line from the data file to be referenced by the assigned variable name. For example, the following code selects a record from the WideCharMapping section, and the record is accessible by using the "MappingData" name.

```
SET UnicodeChar to 0x4e00
SELECT RECORD MappingData from WideCharMapping
     where field 1 matches UnicodeChar
```

Following example will select the line.

```
0x4e00 0xa440
```

Values from selected records are referenced by field number. The following example selects the individual data fields from the selected row.

```
SET MultiByteResult to MappingData.Field2
```

In this example, the value of MultiByteResult is the hexadecimal value 0xa440.

```
CODEPAGE 950            ; Chinese (Taiwan, Hong Kong SAR) - ANSI, OEM
CPINFO 2 0x3f 0x003f    ; DBCS CP, Default Char = Question Mark
...
WCTABLE 20321
0x0000 0x0000;   Null
0x0001 0x0001;   Start Of Heading
0x0002 0x0002;   Start Of Text
0x0003 0x0003;   End Of Text
0x0004 0x0004;   End Of Transmission
0x0005 0x0005;   Enquiry
...
0x4e00 0xa440
0x4e01 0xa442
0x4e03 0xa443
0x4e07 0xc94
```

3.1.5.1.1.2 Pseudocode for Mapping a UTF-16 String to a Codepage String

```
COMMENT  This algorithm maps a Unicode string encoded in UTF-16 to a
string in the specified ANSI codepage. The supported ANSI codepages
are limited to those that can be set as system codepage.

It requires the following externally specified values:

1) CodePage: An integer value to represent an ANSI codepage value.

   If CodePage value is CP_ACP (0), the system default ANSI codepage from
   the OS should be used.
   If CodePage value is CP_OEMCP (1), the sysstem default OEM codepage from
   the OS should be used
```

```
2) UnicodeString: A string encoded in UTF-16. Every Unicode code point
   is an unsigned 16-bit ("WORD") value. Surrogate pair is not
   supported in this algorithm.

3) UnicodeStringLength: The string length in 16-bit ("WORD") unit for
   UnicodeString. When UnicodeStringLength is 0, the length is
   decided by counting from the beginning of the string to a NULL
   character (Unicode value U+0000), including the null character.

4) MultiByteString: A string encoded in ANSI codepage. Every
   character can be an 8-bit (byte) unsigned value or two 8-bit
   unsigned values.

5) MultiByteStringLength: The length in bytes. This should include
   the byte for NULL terminator. When MultiByteStringLength is 0,
   the MultiByteString value will not be used in this algorithm.
   Instead, the length of the result string in ANSI codepage will be
   returned.

6) lpDefaultChar
   Optional. Point to the byte to use if a character cannot be represented in
   the specified code page. The application sets this parameter to NULL if
   the function is to use a system default value. The common default value is
   0x3f, which is the ASCII value for the question mark.


PROCEDURE WideCharToMultiByteFromCodepageDataFile

IF CodePage is CP_ACP THEN
    COMMENT Windows operating system keeps a systemwide value of
            default ANSI system codepage. It is used to provide a default
    COMMENT system codepage to be used by legacy ANSI application.

    SET CodePage to the default ANSI system codepage from the Windows
            operating system.
ELSE IF CodePage is CP_OEMCP THEN
    COMMENT Windows keeps a systemwide value of
            default OEM system codepage. It is used to provide a default
    COMMENT system codepage to be used by legacy console application.

    SET CodePage to the default OEM system codepage from Windows.

ENDIF

IF UnicodeStringLength is 0 THEN
    COMPUTE UnicodeStringLength as the string length in 16-bit units
            of UnicodeString as a NULL-terminated string, including
            NULL terminator.
ENDIF

IF MultiByteStringLength is 0 THEN
    SET IsCountingOnly to True
ELSE
    SET IsCountingOnly to False
ENDIF


SET ResultMultiByteLength to 0
```

```
SET CodePageFileName to the concatenation of strings "Bestfit",
    CodePage as a string, and ".txt"

IF lpDefaultChar is null THEN
    COMMENT No default char is specified by the caller. Read the default
    COMMENT char from CPINFO in the data file

    OPEN SECTION CharacterInfo where section name is CPINFO
    from file with the name of CodePageFileName
    SET lpDefaultChar to CharacterInfo.Field3
ENDIF


OPEN SECTION WideCharMapping where section name is WCTABLE from file
    with the name of CodePageFileName

FOR each Unicode codepoint UnicodeChar in UnicodeString
     SELECT MappingData from WideCharMapping
            where field 1 matches UnicodeChar
     IF MappingData is null THEN
         COMMENT There is no mapping for this Unicode character, use
         COMMENT the default character
         IF IsCountingOnly is False THEN
             SET MultiByteString[ResultMultiByteLength]
                 to lpDefaultChar
         ENDIF
         INCREMENT ResultMultiByteLength
         CONTINUE FOR loop
     ENDIF

     SET MultiByteResult to MappingData.Field2

     IF MultiByteResult is less than 256 THEN
          COMMENT This is a single byte result
          IF IsCountingOnly is True THEN
               INCREMENT ResultMultiByteLength
          ELSE
               SET MultiByteString[ResultMultiByteLength]
                   to MultiByteResult
               INCREMENT ResultMultiByteLength
          ENDIF
     ELSE
          COMMENT This is a double byte result
          IF IsCountingOnly is True THEN
               COMPUTE ResultMultiByteLength as
                       ResultMultiByteLength added by 2
          ELSE
               SET MultiByteString[ResultMultiByteLength] to
                   MultiByteResult divided by 256
               INCREMENT ResultMultiByteLength
               SET MultiByteString[ResultMultiByteLength] to
                   the remainder of MultiByteResult divided by 256
               INCREMENT ResultMultiByteLength
          ENDIF
     ENDIF
END FOR

RETURN ResultMultiByteLength as a 32-bit unsigned integer
```

3.1.5.1.1.3 Pseudocode for Mapping a Codepage String to a UTF-16 String

```
COMMENT  This algorithm maps a Unicode string encoded in the specified codepage to UTF-16.

It requires the following externally specified values:

1) CodePage: An integer value to represent an ANSI codepage value.

   If CodePage value is CP_ACP (0), the system default ANSI codepage from
   the OS should be used.
   If CodePage value is CP_OEMCP (1), the system default OEM codepage from
   the OS should be used

2) MultiByteString: A string encoded in ANSI codepage. Every
   character can be an 8-bit (byte) unsigned value or two 8-bit
   unsigned values.

3) MultiByteStringLength: The length in bytes. This should include
   the byte for terminating null character. When MultiByteStringLength is 0,    the length is
   decided by counting from the beginning of the string to a null
   character (0x00), including the null character.

4) UnicodeString: A string encoded in UTF-16. Every Unicode code point
   is an unsigned 16-bit ("WORD") value. Surrogate pair is not
   supported in this algorithm.

5) UnicodeStringLength: The string length in 16-bit ("WORD") unit for
   UnicodeString. When UnicodeStringLength is 0,
   the UnicodeString value will not be used in this algorithm.
   Instead, the length of the result string in UTF-16 will be
   returned.


PROCEDURE MultiByteToWideCharFromCodepageDataFile

IF CodePage is CP_ACP THEN
    COMMENT Windows keeps a systemwide value of
            default ANSI system codepage. It is used to provide a default
    COMMENT system codepage to be used by legacy ANSI application.

    SET CodePage to the default ANSI system codepage from Windows.

ELSE IF CodePage is CP_OEMCP THEN
    COMMENT Windows keeps a systemwide value of
            default OEM system codepage. It is used to provide a default
    COMMENT system codepage to be used by legacy console application.

    SET CodePage to the default OEM system codepage from Windows.


ENDIF

IF MultiByteStringLength is 0 THEN
    COMPUTE UnicodeStringLength as the string length in 8-bit units
            of MultiByteString as a null-terminated string, including
            terminating null character.
ENDIF
```

```
IF UnicodeStringLength is 0 THEN
    SET IsCountingOnly to True
ELSE
    SET IsCountingOnly to False
ENDIF

SET CodePageFileName to the concatenation of
CodePage as a string, and ".txt"

OPEN SECTION CodePageInfo where section name is CPINFO from file
    with the name of CodePageFileName
COMMENT Read the code page type.
COMMENT The value for Single Byte Code Page (SBCS) is 1
COMMENT The value for Double Byte Code Page (DBCS) is 2

SET CodePageType to CodePageInfo.Field1
SET DefaultUnicodeChar to CodePageInfo.Field3

OPEN SECTION SingleByteMapping where section name is MBTABLE from file
    with the name of CodePageFileName

SET MultiByteIndex = 0
WHILE MultiByteIndex <= to MultiByteStringLength - 1
     SET MultiByteChar = MultiByteString[MultiByteIndex]
     IF CodePageType is 1 THEN
         COMMENT SBCS codepage
         COMMENT Select a record which contains the mapping data
         SELECT MappingData from SingleByteMapping
            where field 1 matches MultiByteChar
         IF MappingData is null THEN
             COMMENT There is no mapping for this single-byte character, use
             COMMENT the default character
             IF IsCountingOnly is False THEN
                 SET MultiByteString[ResultUnicodeLength]
                     to DefaultUnicodeChar
             ENDIF
             INCREMENT ResultMultiByteLength
             INCREMENT MultiByteIndex
             CONTINUE WHILE loop
         ENDIF
         IF IsCountOnly is False THEN
             SET UnicodeString[ResultUnicodeLength]
                   to MappingData.Field2
         ENDIF
         INCREMENT ResultUnicodeLength
    ELSE
         COMMENT DBCS codepage
         COMMENT First, try if this is a single-byte mapping
         SELECT MappingData from SingleByteMapping
            where field 1 matches MultiByteChar
         IF MappingData is not null THEN
             COMMENT This byte is a single-byte character
             IF IsCountOnly is False THEN
                 SET UnicodeString[ResultUnicodeLength]
                     to MappingData.Field2
             ENDIF
             INCREMENT ResultUnicodeLength
         ELSE
             COMMENT Not a single-byte character
```

```
COMMENT Check if this is a valid lead byte for double byte mapping
OPEN SECTION DBCSRanges
    where section name is DBCSRANGE from file
    with the name of CodePageFileName

COMMENT Read the count of DBCS Range count
SET DBCSRangeCount to DBCSRanges.Field1

SET ValidDBCS to False
COMMENT Enumerate through every DBCSRange record to see if
COMMENT the MultiByteChar is a leading byte

FOR Counter i = 1 to DBCSRangeCount
    COMMENT Select the current record
    SELECT DBCSRangeRecord from DBCSRanges
    SET LeadByteStart to DBCSRangeRecord.Field1
    SET LeadByteEnd to DBCSRangeRecord.Field2
    IF MultiByteChar is larger or equal to LeadByteStart AND
       MultiByteChar is less or equal to LeadByteEnd THEN
        COMMENT This is a valid lead byte
        COMMENT Now check if there is a following valid trailing byte
        SET LeadByteTableCount = MultiByteChar - LeadByteStart

        COMMENT Select the current DBCSTABLE section
        OPEN SECTION DBCSTableSection from DBCSRanges
           where section name is DBCSTABLE
        COMMMENT Advance to the right DBCSTABLE section
        FOR LeadByteIndex = 0 to LeadByteTableCount
            ADVANCE SECTION DBCSTableSection
        NEXTFOR
        COMMENT Check if the trailing byte is valid
        IF MultiByteIndex + 1 is less than MultiByteStringLength THEN
            SET TrailByteChar to MultiByteString[MultiByteIndex + 1]
            SELECT MappingData FROM DBCSTABLE
                Where field 1 matches TrailgByteChar
            IF MappingData is not null THEN
                COMMENT Valid trailing byte
                SET ValidDBCS to True
                IF IsCountingOnly is FALSE THEN
                    SET UnicodeString[ResultUnicodeLength] to MappingData.Field2
                ENDIF
                INCREMENT ResultUnicodeLength
                COMMENT Increment the MultiByteIndex.
                COMMENT Note that the MultiByteIndex will
                COMMENT be incremented again for the WHILE loop
                INCREMENT MultiByteIndex
                EXIT FOR loop
            ENDIF
        ENDIF
    ENDIF
COMMENT No valid lead byte is found. Advance to next record
ADVANCE DBCSRangeRecord
NEXTFOR
IF ValidDBCS is FALSE THEN
    COMMENT There is no valid leading byte/trailing byte sequence
    If IsCountingOnly is FALSE THEN
        SET UnicodeString[ResultUnicodeLength] to DefaultUnicodeChar
    ENDIF
    INCREMENT MultiByteIndex
```

```
                INCREMENT ResultUnicodeLength
            ENDIF
        ENDIF
    ENDIF
    INCREMENT MultiByteIndex
ENDWHILE

RETURN ResultMultiByteLength as a 32-bit unsigned integer
```

3.1.5.1.2 Mapping Between UTF-16 Strings and ISO 2022-Based Codepages

[ECMA-035] defines the standard that is fully identical with International Standard ISO/IEC 2022:1994. EUC (Extended Unix Code) is based on ISO-2022 standard.

For more information, see [ECMA-035].

3.1.5.1.3 Mapping between UTF-16 Strings and GB 18030 Codepage

Windows implements GB-18030 based on [GB18030].

For more information, please see [GB18030].

3.1.5.1.4 Mapping Between UTF-16 Strings and ISCII Codepage

Microsoft Windows® implements ISCII-based codepage based on [ISCII].

For more information, see [ISCII].

3.1.5.1.5 Mapping Between UTF-16 Strings and UTF-7

Microsoft Windows® implements UTF-7 codepage based on [RFC2152].

For more information, see [RFC2152].

3.1.5.1.6 Mapping Between UTF-16 Strings and UTF-8

Microsoft Windows® implements UTF-8 codepage based on [UNICODE5.0.0/CH3].

For more information, see [UNICODE5.0.0/CH3].

3.1.5.2 Comparing UTF-16 Strings by Using Sort Keys

To compare strings, a **sort key** is needed for each string. A binary comparison of the sort keys can then be used to arrange the strings in any order.

3.1.5.2.1 Pseudocode for Comparing UTF-16 Strings

This algorithm compares two UTF-16 strings by using linguistically appropriate rules.

```
This algorithm compares two Unicode strings using linguistic
appropriate rules. It requires the following externally specified
values:

     1) StringA: A string encoded in UTF-16
```

```
     2) StringB: A string encoded in UTF-16

CALL GetWindowsSortKey
     WITH StringA
     RETURNING SortKeyA
CALL GetWindowsSortKey
     WITH StringB
     RETURNING SortKeyB

CALL CompareSortKeys
     WITH SortKeyA, SortKeyB
     RETURNING Result

IF Result is "SortKeyA is equal to SortKeyB" THEN
     StringA is considered equal to StringB
ELSE IF Result is "SortKeyA is less than SortKeyB" THEN
     StringA is sorted prior to StringB
ELSE
     StringA is sorted after StringB
ENDIF
```

3.1.5.2.2 CompareSortKey

This algorithm generates sort keys for two strings and uses the sort keys to provide a linguistically appropriate string comparison.

```
COMMENT CompareSortKeys
COMMENT  On Entry: SortKeyA - An array of bytes returned from
COMMENT                       GetWindowsSortKey
COMMENT            SortKeyB - An array of bytes returned from
COMMENT                       GetWindowsSortKey
COMMENT
COMMENT  On Exit:  Result   - A value indicating if SortKeyA
COMMENT                       is less than, equal to, or greater
COMMENT                       than SortKeyB

PROCEDURE CompareSortKeys

SET index to 0
WHILE index is less than Length(SortKeyA) and
     index is also less than Length(SortKeyB)

     IF SortKeyA[index] is less than SortKeyB[index] THEN
          SET Result to "SortKeyA is less than SortKeyB"
          RETURN
     ENDIF
     IF SortKeyA[index] is greater than SortKeyB[index] THEN
          SET Result to "SortKeyA is greater than SortKeyB"
          RETURN
     ENDIF

INCREMENT index
ENDWHILE

IF Length(SortKeyA) is equal to Length(SortKeyB) THEN
     SET Result to "SortKeyA is equal to SortKeyB"
ELSE IF Length(SortKeyA) is less than Length(SortKeyB) THEN
```

```
     SET Result to "SortKeyA is less than SortKeyB"
ELSE
     assert Length(SortKeyA) must be greater than Length(SortKeyB)
     SET Result to "SortKeyA is greater than SortKeyB"
ENDIF
RETURN
```

Any sorting mechanism may be used to arrange these strings by comparing their sort keys.

3.1.5.2.3 Accessing the Windows Sorting Weight Table

Microsoft Windows® gets its sorting data from a data table (see Appendix B (section 7)). Code points are labeled by using UTF-16 values. The file is arranged in sections of tab-delimited field records. Optional comments begin with a semicolon. Each section contains a label and can have a subsection label.

```
SORTKEY                - Section label
   DEFAULT 52086       - Subsection label with record count

0x0001   6   3   2   2; Start Of Heading, U+0001 char record
0x0002   6   4   2   2; Start Of Text
0x0003   6   5   2   2; End Of Text
...
0x00411  4   2   2  18; Latin Capital Letter A, U+0041 char record
0x00421  4   9   2  18; Latin Capital Letter B
0x00431  4  10   2  18; Latin Capital Letter C
   |     |   |   |   |       |
Field 1  2   3   4   5    Comment
```

Note that labels are any field that does not begin with a numerical (0xNNNN) value. Blank lines and characters that follow a ";" are ignored.

This document uses the following notation to specify the processing of the file.

"OPEN" indicates that queries are made for records in a specific section. To open the preceding section with the SORTKEY label and DEFAULT sublabel, the following syntax is used. The OPEN SECTION is accessible by using the "DefaultTable" name.

```
OPEN SECTION DefaultTable where name is
     SORTKEY\DEFAULT from unisort.txt
```

"SELECT" assigns a line from the data file to be referenced by the assigned variable name. To select the highlighted row preceding, this document uses this notation. The selected row is accessible by using the name "CharacterRow".

```
SET UnicodeChar to 0x0041
SELECT RECORD CharacterRow FROM DefaultTable
     WHERE field 1 matches UnicodeChar
```

Values from selected records are referenced by field number. The following would select the individual data fields from the selected row.

```
SET CharacterWeight.ScriptMember to CharacterRow.Field2
SET CharacterWeight.PrimaryWeight to CharacterRow.Field3
SET CharacterWeight.DiacriticWeight to CharacterRow.Field4
SET CharacterWeight.CaseWeight to CharacterRow.Field5
```

Some sections of the data file are referenced by a locale language code identifier (LCID). For more information, see [MS-LCID].

```
SORTTABLES
...
   COMPRESSION   19          - 19 Locales have contractions
   LCID 0x0000041a; Croatian
   TWO  9                    - 9 Records in this subsection
0x0064  0x017e  14  29  4  2 ;d z Hacek
0x0044  0x017e  14  29  4  18;D z Hacek
0x0044  0x017d  14  29  4  26;D Z Hacek
...
   LCID 0x00000405; Czech
   TWO  3                    - Czech as 3 TWO character contractions
0x0063  0x0068  14  46  2  2 ;ch
0x0043  0x0068  14  46  2  18;Ch
0x0043  0x0048  14  46  2  26;CH
   |      |     |   |   |  |  |
Field 1   2     3   4   5  6  Comment
```

To select the record for characters 0x0043 and 0x0068 with LCID 0x0405, the following notation is used.

```
SET Character1 to 0x0043
SET Character2 to 0x0068
SET SortLocale to 0x0405

OPEN SECTION ContractionTable where name is
     SORTTABLES\COMPRESSION\LCID[SortLocale]\TWO from unisort.txt
SELECT RECORD ContractionRow FROM ContractionTable WHERE field 1
     matches Character1 and field 2 matches Character2
SET CharacterWeight.ScriptMember to ContractionRow.Field3
SET CharacterWeight.PrimaryWeight to ContractionRow.Field4
SET CharacterWeight.DiacriticWeight to ContractionRow.Field5
SET CharacterWeight.CaseWeight to ContractionRow.Field6
```

3.1.5.2.4 GetWindowsSortKey Pseudocode

This algorithm specifies the generation of sort keys for a specific UTF-16 string.

```
STRUCTURE CharacterWeightType
(
     ScriptMember:    8 bit integer
```

```
     PrimaryWeight:   8 bit integer
     DiacriticWeight: 8 bit integer
     CaseWeight:      8 bit integer
)

STRUCTURE UnicodeWeightType
(
     ScriptMember:    8 bit integer
     PrimaryWeight:   8 bit integer
     ThirdByteWeight: 8 bit integer
)

STRUCTURE SpecialWeightType
(
     Position:       16 bit integer
     ScriptMember:    8 bit integer
     PrimaryWeight:   8 bit integer
)

STRUCTURE ExtraWeightType
(
     W6:              8 bit integer
     W7:              8 bit integer
)
SET constant LCID_KOREAN to 0x0412
SET constant LCID_KOREAN_UNICODE_SORT to 0x010412
SET constant LCID_HUNGARIAN to 0x040e

SET constant SORTKEY_SEPARATOR to 0x01
SET constant SORTKEY_TERMINATOR to 0x00

SET global KoreanScriptMap to InitKoreanScriptMap

//
//  Script Member Values.
//
SET constant UNSORTABLE       to 0
SET constant NONSPACE_MARK    to 1
SET constant EXPANSION        to 2
SET constant EASTASIA_SPECIAL to 3
SET constant JAMO_SPECIAL     to 4
SET constant EXTENSION_A      to 5
SET constant PUNCTUATION      to 6

SET constant SYMBOL_1         to 7
SET constant SYMBOL_2         to 8
SET constant SYMBOL_3         to 9
SET constant SYMBOL_4         to 10
SET constant SYMBOL_5         to 11
SET constant SYMBOL_6         to 12

SET constant DIGIT            to 13

SET constant LATIN            to 14
SET constant KANA             to 34
SET constant IDEOGRAPH        to 128

IF Windows version is Windows Vista, Windows Server 2008, Windows 7, or
   Windows Server 2008 R2 THEN
```

```
SET constant MAX_SPECIAL_CASE to SYMBOL_6

ELSE
SET constant MAX_SPECIAL_CASE to SYMBOL_5
ENDIF

IF Windows version is Windows Server 2008 R2 or Windows 7 THEN
    COMMENT Set the constant for fhe first script member of the Unicode
    COMMENT Private Use Area (PUA) range
    SET constant PUA3BYTESTART to 0xA9
    COMMENT Set the constant for the last script member of the Unicode
    COMMENT Private Use Area (PUA) range
    SET constant PUA3BYTEEND to 0xAF

    COMMENT Set the constant for the first script member of CJK
    COMMENT(Chinese/Japanese/Korean) 3 byte weight range
    SET constant CJK3BYTESTART to 0xC0
    COMMMENT Set the constant for the last script member of CJK
    COMMENT (Chinese/Japanese/Korean) 3 byte weight range
    SET constant CJK3BYTEEND to 0xEF
ENDIF


SET constant FIRST_SCRIPT     to LATIN
SET constant MAX_SCRIPTS      to 256

//
//  Values for CJK Unified Ideographs Extension A range.
//    0x3400 thru 0x4dbf
//
SET constant SCRIPT_MEMBER_EXT_A  to 254       // SM for Extension A
SET constant PRIMARY_WEIGHT_EXT_A to 255       // AW for Extension A

//
//  Lowest weight values.
//  Used to remove trailing DW and CW values.
//  Also used to keep illegal values out of sort keys.
//

SET constant MIN_DW to 2
SET constant MIN_DW to 2

//
//  Bit mask values.
//
//  Case Weight (CW) - 8 bits:
//    bit 0   => width
//    bit 1,2 => small kana, sei-on
//    bit 3,4 => upper/lower case
//    bit 5   => kana
//    bit 6,7 => contraction
//

IF Windows version is Windows Server 2008 R2 or Windows 7 THEN
    COMMENT Windows Server 2008 R2 and Windows 7 supports up to 8-character
    COMMENT contraction
    COMMENT Set the necessary constants for the support
    SET constant CONTRACTION_8_MASK to 0xc0
    SET constant CONTRACTION_7_MASK to 0xc0
```

```
    SET constant CONTRACTION_6_MASK to 0xc0
    SET constant CONTRACTION_5_MASK to 0x80
    SET constant CONTRACTION_4_MASK to 0x80
    SET constant CONTRACTION_3_MASK to 0x40
    SET constant CONTRACTION_2_MASK to 0x40

    SET constant CONTRACTION_MASK to 0xc0

ELSE
    COMMENT Otherwise, only 2-character or 3-character contractions are supported.
SET constant CONTRACTION_3_MASK to 0xc0  // Bit-mask to check 2 character contraction or 3
//character contraction
SET constant CONTRACTION_2_MASK to 0x80  // Bit-mask to check 2 character contraction
ENDIF

SET constant CASE_UPPER_MASK to 0xe7  // zero out case bits
SET constant CASE_KANA_MASK  to 0xdf  // zero out kana bit
SET constant CASE_WIDTH_MASK to 0xfe  // zero out width bit

//
//  Masks to isolate the various bits in the case weight.
//
//  NOTE: Bit 2 must always equal 1 to avoid getting
//        a byte value of either 0 or 1.
//

SET constant CASE_EXTRA_WEIGHT_MASK to 0xc4
SET constant ISOLATE_KANA to
            (~CASE_KANA_MASK) | CASE_EXTRA_WEIGHT_MASK
SET constant ISOLATE_WIDTH to
            (~CASE_WIDTH_MASK) | CASE_EXTRA_WEIGHT_MASK

//
//  Values for East Asia special case primary weights.
//
SET constant PW_REPEAT      to 0
SET constant PW_CHO_ON      to 1
SET constant MAX_SPECIAL_PW to PW_CHO_ON

//
//  Values for weight 5 - East Asia Extra Weights.
//
SET constant WT_FIVE_KANA to 3
SET constant WT_FIVE_REPEAT to 4
SET constant WT_FIVE_CHO_ON to 5

//
//  PW Mask for Cho-On:
//  Leaves bit 7 on in PW, so it becomes Repeat
//  if it follows Kana N.
//
SET constant CHO_ON_PW_MASK to 0x87

//
//  Special weight values
//
SET constant MAP_INVALID_WEIGHT to 0xff

//
```

```
//  Some Significant Values for Korean Jamo.
//  The L, V & T syllables in the 0x1100 Unicode range
//  can be composed to characters in the 0xac00 range.
//  See The Unicode Standard for details.
//
SET constant NLS_CHAR_FIRST_JAMO       to 0x1100 // Begin Jamo range
SET constant NLS_CHAR_LAST_JAMO        to 0x11f9 // End Jamo range
SET constant NLS_CHAR_FIRST_VOWEL_JAMO to 0x1160 // First Vowel Jamo
SET constant
    NLS_CHAR_FIRST_TRAILING_JAMO to 0x11a8   // First Trailing Jamo
SET constant
    NLS_JAMO_VOWEL_COUNT to 21           // Number of vowel Jamo (V)
SET constant
    NLS_JAMO_TRAILING_COUNT to 28     // Number of trailing Jamo (L)
SET constant
    NLS_HANGUL_FIRST_COMPOSED to 0xac00      // Begin composed range

//
//  Values for Unicode Weight extra weights (e.g. Jamo (old Hangul)).
//  The following uses SM for extra UW weights.
//
SET constant ScriptMember_Extra_UnicodeWeight to 255
//  Leading Weight / Vowel Weight / Trailing Weight
//  according to the current Jamo class.
//
STRUCTURE JamoSortInfoType
(
     // true for an old Hangul sequence
     OldHangulFlag : Boolean

     // true if U+1160 (Hangul Jungseong Filler) used
     FillerUsed : Boolean

     // index to the prior modern Hangul syllable (L)
     LeadingIndex : 8 bit integer

     // index to the prior modern Hangul syllable (V)
     VowelIndex : 8 bit integer

     // index to the prior modern Hangul syllable (T)
     TrailingIndex : 8 bit integer

     // Weight to offset from other old hangul (L)
     LeadingWeight : 8 bit integer

     // Weight to offset from other old hangul (V)
     VowelWeight : 8 bit integer

     // Weight to offset from other old hangul (T)
     TrailingWeight : 8 bit integer
)

// This is the raw data record type from the data table
STRUCTURE JamoStateDataType
(
     // true for an old Hangul sequence
     OldHangulFlag : Boolean

     // index to the prior modern Hangul syllable (L)
```

```
      LeadingIndex : 8 bit integer

      // index to the prior modern Hangul syllable (V)
      VowelIndex : 8 bit integer

      // index to the prior modern Hangul syllable (T)
      TrailingIndex : 8 bit integer

      // weight to distinguish from old Hangul
      ExtraWeight : 8 bit integer

      // number of additional records in this state
      TransitionCount : 8 bit integer

      // Current record in unisort.txt Jamo table:
      JamoRecord : data record

      // SORTTABLES\JAMOSORT\[Character] section
  )
  COMMENT GetWindowsSortKey
  COMMENT
  COMMENT  On Entry:  SourceString - Unicode String to compute a
  COMMENT                            sort key for
  COMMENT             SortLocale   - Locale to determine correct
  COMMENT                            linguistic sort
  COMMENT             Flags        - Bit Flag to control behavior
  COMMENT                            of sort key generation.
  COMMENT
  COMMENT  NORM_IGNORENONSPACE    Ignore diacritic weight
  COMMENT  NORM_IGNORECASE:       Ignore case weight
  COMMENT  NORM_IGNOREKANATYPE:   Ignore Japanese Katakana/Hiraga
  COMMENT                         difference
  COMMENT  NORM_IGNOREWIDTH:      Ignore Chinese/Japanese/Korean
  COMMENT                         half-width and full-width difference.
  COMMENT
  COMMENT  On Exit:   SortKey      - Byte array containing the
  COMMENT                            computed sort key.
  COMMENT

  PROCEDURE GetWindowsSortKey(IN SourceString : Unicode String,
                              IN SortLocale :   LCID,
                              IN Flags : 32 bit integer,
                              OUT SortKey : BYTE String)

  COMMENT Compute flags for sort conditions
  COMMENT Based on the case/kana/width flags,
  COMMENT   turn off bits in case mask when comparing case weight.

  SET CaseMask to 0xff

  If (NORM_IGNORECASE bit is on in Flags) THEN
      SET CaseMask to CaseMask LOGICAL AND with CASE_UPPER_MASK
  ENDIF

  If (NORM_IGNOREKANATYPE bit is on in Flags) THEN
      SET CaseMask to CaseMask LOGICAL AND with CASE_KANA_MASK
  ENDIF

  If (NORM_IGNOREWIDTH bit is on in Flags) THEN
```

```
    SET CaseMask to CaseMask LOGICAL AND with CASE_WIDTH_MASK
ENDIF

COMMENT Windows 7 and Windows Server 2008 R2 use 3-byte (instead of 2-byte) sequence for
COMMENT Unicode Weights
COMMENT for Private Use Area (PUA) and some Chinese/Japanese/Korean (CJK) script members.

COMMENT Does this sort have a 3-byte Unicode Weight (CJK sorts)?
IF Windows version is Windows 7 and Windows Server 2008 R2 THEN
   COMMENT Check if the locale can have 3-byte Unicode weight
   SET Is3ByteWeightLocale to CALL Check3ByteWeightLocale(SortLocale)
ENDIF


IF Windows version is Windows Vista, Windows Server 2008, Windows 7, or Windows Server 2008
R2 THEN
    COMMENT For Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2,
    COMMENT the algorithm
    COMMENT does not remap the script for Korean locale
    SET IsKoreanLocale to false
ELSE


   IF SortLocale is LCID_KOREAN or
      SortLocale is LCID_KOREAN_UNICODE_SORT THEN
         SET IsKoreanLocale to true
         IF KoreanScriptMap is null THEN
             CALL InitKoreanScriptMap
   ELSE
       SET IsKoreanLocale to false
   ENDIF
ENDIF

//
//  Allocate buffer to hold different levels of sort key weights.
//  UnicodeWeights/ExtraWeights/SpecialWeights will be eventually
//  to be collected together, in that order, into the returned
//  Sortkey byte string.
//
//  Maximum expansion size is 3 times the input size
//

// Unicode Weight => 4 word (16 bit) length
// (extension A and Jamo need extra words)
SET UnicodeWeights to new empty string of UnicodeWeightType

SET DiacriticWeights to new empty string of BYTE
SET CaseWeights to new empty string of BYTE

// Extra Weight=>4 byte length (4 weights, 1 byte each) FE Special
SET ExtraWeights to new empty string of ExtraWeightType

// Special Weight => dword length (2 words each of 16 bits)
SET SpecialWeights to new empty string of SpecialWeightType

//
// Go through the string, code point by code point,
// testing for contractions and Hungarian special character sequence
//
```

```
// loop presumes 0 based index for source string
FOR SourceIndex is 0 to Length(SourceString) -1
    //
    // Get weights
    // CharacterWeight will contain all of the weight information
    // for the character tested.
    //

    SET CharacterWeight to CALL GetCharacterWeights
        WITH (SortLocale, SourceString[SourceIndex])

    SET ScriptMember to CharacterWeight.ScriptMember

    // Special case weights have script members less than
    // MAX_SPECIAL_CASE (11)
    IF ScriptMember is greater than MAX_SPECIAL_CASE  THEN

        //
        //  No special case on character, but must check for
        //  contraction characters and Hungarian special character sequence
        //  characters.
        //

        SET HasHungarianSpecialCharacterSequence to CALL
            TestHungarianCharacterSequences
                 WITH (SortLocale, SourceString, SourceIndex)

        SET Result to CALL GetContractionType WITH (CharacterWeight)

        CASE Result OF

           "3-character Contraction":
               COMMENT This is only possible for Windows versions that are Windows NT 4.0
               COMMENT through Windows Server 2003
               Set ContractionFound to CALL SortkeyContractionHandler
                 WITH (SortLocale, SourceString, SourceIndex,
                        HasHungarianSpecialCharacterSequence, 3,
                        UnicodeWeights, DiacriticWieghts, CaseWeights)
               IF ContractionFound is true THEN
                   COMMENT Break out of the case statement
                   BREAK
               ENDIF
               IF ContractionFound is true THEN
                   COMMENT Break out of the case statement
                   BREAK
               ENDIF
               COMMENT If no contraction is found, fall through into the additional cases.
               FALLTHROUGH

           "2-character Contraction":
               COMMENT This is only possible for Windows versions that are Windows NT 4.0
               COMMENT through Windows Server 2003
               Set ContractionFound to CALL SortkeyContractionHandler
                WITH (SortLocale, SourceString, SourceIndex,
                       HasHungarianSpecialCharacterSequence, 2,
                       UnicodeWeights, DiacriticWieghts, CaseWeights)
               IF ContractionFound is true THEN
                   COMMENT Break out of the case statement
```

```
            BREAK
        ENDIF
        COMMENT If no contraction is found, fall through into the OTHER case.
        COMMENT Since "3-character contraction" or "2-character contraction" are the
        COMMENT only two possible values for
        COMMENT Windows NT 4.0 through Windows Server 2003, all calls to
        COMMENT SortkeyContractionHandler  will return false.
        COMMENT So, the fallthrough will go directly to the OTHERS section
        FALLTHROUGH

    "6-character contraction, 7-character contraction, or 8-character contraction":
        Set ContractionFound to CALL SortkeyContractionHandler
         WITH (SortLocale, SourceString, SourceIndex,
               HasHungarianSpecialCharacterSequence, 8,
               UnicodeWeights, DiacriticWieghts, CaseWeights)
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ELSE
            Set ContractionFound to CALL SortkeyContractionHandler
             WITH (SortLocale, SourceString, SourceIndex,
                   HasHungarianSpecialCharacterSequence, 7,
                   UnicodeWeights, DiacriticWieghts, CaseWeights)
        ENDIF
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ELSE
            Set ContractionFound to CALL SortkeyContractionHandler
             WITH (SortLocale, SourceString, SourceIndex,
                   HasHungarianSpecialCharacterSequence, 6,
                   UnicodeWeights, DiacriticWieghts, CaseWeights)
        ENDIF
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ENDIF
        COMMENT If no contraction is found, fall through into additional cases.
        FALLTHROUGH

    "4-character contraction or 5-character contraction":
        Set ContractionFound to CALL SortkeyContractionHandler
         WITH (SortLocale, SourceString, SourceIndex,
               HasHungarianSpecialCharacterSequence, 5,
               UnicodeWeights, DiacriticWieghts, CaseWeights)
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ELSE
            Set ContractionFound to CALL SortkeyContractionHandler
             WITH (SortLocale, SourceString, SourceIndex,
                   HasHungarianSpecialCharacterSequence, 4,
                   UnicodeWeights, DiacriticWieghts, CaseWeights)
        ENDIF
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ENDIF
        COMMENT If no contraction is found, fall through into additional cases.
```

```
        FALLTHROUGH

    "2-character contraction or 3-character contraction":
        Set ContractionFound to CALL SortkeyContractionHandler
         WITH (SortLocale, SourceString, SourceIndex,
               HasHungarianSpecialCharacterSequence, 3,
               UnicodeWeights, DiacriticWieghts, CaseWeights)
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ELSE
            Set ContractionFound to CALL SortkeyContractionHandler
             WITH (SortLocale, SourceString, SourceIndex,
                   HasHungarianSpecialCharacterSequence, 2,
                   UnicodeWeights, DiacriticWieghts, CaseWeights)
        ENDIF
        IF ContractionFound is true THEN
            COMMENT Break out of the case statement
            BREAK
        ENDIF
        COMMENT If no contraction is found, fall through into additional cases.
        FALLTHROUGH


    OTHERS :
       IF Windows version is greater than Windows Server 2008 R2 or Windows 7 THEN
           COMMENT In Windows Server 2008 R2 or Windows 7, Private Use Area (PUA) code
           COMMENT points
           COMMENT and some CJK (Chinese/Japanese/Korean) sorts may need 3 byte
           COMMENT weights
           COMMENT Store normal Unicode weight first. Note that there is no
           COMMENT adjustment of Korean weight anymore.
           SET UnicodeWeight to
              CorrectUnicodeWeight(CharacterWeight, FALSE)
           COMMENT Assume 3-byte Unicode Weight is not used first. The alogorithm will
           COMMENT check this later.
            SET UnicodeWeight.ThirdByteWeight to 0

           IF (ScriptMember is equal to or greater than PUA3BYTESTART)
              AND
              (ScriptMember is less than or equal to PUA3BYTEEND) THEN
               SET IsScriptMemberPUA3BYTEWeight to true
           ELSE
               SET IsScriptMemberPUA3ByteWeight to false
           ENDIF


           IF (ScriptMember is equal to or greater than CJK3BYTESTART) AND
              (ScriptMember is less than or equal to CJK3BYTEEND) THEN
               SET IsScriptMemberCJK3ByteWeight to true
           ELSE
            SET IsScriptMemberCJK3ByteWeight to false
           ENDIF
           IF (IsScriptMemberPUA3ByteWeight is true) OR
              (Is3ByteWeightLocale AND
               IsScriptMemberCJK3ByteWeight is true) THEN
               COMMENT PUA code points and some CJK sorts need 3 byte weights
               SET UnicodeWeight.ThirdByteWeight to CharacterWeight.DiacriticWeight
           ELSE
```

```
                    COMMENT Normal Diacritic Weight
                    APPEND CharacterWeight.DiacriticWeight to DiacriticWeights as a BYTE
                ENDIF
                APPEND UnicodeWeight to UnicodeWeights

                SET CaseWeight to GetCaseWeight(CharacterWeight)
                APPEND CharacterWeight.CaseWeight to CaseWeights as a BYTE

            ELSE

                SET UnicodeWeight to
                   CorrectUnicodeWeight(CharacterWeight, IsKoreanLocale)
                APPEND UnicodeWeight to UnicodeWeights
                APPEND CharacterWeight.DiacriticWeight to DiacriticWeights
                       as a BYTE
                SET CaseWeight to GetCaseWeight(CharacterWeight)
                APPEND CharacterWeight.CaseWeight to CaseWeights as a BYTE
            ENDIF
       ENDCASE
    ELSE
       CALL SpecialCaseHandler WITH (SourceString, SourceIndex,
                UnicodeWeights, ExtraWeights, SpecialWeights,
                SortLocale, IsKoreanLocale)
    ENDIF
ENDFOR

//
//  Store the Unicode Weights in the destination buffer.
//
FOR each UnicodeWeight in UnicodeWeights
    //
    //  Copy Unicode weight to destination buffer.
    //
    APPEND UnicodeWeight.ScriptMember to SortKey as a BYTE
    APPEND UnicodeWeight.PrimaryWeight to SortKey as a BYTE
   IF Windows version is greater than Windows Server 2008 R2 or Windows 7 THEN
       IF UnicodeWeight.ThirdByteWeight is not 0 THEN
           COMMENT When 3-byte Unicode Weight is used, append the additional BYTE into
           COMMENT SortKey
           APPEND UnicodeWeight.ThirdByteWeight to SortKey as a BYTE
       ENDIF
    ENDIF

ENDFOR

//
//  Copy Separator to destination buffer.
//
APPEND SORTKEY_SEPARATOR to SortKey as a BYTE

//
//  Store Diacritic Weights in the destination buffer.
//
IF (NORM_IGNORENONSPACE bit is not turned on in Flags) THEN
    IF (IsReverseDW is TRUE) THEN
       //
       //  Reverse diacritics:
       //    - remove diacritics from left  to right.
```

```
        //    - store  diacritics from right to left.
        //
        FOR each DiacriticWeight in
            DiacriticWeights in the "first in first out" order
           IF DiacriticWeight <= MIN_DW THEN
              REMOVE DiacriticWeight from DiacriticWeights
           ELSE
              BREAK from the current FOR loop
           ENDIF
        ENDFOR

        FOR each DiacriticWeight in
            DiacriticWeights in the "last in first out" order
           //
           //  Copy Unicode weight to destination buffer.
           //
           APPEND DiacriticWeight to SortKey as a BYTE
        ENDFOR
     ELSE
        //
        //  Regular diacritics:
        //    - remove diacritics from right to left.
        //    - store  diacritics from left  to right.
        FOR each DiacriticWeight in
            DiacriticWeights in the "last in first out" order
            IF DiacriticWeight <= MIN_DW THEN
               REMOVE DiacriticWeight from DiacriticWeights
            ELSE
               BREAK from the current FOR loop
            ENDIF
        ENDFOR

        FOR each DiacriticWeight in
            DiacriticWeights in the order of "first in first out"
            //
            //  Copy Unicode weight to destination buffer.
            //
            APPEND DiacriticWeight to SortKey as a BYTE
        ENDFOR
     ENDIF
  ENDIF

  //
  //  Copy Separator to destination buffer.
  //
  APPEND SORTKEY_SEPARATOR to SortKey as a BYTE

  //
  //  Store case Weights
  //
  //    - Eliminate minimum CW.
  //    - Copy case weights to destination buffer.
  //
  IF (NORM_IGNORECASE bit is not turned on in Flags
       OR NORM_IGNOREWIDTH bit is not turned on in Flags) THEN
      FOR each CaseWeight in CaseWeights
          in the "last in first out" order
          IF CaseWeight <= MIN_CW THEN
             REMOVE CaseWeight from CaseWeights
```

```
         ELSE
            BREAK from the current FOR loop
         ENDIF
      ENDFOR

      FOR each CaseWeight in CaseWeights
         //
         //  Copy Unicode weight to destination buffer.
         //
         APPEND CaseWeight to SortKey as a BYTE
      ENDFOR
   ENDIF

   //
   //  Copy Separator to destination buffer.
   //
   APPEND SORTKEY_SEPARATOR to SortKey as a BYTE

   //
   //  Store the Extra Weights in the destination buffer for
   //  EAST ASIA Special.
   //
   //    - Eliminate unnecessary XW.
   //    - Copy extra weights to destination buffer.
   //
   IF Length(ExtraWeights) is greater than 0 THEN
      IF (NORM_IGNORENONSPACE bit is turned on in Flag) THEN
         APPEND 0xff to SortKey as a BYTE
         APPEND 0x02 to SortKey as a BYTE
      ENDIF

      // Append W6 group to SortKey
      // Trim unused values from the end of the string
      SET EndExtraWeight to Length(ExtraWeights) - 1

      WHILE EndExtraWeight greater than 0 and
           ExtraWeightSeparator[EndExtraWeight].W6 == 0xe4
         DECREMENT EndExtraWeight
      ENDWHILE

      SET ExtraWeightIndex to 0
      WHILE ExtraWeightIndex is less than or equal to EndExtraWeight
         APPEND ExtraWeightSeparator[ExtraWeightIndex].W6
           to SortKey as a BYTE
         INCREMENT ExtraWeightIndex
      ENDWHILE

      // Append W6 separator
      APPEND 0xff to SortKey as a BYTE

      // Append W7 group to SortKey
      // Trim unused values from the end of the string
      SET EndExtraWeight to Length(ExtraWeights) - 1
      WHILE EndExtraWeight greater than 0 and
            ExtraWeightSeparator[EndExtraWeight].W7 == 0xe4
         DECREMENT EndExtraWeight
      ENDWHILE

      SET ExtraWeightIndex to 0
```

```
   WHILE ExtraWeightIndex is less than or equal to EndExtraWeight
      APPEND ExtraWeightSeparator[ExtraWeightIndex].W7 to SortKey
      INCREMENT ExtraWeightIndex
   ENDWHILE

   // Append W7 separator
   APPEND 0xff to SortKey as a BYTE
ENDIF

//
//  Copy Separator to destination buffer.
//
APPEND SORTKEY_SEPARATOR to SortKey as a BYTE

//
//  Store the Special Weights in the destination buffer.
//
//    - Copy special weights to destination buffer.
//
FOR each SpecialWeight in SpecialWeights
   // High byte (most significant)
   SET Byte1 to SpecialWeight.Position >> 8
   // Low byte (least significant)
   SET Byte2 to SpecialWeight.Position & 0xff
   APPEND Byte1 to SortKey as a BYTE
   APPEND Byte2 to SortKey as a BYTE
   APPEND SpecialWeight.Script to SortKey as a BYTE
   APPEND SpecialWeight.Weight to SortKey as a BYTE
ENDFOR

//
//  Copy terminator to destination buffer.
//
APPEND SORTKEY_TERMINATOR to SortKey

RETURN SortKey
```

3.1.5.2.5 TestHungarianCharacterSequences

This algorithm checks if the specified UTF-16 string has a Hungarian special-character sequence for the specified locale in the specific string index.

Hungarian contains special character sequences in which the first character of the string designates a string that is equivalent to the last three characters of the string, for example, the string “ddzs” is actually treated as the string “dzsdzs” for the purposes of generating the sort key. This function checks to see if the specified locale is Hungarian, and it also checks to see if the next two characters starting in the specified index are the same. This indicates that it is a likely Hungarian special-character sequence.

```
COMMENT TestHungarianCharacterSequences
COMMENT
COMMENT  On Entry:  SortLocale    - Locale to use for linguistic data
COMMENT             SourceString  - Unicode String to look for Hungarian
COMMENT                             special character sequence in
COMMENT             SourceIndex   - Index of character in string to
COMMENT                             look for start of
COMMENT                             Hungarian special character sequence
```

```
COMMENT
COMMENT  On Exit:   Result        - Set to true if a Hungarian special
COMMENT                             character sequence
COMMENT                             was found
COMMENT

PROCEDURE TestHungarianCharacterSequences(IN SortLocale : LCID,
                                IN SourceString : Unicode String,
                                IN SourceIndex : 32 bit integer,
                                OUT Result : Boolean)

// Hungarian special character sequence only happen to Hungarian
// Note that this can be found in unisort.txt in the
// SORTTABLES\DOUBLECOMPRESSION section, however since
// there's only 1 locale we just hard code it here.
IF SortLocale not equal to LCID_HUNGARIAN) THEN
    SET Result to false
    RETURN
ENDIF

// first test to make sure more data is available
IF SourceIndex + 1 is greater than or equal to
                     Length(SourceString) THEN
    SET Result to false
    RETURN
ENDIF

// CMP_MASKOFF_CW (e7) is not necessary
// since it was already masked off
SET FirstWeight to CALL GetCharacterWeights WITH
        (SortLocale, SourceString[SourceIndex])
SET SecondWeight to CALL GetCharacterWeights WITH
        (SortLocale, SourceString[SourceIndex + 1])

IF FirstWeight is equal to SecondWeight THEN
    SET Result to true
ELSE
    SET Result to false
ENDIF

RETURN
```

3.1.5.2.6 GetContractionType

This algorithm specifies the checking of the type of contraction based on the character weight. Contraction is defined by [UNICODE-COLLATION] section 3.2.

For instance, "ll" acts as a single unit in Spanish so that it comes between l and m. This is a two-character contraction. Similarly, "dzs" acts as a single unit in Hungarian, so it is a three-character contraction.

These functions will specify if the weights will not be at the beginning of a contraction, the beginning of a two-character contraction, or the beginning of a three-character contraction.

```
COMMENT GetContractionType
COMMENT
COMMENT  On Entry:  CharacterWeight - Weights structure to test for
```

```
COMMENT                                a contraction
COMMENT
COMMENT  On Exit:   Result          - Type of contraction found:
COMMENT                                "No contraction"
COMMENT                                "3-character contraction"
COMMENT                                "2-character contraction"
COMMENT                                The following results are only possible for
COMMENT                                Windows Vista, Windows Server 2008, Windows 7, and
COMMENT                                Windows Server 2008 R2
COMMENT                                "6-character contraction, 7-character contraction or
COMMENT                                 8-character contraction"
COMMENT                                "4-character contraction or 5-character contraction"
COMMENT                                "2-character contraction or 3-character contraction"


PROCEDURE GetContractionType(IN CharacterWeight : CharacterWeightType,
                        OUT Result)
   IF Windows version is Windows NT 4.0 to Windows 2003 THEN
       CASE CharacterWeight.CaseWeight & CONTRACTION_3_MASK OF
            CONTRACTION_3_MASK : SET Result = "3-character contraction"
            CONTRACTION_2_MASK : SET Result = "2-character contraction"
            OTHERS : SET Result = "No contraction"
       ENDCASE
       ELSE
       COMMENT Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2
       CASE CharacterWeight.CaseWeight & CONTRACTION_MASK OF
           CONTRACTION_6_MASK : SET Result = "6-character contraction, 7-
           character contraction or 8-character contraction"
           CONTRACTION_4_MASK : SET Result = "4-character contraction or 5-
           character contraction"
           CONTRACTION_2_MASK : SET Result = "2-character contraction or 3-
           character contraction"
           OTHERS : SET Result = "No contraction"
       ENDCASE
   ENDIF

RETURN
```

3.1.5.2.7 CorrectUnicodeWeight

This algorithm specifies the processing of the corrected Unicode weight for the specific character weight, and whether the locale is a Korean locale.

```
COMMENT CorrectUnicodeWeight
COMMENT
COMMENT  On Entry:  CharacterWeight - Weights structure to get Unicode
COMMENT                               weight of
COMMENT             IsKoreanLocale  - True if this locale needs
COMMENT                               adjustment for
COMMENT                               Korean mapped scripts behavior.
COMMENT
COMMENT  On Exit:   UnicodeWeight   - Corrected Unicode Weight
COMMENT

PROCEDURE
        CorrectUnicodeWeight(IN CharacterWeight : CharacterWeightType,
                             IN IsKoreanLocale : boolean,
```

```
                              OUT UnicodeWeight : UnicodeWeightType)

  SET UnicodeWeight to CALL MakeUnicodeWeight WITH
          (CharacterWeight.ScriptMember, CharacterWeight.PrimaryWeight,
           IsKoreanLocale)

  RETURN UnicodeWeight
```

3.1.5.2.8 MakeUnicodeWeight

This algorithm specifies the generation of the Unicode weight based on the script member, the primary weight, and whether the locale is a Korean locale.

```
  COMMENT MakeUnicodeWeight
  COMMENT
  COMMENT  On Entry:  ScriptMember    - Script member to use for
  COMMENT                               Unicode weight
  COMMENT             PrimaryWeight   - Primary weight to use for
  COMMENT                               Unicode weight
  COMMENT             IsKoreanLocale  - True if this locale needs
  COMMENT                               adjustment for Korean mapped
  COMMENT                               scripts behavior.
  COMMENT
  COMMENT  On Exit:   UnicodeWeight   - Corrected Unicode Weight
  COMMENT

  PROCEDURE MakeUnicodeWeight(IN ScriptMember : 8 bit byte,
                              IN PrimaryWeight : 8 bit byte,
                              IN IsKoreanLocale : boolean,
                              OUT UnicodeWeight : UnicodeWeightType)

  IF IsKoreanLocale is true THEN
      SET UnicodeWeight.ScriptMember to
      KoreanScriptMap[ScriptMember]
  ELSE
      SET UnicodeWeight.ScriptMember to ScriptMember
  ENDIF

  SET UnicodeWeight.PrimaryWeight to PrimaryWeight
  RETURN UnicodeWeight
```

3.1.5.2.9 GetCharacterWeights

This algorithm specifies the retrieval of the character weight based on the specified locale and the specified UTF-16 code point.

```
  COMMENT GetCharacterWeights
  COMMENT
  COMMENT  On Entry:  SortLocale      - Locale to use for linguistic
  COMMENT                               data
  COMMENT             SourceCharacter - Unicode Character to return
  COMMENT                               weight for
  COMMENT
  COMMENT  On Exit:   Result          - A structure containing the
  COMMENT                               weights for this character
```

```
COMMENT

PROCEDURE GetCharacterWeights(IN SortLocale : LCID,
                              IN SourceCharacter : Unicode Character,
                              OUT Result : CharacterWeightType)

// Search for the character in the exception table
OPEN SECTION ExceptionTable where name is
             SORTTABLES\EXCEPTION\LCID[SortLocale] from unisort.txt
SELECT RECORD CharacterRow FROM ExceptionTable WHERE field 1
             matches SourceCharacter

IF CharacterRow is null THEN
    // Not found, search for the character in the default table
    OPEN SECTION DefaultTable where name is
             SORTKEY\DEFAULT from unisort.txt
    SELECT RECORDCharacterRow from DefaultTable where field 1
             matches SourceCharacter

    IF CharacterRow is null THEN
        // Not found in default table either, check expansions
        SET Expansion to GetExpandedCharacters(SourceCharacter)

        IF Expansion is not null THEN
            // Has an expansion, set appropriate weights
            SET Result.ScriptMember to EXPANSION
        ELSE
            // No expansion, set appropriate weights
            SET Result.ScriptMember to UNSORTABLE
        ENDIF

        SET Result.PrimaryWeight to 0
        SET Result.DiacriticWeight to 0
        SET Result.CaseWeight to 0

        RETURN Result
    ENDIF
ENDIF

SET Result.ScriptMember to CharacterRow.Field2
SET Result.PrimaryWeight to CharacterRow.Field3
SET Result.DiacriticWeight to CharacterRow.Field4
SET Result.CaseWeight to CharacterRow.Field5

RETURN Result
```

3.1.5.2.10 GetExpansionWeights

This algorithm specifies the generation of a character weight for the specified character that has the Expansion behavior, as defined in [UNICODE-COLLATION] section 3.2.

```
COMMENT GetExpansionWeights
COMMENT
COMMENT  On Entry:  SourceCharacter - Character to look up
COMMENT                               expansions for
COMMENT             SortLocale      - Locale to get sort weights for
COMMENT
```

```
COMMENT  On Exit:   Weights         - String of 2 or 3 weights for
COMMENT                               this character
COMMENT

PROCEDURE GetExpansionWeights(IN SourceCharacter : Unicode Character,
                          IN SortLocale : LCID,
                          OUT Weights : CharacterWeightType String)

SET Weights to new empty string of CharacterWeightType
SET ExpandedCharacters to CALL GetExpandedCharacters WITH
                  (SourceCharacter)

// Append first weight
SET Weight to CALL GetCharacterWeights WITH
                  (SortLocale, ExpandedCharacters[0])
APPEND Weight to Weights

// Get second weight, it may expand again
SET Weight to CALL GetCharacterWeights WITH
                  (SortLocale, ExpandedCharacters[1])

IF Weight.ScriptMember is EXPANSION THEN
    // second weight expands again, get new expansion
    // note that this can only happen once, as it does
    // with the U=fb03 (ffi ligature)

    SET ExpandedCharacters to CALL
                 GetExpandedCharacters(ExpandedCharacters[1])

    // Append second expansion's first weight
    SET Weight to CALL GetCharacterWeights WITH
                  (SortLocale, ExpandedCharacters[0])
    APPEND Weight to Weights

    // Get second weight for second expansion, it will not expand again
    SET Weight to CALL GetCharacterWeights WITH
                  (SortLocale, ExpandedCharacters[1])
ENDIF

// Finish appending second weight to weights string
APPEND Weight to Weights

RETURN Result
```

3.1.5.2.11 GetExpandedCharacters

This algorithm specifies the generation of the array of expanded characters, if the specified character can be expanded.

```
COMMENT GetExpandedCharacters
COMMENT
COMMENT  On Entry:  SourceCharacter - Character to look for in
COMMENT                               expansion table
COMMENT
COMMENT  On Exit:   Result          - Array of two unicode characters
COMMENT                               for the expansion or null if no
COMMENT                               expansion found
```

```
COMMENT
COMMENT NOTE: Look for default table characters first, some entries
COMMENT       in the expansion table are only used in exception tables
COMMENT       for some locales (ie: 0x00c4 Ä)

PROCEDURE
    GetExpandedCharacters(IN SourceCharacter : Unicode Character,
                          OUT Result : Unicode Character[2])

// Search for the expansion in the expansion table
OPEN SECTION ExpansionTable where name is
   SORTTABLES\EXPANSION from unisort.txt

SELECT RECORD ExpansionRow FROM ExceptionTable WHERE field 1
   matches SourceCharacter

IF ExpansionRow is null THEN
    SET Result to null
    RETURN Result
ENDIF

SET Result[0] to ExpansionRow.Field2
SET Result[1] to ExpansionRow.Field3

RETURN Result
```

3.1.5.2.12 SortkeyContractionHandler

This algorithm checks where the next few characters in the specified string and index has 8-character, 7-character, 6-character, 5-character, 4-character, 3-character, or 2-character contraction sequence. If yes, these characters are given just one character weight. This algorithm also handles the Hangiran special character sequence.

```
COMMENT SortkeyContractionHandler
COMMENT
COMMENT  On Entry: SourceString – Source Unicode String
COMMENT            SourceIndex – Current index within source string
COMMENT            HasHungarianSpecialCharacterSequence: Is the character that the current
COMMENT               index points to
COMMENT               the starting of the Hungarian special character sequence
COMMENT            ContractionType: The contraction type, from 2-character to 8-character
COMMENT                             contraction, to be checked against
COMMENT            UnicodeWeights  - String of UnicodeWeightType to
COMMENT                              append additional weight(s) to
COMMENT            DiacriticWeights - String of Diacritic Weight to
COMMENT                              append extra weight(s) to if
COMMENT                              needed
COMMENT            CaseWeights  - String of Case Weight to
COMMENT                              append special weight(s) to
COMMENT                              if needed
COMMENT
COMMENT  On Exit: Result: a string to indicate the type of contraction from the specified
COMMENT                   string
COMMENT             UnicodeWeights  - The UnicodeWeight of the
COMMENT                               processed character(s) is
COMMENT                               appended to this string.
COMMENT             DiacriticWeights    - The Diacritic weight, if any, of
```

```
COMMENT                                 the processed character(s) is
COMMENT                                 appended to this string.
COMMENT             CaseWeights  - The Case Weight, if any,
COMMENT                                 of the processed character(s)
COMMENT                                 is appended to this string.
COMMENT
PROCEDUE SortkeyContractionHandler (IN SortLocale: LCID,
     IN SourceString: Unicode String,
     IN SourceIndex: 32-bit integer,
     IN HasHungarianSpecialCharacterSequence: boolean
     IN ContractionType: integer number from 2 to 8
     INOUT UnicodeWeights: string of UnicodeWeightType
     INOUT DiacriticWeights: string of BYTE
     INOUT CaseWeights: string of BYTE)

Result: CharacterWeightType

IF HasHungarianSpecialCharacterSequence is true THEN
    COMMENT We are in the beginning of Hungarian special character sequence,
    COMMENT advance one character before we start to check for contraciton sequence
    SET SourceIndex to SourceIndex + 1
ENDIF

IF SourceIndex + ContractionType is greater than or equal to SourceString.Length THEN
    SET Result to null
    RETURN false
ENDIF

COMMENT Search for the character in the character contraction table
COMMENT Search for contraction section based on ContractionType

CASE ContractionType
    "8":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\EIGHT from unisort.txt
    "7":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\SEVEN from unisort.txt
    "6":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\SIX from unisort.txt
    "5":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\FIVE from unisort.txt
    "4":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\FOUR from unisort.txt
    "3":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\THREE from unisort.txt
    "2":
OPEN SECTION ContractionTable where name is
          SORTTABLES\COMPRESSION\LCID[SortLocale]\TWO from unisort.txt

ENDCASE


COMMENT Contraction table may not be found if locale doesn't have them
IF ContractionTable is null THEN
```

```
    SET Result to null
    RETURN false
ENDIF

CASE ContractionType
      "8":
          SELECT RECORD ContractionRow FROM ContractionTable
               WHERE field 1 matches SourceString[SourceIndex] and
               WHERE field 2 matches SourceString[SourceIndex + 1] and
               WHERE field 3 matches SourceString[SourceIndex + 2] and
               WHERE field 4 matches SourceString[SourceIndex + 3] and
               WHERE field 5 matches SourceString[SourceIndex + 4] and
               WHERE field 6 matches SourceString[SourceIndex + 5] and
               WHERE field 7 matches SourceString[SourceIndex + 6] and
               WHERE field 8 matches SourceString[SourceIndex + 7]

          COMMENT If this sequence isn't a contraction we won't find one
          IF ContractionRow is null THEN
              SET Result to null
          RETURN false
          ENDIF

          COMMENT Found a contraction, get its weights
          SET Result.ScriptMember to ContractionRow.Field9
          SET Result.PrimaryWeight to ContractionRow.Field10

          SET Result.DiacriticWeight to ContractionRow.Field11
          SET Result.CaseWeight to ContractionRow.Field12
    "7":
          SELECT RECORD ContractionRow FROM ContractionTable
              WHERE field 1 matches SourceString[SourceIndex] and
              WHERE field 2 matches SourceString[SourceIndex + 1] and
              WHERE field 3 matches SourceString[SourceIndex + 2] and
              WHERE field 4 matches SourceString[SourceIndex + 3] and
              WHERE field 5 matches SourceString[SourceIndex + 4] and
              WHERE field 6 matches SourceString[SourceIndex + 5] and
              WHERE field 7 matches SourceString[SourceIndex + 6]

           COMMENT If this sequence isn't a contraction we won't find one
           IF ContractionRow is null THEN
              SET Result to null
              RETURN false
           ENDIF

           COMMENT Found a contraction, get its weights
           SET Result.ScriptMember to ContractionRow.Field8
           SET Result.PrimaryWeight to ContractionRow.Field9

           SET Result.DiacriticWeight to ContractionRow.Field10
           SET Result.CaseWeight to ContractionRow.Field11

    "6":
       SELECT RECORD ContractionRow FROM ContractionTable
          WHERE field 1 matches SourceString[SourceIndex] and
          WHERE field 2 matches SourceString[SourceIndex + 1] and
          WHERE field 3 matches SourceString[SourceIndex + 2] and
          WHERE field 4 matches SourceString[SourceIndex + 3] and
          WHERE field 5 matches SourceString[SourceIndex + 4] and
          WHERE field 6 matches SourceString[SourceIndex + 5]
```

```
      COMMENT If this sequence isn't a contraction we won't find one
      IF ContractionRow is null THEN
         SET Result to null
         RETURN false
      ENDIF

      COMMENT Found a contraction, get its weights
      SET Result.ScriptMember to ContractionRow.Field7
      SET Result.PrimaryWeight to ContractionRow.Field8

      SET Result.DiacriticWeight to ContractionRow.Field9
      SET Result.CaseWeight to ContractionRow.Field10

   "5":
      SELECT RECORD ContractionRow FROM ContractionTable
         WHERE field 1 matches SourceString[SourceIndex] and
         WHERE field 2 matches SourceString[SourceIndex + 1] and
         WHERE field 3 matches SourceString[SourceIndex + 2] and
         WHERE field 4 matches SourceString[SourceIndex + 3] and
         WHERE field 5 matches SourceString[SourceIndex + 4]

      COMMENT If this sequence isn't a contraction we won't find one
      IF ContractionRow is null THEN
         SET Result to null
         RETURN false
      ENDIF

      COMMENT Found a contraction, get its weights
      SET Result.ScriptMember to ContractionRow.Field6
      SET Result.PrimaryWeight to ContractionRow.Field7

      SET Result.DiacriticWeight to ContractionRow.Field8
      SET Result.CaseWeight to ContractionRow.Field9

   "4":
      SELECT RECORD ContractionRow FROM ContractionTable
         WHERE field 1 matches SourceString[SourceIndex] and
         WHERE field 2 matches SourceString[SourceIndex + 1] and
         WHERE field 3 matches SourceString[SourceIndex + 2] and
         WHERE field 4 matches SourceString[SourceIndex + 3]

      COMMENT If this sequence isn't a contraction we won't find one
      IF ContractionRow is null THEN
         SET Result to null
         RETURN false
      ENDIF

      COMMENT Found a contraction, get its weights
      SET Result.ScriptMember to ContractionRow.Field5
      SET Result.PrimaryWeight to ContractionRow.Field6

      SET Result.DiacriticWeight to ContractionRow.Field7
      SET Result.CaseWeight to ContractionRow.Field8

   "3":
     SELECT RECORD ContractionRow FROM ContractionTable
         WHERE field 1 matches SourceString[SourceIndex] and
         WHERE field 2 matches SourceString[SourceIndex + 1] and
```

```
            WHERE field 3 matches SourceString[SourceIndex + 2]

        COMMENT If this sequence isn't a contraction we won't find one
        IF ContractionRow is null THEN
            SET Result to null
            RETURN false
        ENDIF

        COMMENT Found a contraction, get its weights
        SET Result.ScriptMember to ContractionRow.Field4
        SET Result.PrimaryWeight to ContractionRow.Field5

        SET Result.DiacriticWeight to ContractionRow.Field6
        SET Result.CaseWeight to ContractionRow.Field7

      "2":
         SELECT RECORD ContractionRow FROM ContractionTable
            WHERE field 1 matches SourceString[SourceIndex] and
            WHERE field 2 matches SourceString[SourceIndex + 1]

         COMMENT If this sequence isn't a contraction we won't find one
         IF ContractionRow is null THEN
             SET Result to null
             RETURN false
         ENDIF

         COMMENT Found a contraction, get its weights
         SET Result.ScriptMember to ContractionRow.Field3
         SET Result.PrimaryWeight to ContractionRow.Field4

         SET Result.DiacriticWeight to ContractionRow.Field5
         SET Result.CaseWeight to ContractionRow.Field6


  ENDCASE

  SET UnicodeWeight to
     CorrectUnicodeWeight(Result, IsKoreanLocale)
  APPEND UnicodeWeight to UnicodeWeights
  APPEND Result.DiacriticWeight to DiacriticWeights as a BYTE
  APPEND Result.CaseWeight to CaseWeights as a BYTE

  COMMENT Advance the source index
  SET SourceIndex to SourceIndex + ContractionType

  RETURN true
```

3.1.5.2.13 Check3ByteWeightLocale

This algorithm checks if the specified locale is a CJK (Chinese/Japanese/Korean) sorting locale that uses third byte in Unicode weight.

```
  COMMENT Check3ByteWeightLocale
  COMMENT
  COMMENT  On Entry: SortLocale – Locale to use for linguistic sorting data
```

```
COMMENT
COMMENT  On Exit: Result: Set to true if the specified locale is a CJK
COMMENT       (Chinese/Japanese/Korean) locale that uses third byte in Unicode weight
COMMENT

SET Result to false

CASE SortLocale
   "0x0404":  // Taiwan (Stroke Count)
   "0x0804":  // China (Pronunciation)
   "0x0c04":  // Hong Kong (Stroke Count)
   "0x1004":  // Singapore (pronunciation)
   "0x1404":  // Macau (pronunciation)
   "0x20804": // China (Stroke Count)
   "0x21004": // Singapore (Stroke Count)
   "0x21404": // Macau (Stroke Count)
   "0x30404": // Taiwan (Bopomofo)
   "0x40411": // Japanese (Radical / Stroke)
        SET Result to true
ENDCASE

RETURN Result
```

3.1.5.2.14 SpecialCaseHandler

This algorithm specifies the special processing that is needed based on a different script member type.

```
COMMENT SpecialCaseHandler
COMMENT
COMMENT  On Entry:  SourceString    - Source Unicode String
COMMENT             SourceIndex     - Current Index within source
COMMENT                               string
COMMENT             UnicodeWeights  - String of UnicodeWeightType to
COMMENT                               append additional weight(s) to
COMMENT             ExtraWeights    - String of ExtraWeightType to
COMMENT                               append extra weight(s) to if
COMMENT                               needed
COMMENT             SpecialWeights  - String of SpecialWeightType to
COMMENT                               append special weight(s) to
COMMENT                               if needed
COMMENT             SortLocale      - Locale to use for linguistic
COMMENT                               sorting data
COMMENT             IsKoreanLocale  - True if this locale needs
COMMENT                               Korean special casing of the
COMMENT                               ScriptMember value
COMMENT  On Exit:   SourceIndex     - Index of last character
COMMENT                               processed, caller will need to
COMMENT                               loop increment to continue
COMMENT                               Korean Jamo cases can increment
COMMENT                               this beyond its input value
COMMENT             UnicodeWeights  - The UnicodeWeight of the
COMMENT                               processed character(s) is
COMMENT                               appended to this string.
COMMENT             ExtraWeights    - The ExtraWeight, if any, of
COMMENT                               the processed character(s) is
```

```
COMMENT                                 appended to this string.
COMMENT             SpecialWeights  - The Special Weight, if any,
COMMENT                                 of the processed character(s)
COMMENT                                 is appended to this string.
COMMENT

PROCEDURE SpecialCaseHandler (INSourceString : Unicode String
INOUT SourceIndex : 32 bit integer
INOUT UnicodeWeights : UnicodeWeightType String,
INOUT ExtraWeights : ExtraWeightType String,
INOUT SpecialWeights : SpecialWeightType String,
IN SortLocale : LCID,
IN IsKoreanLocale : boolean)

// Get the weight for the current character
SET CharacterWeights to CALL GetCharacterWeights WITH
    (SortLocale, SourceString[SourceIndex])
CASE CharacterWeight.ScriptMember OF
    UNSORTABLE :
        // Character is unsortable, so skip it
        RETURN
    NONSPACE_MARK :
        // Character is a nonspace mark, so only store the
        // diacritic weight.
        If (Length(DiacriticWeights) is greater than 0) THEN
           SET last DiacriticWeight in DiacriticWeights to
           DiacritcWeight + CharacterWeights.DiacrticWeight
        ELSE
           APPEND CharacterWeights.DiacriticWeight to DiacriticWeights as a BYTE
        ENDIF
        RETURN
    EXPANSION :
        // Expansion character, each character has 2 weights, store
        // each weight separately
        SET Weights to CALL GetExpansionWeights WITH
            (SourceString[SourceIndex], SortLocale)
        // Store the appropriate weights, there should be 2 or 3
        FOR each Weight in Weights
            // Store the weight of the first character of the
            // expansion
            SET UnicodeWeight to CALL CorrectUnicodeWeight WITH
                        (Weights, IsKoreanLocale)
            APPEND UnicodeWeight to UnicodeWeights
            APPEND Weights.DiacriticWeight to DiacriticWeights as a BYTE
            APPEND Weights.CaseWeight to CaseWeights as a BYTE
        ENDFOR
        RETURN
    PUNCTUATION :
        SET Position to Length(UnicodeWeights) as 16 bit integer
        APPEND Position into SpecialWeights as 16 bit integer
        SET SpecialWeight to CALL MakeUnicodeWeight WITH
                (CharacterWeight.ScriptMember,
                CharacterWeight.PrimaryWeight, False)
        APPEND SpecialWeight to SpecialWeights as 16 bit integer
        RETURN
    SYMBOL_1 :
    SYMBOL_2 :
    SYMBOL_3 :
    SYMBOL_4 :
```

```
SYMBOL_5 :
SYMBOL_6 :
    // Character is a symbol, store Unicode Weights
    SET UnicodeWeight to CALL CorrectUnicodeWeight WITH
                        (Weights[0], IsKoreanLocale)
    APPEND UnicodeWeight to UnicodeWeights
    APPEND CharacterWeights.DiacriticWeight to DiacriticWeights as a BYTE
    APPEND CharacterWeights.CaseWeight to CaseWeights as a BYTE
    RETURN
EASTASIA_SPECIAL :
    // Get the primary and case weight of the current code point
    SET PrimaryWeight to UnicodeWeight.PrimaryWeight
    SET ExtraWeight to UnicodeWeight.CaseWeight
    // Mask off the bits we don't want
    SET ExtraWeight to (ExtraWeight & CaseMask) |
         CASE_EXTRA_WEIGHT_MASK
    // Special case Repeat and Cho-On
    //  PrimaryWeight = 0  => Repeat
    //    PrimaryWeight = 1  => Cho-On
    //    PrimaryWeight = 2+ => Kana
    IF PrimaryWeight is less than or equal to MAX_SPECIAL_PW THEN
        // If the script member of the previous character is
        // invalid, then give the special character
        // invalid weight (highest possible weight) so that it
        // will sort AFTER everything else.
        SET PreviousIndex to SourceIndex - 1
        SET UnicodeWeight.ScriptMember to MAP_INVALID_WEIGHT
        SET UnicodeWeight.PrimaryWeight to MAP_INVALID_WEIGHT
        WHILE PreviousIndex is greater than or equal to 0
        SET PreviousWeight to CALL GetCharacterWeights WITH
             (SortLocale, SourceString[PreviousIndex])
            IF PreviousWeight.ScriptMember is less than
                     EASTASIA_SPECIAL THEN
                IF PreviousWeight.ScriptMember is not equal to
                      EXPANSION THEN
                // UNSORTABLE or NONSPACE_MARK
                // Ignore these since we're trying to get the
                // previous ScriptMember/PrimaryWeight
                DECREMENT PreviousIndex
                CONTINUE WHILE PreviousIndex
                ENDIF
            ELSE IF PreviousWeight.ScriptMember is equal to
                     EASTASIA_SPECIAL THEN
                IF PreviousWeight.PrimaryWeight is less than or equal to
                      MAX_SPECIAL_PW THEN
                // Handle case where two special chars follow
                // each other.  Keep going back in the string
                DECREMENT PreviousIndex
                CONTINUE WHILE PreviousIndex
                ENDIF
            SET UnicodeWeight to
            CALL MakeUnicodeWeight WITH (KANA,
                  PreviousWeight.PrimaryWeight, IsKoreanLocale)
            // Only build weights W6 & W7 if the previous
            // character is KANA.
            // ignores W4 & W5
            // Always:
            // W6 = previous CW & ISOLATE_KANA
            SET PreviousExtraWeight to PreviousWeight.CaseWeight
```

```
                // Mask off the bits we don't want
                SET PreviousExtraWeight to CASE_EXTRA_WEIGHT_MASK |
                        (PreviousExtraWeight & CaseMask)
                // We ignore kana and width
                // so these are merely CASE_EXTRA_WEIGHT_MASK
                SET ExtraWeight.W6 to CASE_EXTRA_WEIGHT_MASK
                SET ExtraWeight.W7 to CASE_EXTRA_WEIGHT_MASK
                // Repeat is already done, which is:
                // UW = previous UW (set above)
                // W5 = ignored
                // W7 = previous CW & ISOLATE_WIDTH (done above)
                IF PrimaryWeight is not equal to PW_REPEAT THEN
                    // Cho-On:
                    // UW = previous UW & CHO_ON_UW_MASK
                    // W5 = ignored
                    // W7 = current CW & ISOLATE_WIDTH (done above)
                    SET UnicodeWeight.PrimaryWeight to
                    UnicodeWeight.PrimaryWeight & CHO_ON_PW_MASK
                ENDIF
               // Append the calculated ExtraWeight
               //   APPEND ExtraWeight to ExtraWeights
            ELSE
               // The previous weight is not EASTASIA_SPECIAL, so just
               // store the previous weight
               SET UnicodeWeight to CorrectUnicodeWeight
                         (PreviousWeight, IsKoreanLocale)
               // Append the weight we found
               APPEND UnicodeWeight to UnicodeWeights
            ENDIF
        ENDWHILE
        ELSE
            // Kana
            //    ScriptMember = KANA
            //    PrimaryWeight = current PrimaryWeight
            //    W4 = current CaseWeight & ISOLATE_SMALL
            //    W5 = WT_FIVE_KANA
            //    W6 = current CaseWeight & ISOLATE_KANA
            //    W7 = current CaseWeight & ISOLATE_WIDTH
            SET UnicodeWeight to CALL MakeUnicodeWeight WITH ( KANA,
                     CharacterWeight.PrimaryWeight, IsKoreanLocale)
            APPEND UnicodeWeight to UnicodeWeights
            SET TempExtraWeight.W4 to ExtraWeight & ISOLATE_SMALL
            SET TempExtraWeight.W5 to WT_FIVE_KANA
            SET TempExtraWeight.W6 to ExtraWeight & ISOLATE_KANA
            SET TempExtraWeight.W7 to ExtraWeight & ISOLATE_WIDTH
            APPEND TempExtraWeight to ExtraWeights
        ENDIF
            APPEND CharacterWeight.DiacriticWeight to DiacriticWeights as a BYTE
            APPEND MIN_CW to CaseWeights as a BYTE
            RETURN

    JAMO_SPECIAL :
        // See if it's a leading Jamo
        IF (CALL IsJamoLeading(SourceString[SourceIndex])) is true
                                        THEN
            // If the characters beginning at SourceIndex are a valid
            // old Hangul composition, create the SortKey
            // according to the old Hangul rule
            SET OldHangulCount to
```

```
            CALL MapOldHangulSortKey WITH (SourceString,
              SourceIndex, SortLocale, UnicodeWeights, IsKoreanLocale)
            IF OldHangulCount is greater than 0 THEN
                // Decrement OldHangulCount because the caller's loop
                // will increment the SourceIndex as well
                DECREMENT OldHangulCount
                SET SourceIndex to SourceIndex + OldHangulCount
                RETURN
            ENDIF
            ENDIF
           // Otherwise, fall back to the normal behavior
           // No special case on the character, so store the Jamo's
           // weights.
           // We store the real script member in the diacritic weight
           // in the tables since both the diacritic weight and the
           // case weight are not used in Korean
           // For example, from unisort.txt:
           // 0x1101 4 84 83 2 ;   Choseong Ssangkiyeok
           // Field 2 has a value of 4 to trigger the code case for JAMO_SPECIAL.
           // Field 3 (84) is the real primary weight for this Jamo.
           // Field 4 (83) is the real script member for this Jamo.
           SET UnicodeWeight to CALL MakeUnicodeWeight WITH
               (CharacterWeight.DiacriticWeight,
                  CharacterWeight.PrimaryWeight, IsKoreanLocale)
            APPEND UnicodeWeight to UnicodeWeights
            APPEND MIN_DW to DiacriticWeights as a BYTE
            APPEND MIN_CW to DiacriticWeights as a BYTE
            RETURN
    EXTENSION_A :
        // Extension A gives us two weights
        // UnicodeWeight = SM_EXT_A, AW_EXT_A, AW, DW
        // First Weight
        SET UnicodeWeight to CALL MakeUnicodeWeight WITH
             (SCRIPT_MEMBER_EXT_A, PRIMARY_WEIGHT_EXT_A,
                                IsKoreanLocale)
        APPEND UnicodeWeight to UnicodeWeights
        // Since the script member is our flag for this EXTENSION_A special
        // case, the real weights are in fields 2 & 3.
        // Example:
        // From unisort.txt:
        //  0x3400 5 16 2 2 ; □  CJK Unified Ideographs Extension A
        // Field 2 is the script member.
        // Field 3 is the primary weight.
        // Second Weight
        SET UnicodeWeight to CALL MakeUnicodeWeight WITH
              (CharacterWeight.PrimaryWeight,
                    CharacterWeight.DiacriticWeight, false)
        APPEND UnicodeWeight to UnicodeWeights
        APPEND MIN_DW to DiacriticWeights as a BYTE
        APPEND MIN_CW to DiacriticWeights as a BYTE
        RETURN
ENDCASE
```

3.1.5.2.15 GetPositionSpecialWeight

This algorithm specifies the retrieval of special weight based on the source index.

```
COMMENT GetPositionSpecialWeight
COMMENT
COMMENT  On Entry:  Position - Position to calculate weight for
COMMENT
COMMENT  On Exit:   Weight   - Resulting weight
COMMENT

PROCEDURE GetPositionSpecialWeight(IN Position : 32 bit integer,
                                   OUT Weight : 16 bit integer)

// We need to add some bits (0x8003) to adjust the weight and because
// some bits are expected. Since we're setting 0x3, we rotate source
// index 2 bits so as to not lose the precision.

// Note that if SourceIndex is larger than 0x1FFF, then some bits
// will be lost on the conversion to 16 bits. Presumably if a string
// is over 8191 characters long, they will differ well before this
// point, so the lost information is irrelevant.

SET Weight to (SourceIndex << 2) | 0x8003
RETURN Weight
```

3.1.5.2.16 MapOldHangulSortKey

This algorithm specifies the generation of Unicode weight based on the strings at the specified index that have a special Old Hangul sequence.

```
COMMENT MapOldHangulSortKey
COMMENT
COMMENT  On Entry:  SourceString   - Unicode String to test
COMMENT             SourceIndex    - Index of leading Jamo to start
COMMENT                              from
COMMENT             SortLocale     - Locale to use for linguistic
COMMENT                              sort data
COMMENT             UnicodeWeights - String to store any Unicode
COMMENT                              weight found
COMMENT                              for this character(s)
COMMENT
COMMENT  On Exit:   CharactersRead - Number of old Hangul found
COMMENT             UnicodeWeights - Any Unicode weights found are
COMMENT                              appended
COMMENT

PROCEDURE MapOldHangulSortKey(IN SourceString : Unicode String,
                  IN SourceIndex : 32 bit integer,
                  IN SortLocale : LCID,
                  IN OUTUnicodeWeights : String of UnicodeWeightType,
                  IN IsKoreanLocale : Boolean,
                  OUT CharactersRead : 32 bit integer)

SET CurrentIndex to SourceIndex
SET JamoSortInfo to empty JamoSortInfoType

// Get any Old Hangul Leading Jamo composition for our Leading Jamo
SET JamoClass to CALL GetJamoComposition WITH (SourceString,
                SourceIndex, "Leading Jamo Class", JamoSortInfo)
```

```
IF JamoClass is equal to "Vowel Jamo Class" THEN
    // We have a Vowel Jamo, try to find an
    // Old Hangul Vowel Jamo composition.
    SET JamoClass to CALL GetJamoComposition WITH (SourceString,
                SourceIndex, "Vowel Jamo Class", JamoSortInfo)
ENDIF

IF JamoClass is equal to "Trailing Jamo Class" THEN
    // We have a Trailing Jamo, try to find an
    // Old Hangul Trailing Jamo composition.
    SET JamoClass CALL GetJamoComposition WITH (SourceString,
                SourceIndex, "Trailing Jamo Class", JamoSortInfo)
ENDIF

// If we have a valid leading and vowel sequence and this is
// old Hangul...
IF JamoSortInfo.OldHangulFlag is true THEN

    // Compute the modern hangul syllable prior to this composition
    // Users formula from Unicode 3.0 Section 3.11 p54
    // "Hangul Syllable Composition"
    // This converts a U+11.. sequence to a U+AC00 character

    SET ModernHangul to (JamoSortInfo.LeadingIndex *
               NLS_JAMO_VOWELCOUNT + JamoSortInfo.VowelIndex) *
               NLS_JAMO_TRAILING_COUNT + JamoSortInfo.TrailingIndex +
               NLS_HANGUL_FIRST_SYLLABLE

    IF JamoSortInfo.FillerUsed is true THEN
        // If the filler is used, sort before the modern Hangul,
        // instead of after
        DECREMENT ModernHangul

        // If we fall off the modern Hangul syllable block...
        IF ModernHangul is less than NLS_HANGUL_FIRST_SYLLABLE THEN
            // Sort after the previous character
            // (Circled Hangul Kiyeok A)
           SET ModernHangul to 0x326e
        ENDIF

        // Shift the leading weight past any old Hangul
        // that sorts after this modern Hangul
        SET JamoSortInfo.LeadingWeight to
            JamoSortInfo.LeadingWeight + 0x80
    ENDIF

    // Store the weights
    SET CharacterWeight to CALL GetCharacterWeights WITH (ModernHangul)
    SET UnicodeWeight to CALL CorrectUnicodeWeight
            WITH (CharacterWeight, IsKoreanLocale)
    APPEND UnicodeWeight to UnicodeWeights

    // Add additional weights
    SET UnicodeWeight to CALL MakeUnicodeWeight WITH
            (ScriptMember_Extra_UnicodeWeight,
             JamoSortInfo.LeadingWeight, false)
    APPEND UnicodeWeight to UnicodeWeights

    SET UnicodeWeight to CALL MakeUnicodeWeight WITH
```

```
            (ScriptMember_Extra_UnicodeWeight,
             JamoSortInfo.VowelWeight, false)

    APPEND UnicodeWeight to UnicodeWeights
    SET UnicodeWeight to CALL MakeUnicodeWeight WITH
            (ScriptMember_Extra_UnicodeWeight,
             JamoSortInfo.TrailingWeight, false)

    APPEND UnicodeWeight to UnicodeWeights

    // Return the characters consumed
    SET CharactersRead to CurrentIndex - SourceIndex
    RETURN CharactersRead
ENDIF

// Otherwise it isn't a valid old Hangul composition
// and we don't do anything with it

SET CharactersRead to 0
RETURN CharactersRead
```

3.1.5.2.17 GetJamoComposition

This algorithm specifies the strings at the specified index that form a valid Old Hangul character that is composed of a Jamo character sequence.

```
COMMENT GetJamoComposition
COMMENT
COMMENT  On Entry:  SourceString - Unicode String to test
COMMENT             CurrentIndex - Index of leading Jamo to start from
COMMENT             JamoClass    - Class of Jamo to look for
COMMENT             JamoSortInfo - Information about the current
COMMENT                           sequence
COMMENT  On Exit:   JamoSortInfo - Updated with information about
COMMENT                            the new sequence
COMMENT             SourceIndex  - Updated to next character if
COMMENT                            Jamo is found
COMMENT             NewJamoClass - New class to look for next
COMMENT
COMMENT NOTE: This function assumes the character at SourceString
COMMENT       [SourceIndex] is a leading Jamo.
COMMENT       Ie: IsJamo() returned true
COMMENT

PROCEDURE GetJamoComposition (IN SourceString : Unicode String,
                          INOUT CurrentIndex : 32 bit integer,
                          IN JamoClass : enumeration,
                          INOUT JamoSortInfo : JamoSortInfoType,
                          OUT NewJamoClass : enumeration)

SET CurrentCharacter to SourceString[CurrentIndex]

// Get the Jamo information for the current character
SET JamoStateData to CALL GetJamoStateData WITH (CurrentCharacter)
SET JamoSortInfo to CALL UpdateJamoSortInfo
    WITH (JamoClass, JamoStateData, JamoSortInfo)
```

```
// Move on to the next character
INCREMENT CurrentIndex

WHILE CurrentIndex is less than Length(SourceString)
    SET CurrentCharacter to SourceString[CurrentIndex]

    IF CALL IsJamo WITH (CurrentCharacter) is not true THEN
        // The current character is not a Jamo,
        // we are done checking for a Jamo composition
        SET NewJamoClass to "Invalid Jamo Sequence"
        RETURN
    ENDIF

    IF CurrentCharacter is equal to 0x1160 THEN
        SET JamoSortInfo.FillerUsed to true
    ENDIF

    // Get the Jamo class of it
    IF CALL IsJamoLeading WITH (CurrentCharacter) is true THEN
        SET NewJamoClass to "Leading Jamo Class"
    ELSE IF CALL IsJamoTrailing WITH (CurrentCharacter) is true THEN
        SET NewJamoClass to "Trailing Jamo Class"
    ELSE
        SET NewJamoClass to "Vowel Jamo Class"
    ENDIF

    IF JamoClass is not equal to NewJamoClass THEN
        RETURN NewJamoClass
    ENDIF

    // Push the current Jamo (SourceString[CurrentIndex])
    // into the state machine to check if we have a valid
    // old Hangul composition. During the check we will also
    // update the sortkey result in:
    JamoSortInfo

    // Find the new record
    SET JamoStateData to CALL FindNewJamoState
        WITH (CurrentCharacter, JamoStateData)

    // We didn't find a valid old Hangul composition for the current
    // character so return the current Jamo class
    // (JamoClass and NewJamoClass are identical)
    IF JamoStateData is null THEN
        RETURN NewJamoClass
    ENDIF

    // We found a match, so update our info.
    SET JamoSortInfo to CALL UpdateJamoSortInfo
        WITH (JamoClass, JamoStateData, JamoSortInfo)

    // We are still in a valid old Hangul composition.
    //Go check the next character.
    INCREMENT CurrentIndex

ENDWHILE CurrentIndex

SET NewJamoClass to "Invalid Jamo Sequence"
```

```
RETURN NewJamoClass
```

3.1.5.2.18 GetJamoStateData

This algorithm specifies the retrieval of state machine information to check whether the specified Jamo sequence forms a valid Old Hangul character.

```
COMMENT GetJamoStateData
COMMENT
COMMENT  On Entry:  Character     - Unicode Character to get Jamo
COMMENT                             information for
COMMENT
COMMENT  On Exit:   JamoStateData - Jamo state information from
COMMENT                             the data file
COMMENT
COMMENT  Jamo State information looks like this in the database:
COMMENT
COMMENT   SORTTABLES
COMMENT     ...
COMMENT     JAMOSORT395
COMMENT     ...
COMMENT   0x11724
COMMENT     0x1172 0x00 0x00 0x11 0x00 0x380x03; U+1172
COMMENT     0x1161 0x01 0x00 0x00 0x00 0x000x01; U+1172,1161
COMMENT     0x1175 0x01 0x00 0x11 0x1b 0x3a0x00; U+1172,1161,1175
COMMENT     0x1169 0x01 0x00 0x11 0x1b 0x3f0x00; U+1172,1169

PROCEDURE GetJamoStateData (IN Character : Unicode Character,
                            OUT JamoStateData : JamoStateDateType)

// Get the Jamo section for this character.
// If Character was 0x1172, this would access the following section:
// 0x11724
//    0x1172 0x00 0x00 0x11 0x00 0x38 0x03 ; U+1172           record 0
//    0x1161 0x01 0x00 0x00 0x00 0x00 0x01 ; U+1172,1161      record 1
//    0x1175 0x01 0x00 0x11 0x1b 0x3a 0x00 ; U+1172,1161,1175 record 2
//    0x1169 0x01 0x00 0x11 0x1b 0x3f 0x00 ; U+1172,1169      record 3
//    |     |    |    |    |    |    |       |
// Field 1  2    3    4    5    6    7       Comment

OPEN SECTION JamoSection
     where name is SORTTABLES\JAMOSORT\[Character] from unisort.txt

// Now open the first record
SELECT RECORD JamoRecord FROM JamoSection WHERE record index is 0

// Now gather the information from that record.
SET JamoStateData.OldHangulFlag   to JamoRecord.Field2
SET JamoStateData.LeadingIndex    to JamoRecord.Field3
SET JamoStateData.VowelIndex      to JamoRecord.Field4
SET JamoStateData.TrailingIndex   to JamoRecord.Field5
SET JamoStateData.ExtraWeight     to JamoRecord.Field6
SET JamoStateData.TransitionCount to JamoRecord.Field7

// Remember the record
SET JamoStateData.DataRecord to JamoRecord
```

```
RETURN JamoStateData
```

3.1.5.2.19 FindNewJamoState

This algorithm specifies retrieval of a new state from the state machine for Jamo processing.

```
COMMENT FindNewJamoState
COMMENT
COMMENT  On Entry:  JamoCharacter    - Unicode Character to get Jamo
COMMENT                                information for
COMMENT             JamoStateData    - Current Jamo state information
COMMENT
COMMENT  On Exit:   JamoStateData    - New Jamo state record from the
COMMENT                                data file, null if an
COMMENT                                appropriate state record is
COMMENT                                not found.
COMMENT

PROCEDURE FindNewJamoState(IN JamoCharacter : Unicode Character,
                           INOUT JamoStateData : JamoStateDataType)

// The current JamoStateData.DataRecord points to the base record.
// There are JamoStateData.TransitionCount following records that may
// match the input JamoCharacter, we're looking for the first one
SET DataRecord to JamoStateData.DataRecord

WHILE JamoStateData.TransitionCount is greater than 0
       // advance to the next record in the data and test if
       // it is the correct record for JamoCharacter
       ADVANCE DataRecord to next record in data table
       IF DataRecord.Field1 is equal to JamoCharacter THEN
          // Found a record, get its info and return it
          // Now gather the information from that record.
          SET JamoStateData.OldHangulFlag   to JamoRecord.Field2
          SET JamoStateData.LeadingIndex    to JamoRecord.Field3
          SET JamoStateData.VowelIndex      to JamoRecord.Field4
          SET JamoStateData.TrailingIndex   to JamoRecord.Field5
          SET JamoStateData.ExtraWeight     to JamoRecord.Field6
          SET JamoStateData.TransitionCount to JamoRecord.Field7

          // Remember the record
          SET JamoStateData.DataRecord to JamoRecord

              RETURN JamoStateData
ENDWHILE

// record not found, return null
SET JamoStateData to null
RETURN JamoStateData
```

3.1.5.2.20 UpdateJamoSortInfo

This algorithm specifies the update of Jamo sorting information based on the current state of the state machine for Jamo processing.

```
COMMENT UpdateJamoSortInfo
COMMENT
COMMENT  On Entry:  JamoClass     - The current Jamo Class
COMMENT             JamoStateData - Information about the new
COMMENT                             character state
COMMENT             JamoSortInfo  - Information about the character
COMMENT                             state
COMMENT
COMMENT  On Exit:   JamoSortInfo  - Updated with information about
COMMENT                             the new state based on JamoClass
COMMENT                             and JamoSortData
COMMENT

PROCEDURE UpdateJamoSortInfo(IN JamoClass : enumeration,
                             IN JamoStateData : JamoStateDataType,
                             INOUT JamoSortInfo : JamoSortInfoType)

// Record if this is a Jamo unique to old Hangul
SET JamoSortInfo.OldHangulFlag to
    JamoSortInfo.OldHangulFlag | JamoStateData.OldHangulFlag

//  Update the indices if the new ones are higher than the current
//  ones.
IF JamoStateData.LeadingIndex
   is greater than JamoSortInfo.LeadingIndex THEN
   SET JamoSortInfo.LeadingIndex to JamoStateData.LeadingIndex;
ENDIF

IF JamoStateData.VowelIndex
   is greater than JamoSortInfo.VowelIndex THEN
   SET JamoSortInfo.VowelIndex to JamoStateData.VowelIndex;
ENDIF

IF JamoStateData.TrailingIndex
   is greater than JamoSortInfo.TrailingIndex THEN
   SET JamoSortInfo.TrailingIndex to JamoStateData.TrailingIndex;
ENDIF

//  Update the extra weights according to the current Jamo class.
CASE JamoClass OF
   "Leading Jamo Class":
      IF JamoStateData.ExtraWeight
         is greater than JamoSortInfo.LeadingWeight THEN
         SET JamoSortInfo.LeadingWeight to JamoStateData.ExtraWeight
      ENDIF

   "Vowel Jamo Class":
      IF JamoStateData.ExtraWeight
         is greater than JamoSortInfo.VowelWeight THEN
         SET JamoSortInfo.VowelWeight to JamoStateData.ExtraWeight
      ENDIF

   "Trailing Jamo Class":
   IF JamoStateData.ExtraWeight
      is greater than JamoSortInfo.TrailingWeight THEN
      SET JamoSortInfo.TrailingWeight to JamoStateData.ExtraWeight
   ENDIF
ENDCASE
```

```
RETURN JamoSortInfo
```

3.1.5.2.21 IsJamo

This algorithm specifies the check for a valid Jamo character.

```
COMMENT IsJamo
COMMENT
COMMENT  On Entry:  SourceCharacter - Unicode Character to test
COMMENT
COMMENT  On Exit:   Result          - true if SourceCharacter is in
COMMENT                               the Jamo range
COMMENT

PROCEDURE IsJamoLeading(IN SourceCharacter : Unicode Character,
                        OUT Result: boolean)

IF (SourceCharacter is greater than or equal to NLS_CHAR_FIRST_JAMO)
     and
   (SourceCharacter is less than or equal to NLS_CHAR_LAST_JAMO) THEN
     SET Result to true
ELSE
     SET Result to false
ENDIF

RETURN Result
```

3.1.5.2.22 IsJamoLeading

This algorithm checks whether the specified Jamo character is a leading Jamo.

```
COMMENT IsJamoLeading
COMMENT
COMMENT  On Entry:  SourceCharacter - Unicode Character to test
COMMENT
COMMENT  On Exit:   Result          - true if SourceCharacter is a
COMMENT                               leading Jamo
COMMENT
COMMENT NOTE: Only call this if the character is known to be a Jamo
COMMENT       syllable. This function only helps distinguish between
COMMENT       the different types of Jamo, so only call it if
COMMENT       IsJamo() has returned true.
COMMENT

PROCEDURE IsJamoLeading(IN SourceCharacter : Unicode Character,
                        OUT Result: boolean)

IF SourceCharacter is less than NLS_CHAR_FIRST_VOWEL_JAMO THEN
     SET Result to true
ELSE
     SET Result to false
ENDIF

RETURN Result
```

3.1.5.2.23 IsJamoTrailing

This algorithm checks whether the specified Jamo character is a trailing Jamo.

```
COMMENT IsJamoTrailing
COMMENT
COMMENT  On Entry:  SourceCharacter - Unicode Character to test
COMMENT
COMMENT  On Exit:   Result          - true if this is a trailing Jamo
COMMENT
COMMENT NOTE: Only call this if the character is known to be a Jamo
COMMENT       syllable. This function only helps distinguish between
COMMENT       the different types of Jamo, so only call it if
COMMENT       IsJamo() has returned true.
COMMENT

PROCEDURE IsJamoTrailing(IN SourceCharacter : Unicode Character,
                         OUT Result: boolean)

IF SourceCharacter is greater than
   or equal to NLS_CHAR_FIRST_VOWEL_JAMO THEN
     SET Result to true
ELSE
     SET Result to false
ENDIF

RETURN Result
```

3.1.5.2.24 InitKoreanScriptMap

This algorithm specifies the initialization of a data structure that is needed for the special processing of Korean script members.

```
COMMENT InitKoreanScriptMap
COMMENT
COMMENT  On Entry:  global KoreanScriptMap - presumed to be null
COMMENT
COMMENT  On Exit:   global KoreanScriptMap - initialized to map
COMMENT                                      scripts to Korean
COMMENT
COMMENT  This procedure initializes the Korean, causing ideographic
COMMENT  scripts to sort prior to other scripts for the Korean.
COMMENT

PROCEDURE InitKoreanScriptMap

SET KoreanScriptMap to new array of 256 null bytes

// Initialize the "scripts" prior to first script (Latin, script 14)
FOR counter is 0 to FIRST_SCRIPT - 1
     SET KoreanScriptMap[counter] to counter
ENDFOR counter

// For Korean the Ideographs sort to the first script,
// so start with that index
SET NewScript to FIRST_SCRIPT
```

```
// Test if the IDEOGRAPH script is part of a multiple weights script

// For convenience we're hard coding the information from the
// unisort.txt section SORTTABLES\MULTIPLEWEIGHTS
// IDEOGRAPHS are 128 through 241,
// we map them to FIRST_SCRIPT through 127
FOR counter is IDEOGRAPH to 241
     SET KoreanScriptMap[counter] to NewScript
     INCREMENT NewScript
ENDFOR

// Now set the remaining unset scripts the next NewScript value
FOR counter is 0 to MAX_SCRIPTS - 1
     // If the value has not been set yet, set it to the next value
     IF KoreanScriptMap[counter] is null THEN
          SET KoreanScriptMap[counter] to NewScript
          INCREMENT NewScript
     ENDIF
ENDFOR
```

3.1.6 Timer Events

There are no timer events.

3.1.7 Other Local Events

There are no other local events.

4 Protocol Examples

There are no protocol examples.

5 Security

The following sections specify security considerations for implementers of the Windows Protocols Unicode Reference.

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

There are no security parameters.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows NT® operating system
- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

7 Appendix B: Windows Sorting Weight Table

This section contains links to detailed character weight specifications that permit consistent sorting and comparison of Unicode strings. The data is not used by itself but is used as one of the inputs to the comparison algorithm. The layout and format of data in this file is also specified there.

- Microsoft Windows NT® 4.0 operating system through Windows Server® 2003 operating system [MSDN-SWT/W2K3]
- Windows Vista® operating system [MSDN-SWT/Vista]
- Windows Server® 2008 operating system [MSDN-SWT/W2K8]
- Windows® 7 operating system through Windows Server® 2008 R2 operating system[MSDN-SWT/Win7]
- 1250 (Central Europe)

  1. W
  2. WW
- 874(Thai)

  1. X
  2. XX
- 1251(Cyrillic)

  1. Y
  2. XX
- 1252 (Latin I)

  1. Z
  2. ZZ

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

W