

Unicode in BS2000/OSD

Introduction

Comments... Suggestions... Corrections...

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to manuals@fujsu-siemens.com.

Certified documentation according to DIN EN ISO 9001:2000

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Copyright © Fujitsu Siemens Computers GmbH 2007.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Contents

1	Introduction	7
1.1	Target group	7
1.2	Concept of the manual	8
2	Overview	9
2.1	What is Unicode?	9
2.2	Motivation for Unicode support	10
3	Unicode encodings	13
3.1	UTF-8	15
3.2	UTF-EBCDIC (UTFE)	17
3.3	UTF-16	22
3.4	UTF-32	22
3.5	Normalization	23
3.6	Sort sequence	24
4	Unicode in BS2000/OSD	27
4.1	Basics of Unicode support in BS2000/OSD	27
4.2	Overview of the affected interfaces	29
4.3	Configuring the terminal emulation MT9750	32

5	Unicode adjustments in BS2000 applications	35
5.1	Character handling in BS2000/OSD (XHCS)	36
5.2	Representing and processing Unicode characters with COBOL	37
5.3	Advanced Interactive Debugger (AID)	38
5.4	Storing, searching for and managing Unicode data in databases	39
5.4.1	Unicode concept in SESAM/SQL	39
5.4.2	Unicode concept in Oracle	41
5.5	Support for Unicode fields in formats	43
5.6	Outputting print jobs with Unicode data	45
5.6.1	Central printers (AFP-IPDS)	45
5.6.2	Decentralized printers (RSO)	45
5.7	Web integration of Unicode-capable applications (<i>WebTransactions</i>)	47
6	Unicode adjustments for file processing	49
6.1	Creating and editing Unicode files (EDT)	50
6.2	Converting and normalizing Unicode files (PERCON)	51
6.3	Sorting Unicode fields (SORT)	51
6.4	Transferring Unicode files (<i>openFT</i>)	52
7	Tips and tricks	53
7.1	Tips on SESAM/SQL	53
7.2	Tips on LMS	54
8	Appendix	57
8.1	Unicode products in BS2000/OSD: overview and dependencies	57
8.2	Extended BS2000 macros	59
8.3	Coded Character Set Names (CCSN): default values	61

8.4	Useful code tables	63
8.4.1	Conversion from ISO8859 to EBCDIC (BS2000/OSD) and vice versa	63
8.4.2	Unicode characters convertible to ISO8859.n	65
 Glossary		89
<hr/>		
Abbreviations		93
<hr/>		
Tables		95
<hr/>		
Related publications		97
<hr/>		
Index		99
<hr/>		

1 Introduction

Unicode combines all worldwide text symbols into a single character set. Unlike earlier coding systems, such as 7- or 8-bit encoding, Unicode is also independent of different vendors, systems and countries.

In the course of the increasing internationalization of software programs, and because more and more customers are opening up their BS2000/OSD applications to the Internet, which uses Unicode encoding, the Unicode character set is gaining in importance also within BS2000/OSD and its applications.

1.1 Target group

This Overview Guide is aimed at application programmers and system administrators who want to get an idea of what Unicode support is offered them in BS2000/OSD and which BS2000 components they need for this.

This guide does not replace the manuals for the individual products.

1.2 Concept of the manual

Chapter 2 provides a brief overview of Unicode and the motivation for Unicode support in BS2000/OSD.

Chapters 3 and 4 provide basic definitions and concepts relating to Unicode, which apply to all BS2000/OSD products affected by Unicode and which supplement their respective manuals.

Chapters 5 and 6 then explain the precise nature of the Unicode support in key individual applications and file-processing programs in BS2000/OSD.

In addition, chapter 7 offers tips and tricks for converting BS2000/OSD applications to Unicode.

In the appendix you will find useful tables to help you with Unicode conversion.

The manual is completed by a glossary, lists of abbreviations, tables, related publications and an index.

2 Overview

2.1 What is Unicode?

Unicode is an alphanumeric character set, standardized by the International Standardization Organization (ISO) and the Unicode Consortium, for the encoding of characters, letters, numbers, punctuation marks, syllable characters, special characters and ideograms. An ideogram is a graphic symbol that represents an idea and in so doing uses symbolic characters for abstract concepts or is composed of two or more pictograms. Unicode combines all worldwide text symbols in a single character set.

Unicode is universal

Unicode thus contains not only the letters of the Latin alphabet with all its country-specific peculiarities, but also the Greek, Cyrillic, Arabic, Hebrew and Thai alphabets as well as the so-called CJK scripts – the various Chinese, Japanese and Korean scripts. It also encodes mathematical, commercial and technical special characters. Unicode is therefore independent of languages and scripts and thus supports internationalization, i.e. the development of software programs which can be easily adapted to all possible languages and cultures. Conversely, Unicode also enables localization, i.e. the adjustment of software to "local" linguistic and cultural conditions without changing the program. This allows the setting of country-specific properties and the uniform display of symbols which are linked with base characters (normalization of diacritical marks).

Multibyte character sets

Before Unicode, most encoding systems represented characters using 7 or 8 bits, which limited the number of characters of the corresponding character sets to 128 or 256. Among the best-known are ASCII and EBCDIC, which both exist in different national and in some cases also vendor-specific forms.

Unicode, on the other hand, is vendor- and system-independent. It uses character sets (also known as coded character sets or code sets) of 2 or 4 bytes to encode each text character, which the ISO refers to as UCS-2 (Universal Character Set 2) or UCS-4.

In UCS-2 the first 256 of the maximum 65,536 characters correspond to the characters of the character set ISO Latin-1 (ISO 8859-1). ISO Latin-1 is widely used and combines the characters of western European languages.

UTF-8 and UTF-16

Instead of the name UCS-2 defined by the ISO, the name UTF-16 (UCS Transformation Format 16-bit) is often used and is a standard defined by the Unicode Consortium. UTF-16 also uses characters of 4 bytes in length (surrogates), which extends the number of characters to more than 1.1 million.

As well as UTF-16, UTF-8 (UCS Transformation Format 8-bit) is also widely used. UTF-8 represents every Unicode character in 1 to 4 bytes, depending on its position. The first 128 characters of UTF-8 correspond to ASCII.

For detailed information, visit the website of the Unicode Consortium at:

<http://www.unicode.org/standard/WhatIsUnicode.html>.

2.2 Motivation for Unicode support

The standard character set in BS2000/OSD is EBCDIC.DF.03IRV (EDF03IRV), a 7-bit character set whose characters correspond to the ASCII 7-bit character set extended to include the second control-character block of ISO8859-1. EDF03IRV contains only 95 printable characters.

You can extend it to 181 printable characters by introducing an 8-bit character set. This extension allows you to set up character sets for specific language areas, but it is not sufficient to map the different characters of all European languages, let alone all active languages, in a single-byte character set. The 8-bit EBCDIC tables familiar from BS2000 are based on the characters of the corresponding ISO8859 tables, even though not every ISO8859-n table has an equivalent in BS2000/OSD.

Applications in the Internet

More and more BS2000/OSD customers are opening up their BS2000/OSD applications to the Internet. The character set used in the Internet is UTF-8. The language Java uses UTF-16 to encode the data. Java applications and Java applets are linked to BS2000/OSD databases via Java Database Connectivity (JDBC). It is anticipated that converting the data from the Internet into single-byte encoding will not be sufficient in the future. Moreover, the literal transfer of words from a non-Latin text into the Latin script (transliteration) is not desired, or rather not possible because the relevant diacritical marks are not present in the set character set.

European languages

If not just western European but also eastern European address data is to be stored in a column of a database table, the pool of characters in an EBCDIC table is no longer sufficient, because international postal regulations on how to write an address state that the sender must specify the name and address of the recipient in the language of the country of destination and with its characters. Equally, the destination in foreign addresses must be written in capital letters in the notation of the destination country. This means that the character set of the "Name", "Address" and "City" columns of the database table must be able to accept Latin, Greek, Cyrillic etc. characters.

3 Unicode encodings

In Unicode, each character is assigned a number, a so-called code point.


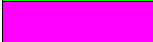





A Unicode code point is generally given in the form U+n, where n consists of 4 to 6 hexadecimal numbers.

The set of all code points forms the so-called code space. The code space of the Unicode standard V4 comprises 1,114,112 code points, most of which have not yet been assigned. The code space is divided into so-called planes, each containing 65,536 code points. The most important of these is Plane 0, the Basic Multilingual Plane (BMP), which covers the code points from U+0000 through U+FFFF. The code points from U+0000 through U+00FF correspond to those of ISO8859-1. Planes 15 and 16 are reserved by the Unicode standard for the definition of private characters (Private Use Area).

Plane		– 0FFF	– 1FFF	– 2FFF	– 3FFF	– 4FFF	– 5FFF	– 6FFF	– 7FFF	– 8FFF	– 9FFF	– AFFF	– BFFF	– CFFF	– DFFF	– EFFF	– FFFF
0	00000																
1	10000																
2	20000																
3	30000																
4	40000																
5	50000																
6	60000																
7	70000																
8	80000																
9	90000																
10	A0000																
11	B0000																
12	C0000																
13	D0000																
14	E0000																
15	F0000																
16	100000																

Table 1: Code space of the Unicode standard V4

Key:

	General scripts, symbols
	Chinese, Japanese and Korean scripts (CJK)
	Surrogates
	Private Use Area
	Composite characters
	Unused code points
	Tag characters

In Unicode support, it is of crucial importance how these code points are encoded in bytes. The Unicode Consortium defines three different coding possibilities for this: UTF-8, UTF-16 and UTF-32.

A special case, which is not directly part of the Unicode standard, is UTF-EBCDIC, a Unicode encoding for servers which use the EBCDIC character set.

The following sections provide details of the various coding possibilities as well as the normalization and sorting of Unicode strings.

3.1 UTF-8

UTF-8 uses a variable number of bytes for encoding the Unicode characters. The byte representation of the ASCII characters remains unchanged. For all other characters, the number of leading ones in the first byte indicates how many bytes belong to a character. Subsequent bytes always begin with 10.

UTF-8	Serialized bytes			
Unicode range	1st byte	2nd byte	3rd byte	4th byte
U+000000 - U+00007F	<i>0</i> nnnnnnn			
U+000080 - U+0007FF	<i>110</i> nnnnn	<i>10</i> nnnnnn		
U+000800 - U+00FFFF	<i>1110</i> nnnn	<i>10</i> nnnnnn	<i>10</i> nnnnnn	
U+010000 - U+10FFFF	<i>11110</i> nnn	<i>10</i> nnnnnn	<i>10</i> nnnnnn	<i>10</i> nnnnnn

Example

Unicode character	UTF-8
U+ 20AC (euro symbol)	<i>11100010 10000010 10101100</i>

UTF-8 has the advantage over older multibyte character sets that, with a character that is encoded with multiple bytes, none of the individual bytes represents a valid character.

Because UTF-8 is ASCII-compatible, the memory requirement for character strings with this type of Unicode encoding does not change in English. For all other Latin-x character sets, texts become on average 10% longer. Greek, Cyrillic or Arabic characters lie in the range between 128 and 2047 (U+00000080 through U+000007FF), making the memory requirements for texts in these languages around 70% higher.

The characters of the east-Asian languages lie in the range above this and require approximately 3 bytes per character.







For standard-compliant UTF-8 encoding, the shortest encoding always applies, i.e. the code point U+7F must always be represented as 0111 1111 = x'7F' and not as the two-byte encoding 1100 0001 1011 1111 = x'C1BF'.

This rule results in the following table of valid UTF-8-byte allocations.

	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-	
-0	00	10	20	30	40	50	60	70						400-43F	800-FFF	1000-3FFF	
-1	01	11	21	31	41	51	61	71						440-47F	1000-1FFF	4000-7FFF	
-2	02	12	22	32	42	52	62	72						80-BF	480-4BF	2000-2FFF	8000-BFFF
-3	03	13	23	33	43	53	63	73						C0-FF	4C0-4FF	3000-3FFF	C000-FFFF
-4	04	14	24	34	44	54	64	74						100-13F	500-53F	4000-4FFF	10000-10FFFF
-5	05	15	25	35	45	55	65	75						140-17F	540-57F	5000-5FFF	
-6	06	16	26	36	46	56	66	76						180-1BF	580-5BF	6000-6FFF	
-7	07	17	27	37	47	57	67	77						1C0-1FF	5C0-5FF	7000-7FFF	
-8	08	18	28	38	48	58	68	78						200-23F	600-63F	8000-8FFF	
-9	09	19	29	39	49	59	69	79						240-27F	640-67F	9000-9FFF	
-A	0A	1A	2A	3A	4A	5A	6A	7A						280-2BF	680-6BF	A000-AFFF	
-B	0B	1B	2B	3B	4B	5B	6B	7B						2C0-2FF	6C0-6FF	B000-BFFF	
-C	0C	1C	2C	3C	4C	5C	6C	7C						300-33F	700-73F	C000-CFFF	
-D	0D	1D	2D	3D	4D	5D	6D	7D						340-37F	740-77F	D000-DFFF	
-E	0E	1E	2E	3E	4E	5E	6E	7E						380-3BF	780-7BF	E000-EFFF	
-F	0F	1F	2F	3F	4F	5F	6F	7F						3C0-3FF	7C0-7FF	F000-FFFF	

Table 2: Byte allocation in UTF-8 representation

Key:

	Single-byte encoding (number matches the Unicode code point)
	Subsequent byte of a multibyte representation
	First byte of a two-byte encoding
	First byte of a three-byte encoding
	Outside the code space of Unicode V4.0 (not a valid UTF-8 encoding)
	First byte of a four-byte encoding (outside the BMP)

3.2 UTF-EBCDIC (UTFE)

For systems which use EBCDIC, the Unicode Consortium has a technical report which proposes conversion of Unicode characters into EBCDIC:

The first step of the procedure described there extends the range of single-byte encoding of UTF-8 Unicode characters to include the second control-character block (U+80 through U+9F) and limits the range of subsequent bytes of a multibyte encoding to the range x'A0' through x'BF'. The resulting encoding is called modified UTF-8 (UTF-8MOD).

UTF-8MOD	Serialized bytes				
	1st byte	2nd byte	3rd byte	4th byte	5th byte
U+000000 - U+00007F	<i>0</i> nnnnnnn				
U+000080 - U+00009F	<i>100</i> nnnnn				
U+0000A0 - U+0003FF	<i>110</i> nnnnn	<i>101</i> nnnnn			
U+000400 - U+003FFF	<i>1110</i> nnnn	<i>101</i> nnnnn	<i>101</i> nnnnn		
U+004000 - U+03FFFF	<i>11110</i> nnn	<i>101</i> nnnnn	<i>101</i> nnnnn	<i>101</i> nnnnn	
U+040000 - U+10FFFF	<i>111110</i> n	<i>101</i> nnnnn	<i>101</i> nnnnn	<i>101</i> nnnnn	<i>101</i> nnnnn

In the second step, the standard conversion from ISO8859-n to EBCDIC.DF.04.n is used to transform the modified UTF-8 into UTF-EBCDIC (UTFE).

Example

Unicode character	UTF-8MOD	UTF-EBCDIC
U+ 20AC (euro symbol)	<i>11101000 10100101 10101100</i> E8 A5 AC	80 B2 BA

The resulting single-byte range of the UTFE encoding (see [table 4 on page 21](#)) corresponds to the character set EBCDIC.DF.03IRV (EDF03IRV), the standard character set in BS2000/OSD. Unlike the other Unicode encodings, UTFE can be parsed by all BS2000/OSD applications which are restricted to EDF03IRV. This applies particularly to BS2000/OSD itself. A BS2000/OSD command only ever contains characters from the single-byte range of UTFE. If a character from the multibyte range is included in the input string, a syntax error is correctly identified. All control characters are also contained in the single-byte range.



This conversion deviates from the proposal of the technical report of the Unicode standard, because the EBCDIC encoding in BS2000/OSD and in z/OS are not identical. This means that the UTFE encoding in ORACLE does not correspond to that of BS2000/OSD.









However, because both are based on the UTF-8MOD encoding, one form can easily be converted into the other via a suitable conversion table.

For a standard-compliant UTF-8MOD encoding, the shortest encoding always applies, i.e. the code point U+7F can, according to the above schema, only be represented by 0111 1111 = x'7F', not by the two-byte encoding 1100 0011 1011 1111 = x'C3BF'. This rule results in the following table 3 with the valid UTF-8MOD byte allocations as a pre-stage to UTFE.

	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	00	10	20	30	40	50	60	70	80	90				200-21F		4000-7FFF
-1	01	11	21	31	41	51	61	71	81	91				220-23F	400-7FF	8000-FFFF
-2	02	12	22	32	42	52	62	72	82	92				240-25F	800-BFF	10000-17FFF
-3	03	13	23	33	43	53	63	73	83	93				260-27F	C00-FFF	18000-1FFFF
-4	04	14	24	34	44	54	64	74	84	94				280-29F	1000-13FF	20000-27FFF
-5	05	15	25	35	45	55	65	75	85	95			A0-BF	2A0-2BF	1400-17FF	28000-2FFFF
-6	06	16	26	36	46	56	66	76	86	96			C0-DF	2C0-2DF	1800-1BFF	30000-37FFF
-7	07	17	27	37	47	57	67	77	87	97			E0-FF	2E0-2FF	1C00-1FFF	38000-3FFFF
-8	08	18	28	38	48	58	68	78	88	98			100-11F	300-31F	2000-23FF	40000-FFFFF
-9	09	19	29	39	49	59	69	79	89	99			120-13F	320-33F	2400-27FF	100000-10FFFF
-A	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A			140-15F	340-35F	2800-2BFF	
-B	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B			160-17F	360-37F	2C00-2FFF	
-C	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C			180-19F	380-39F	3000-33FF	
-D	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D			1A0-1BF	3A0-3BF	3400-37FF	
-E	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E			1C0-1DF	3C0-3DF	3800-3BFF	
-F	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F			1E0-1FF	3E0-3FF	3C00-3FFF	

Table 3: Byte allocation in UTF-8MOD representation

Key for [table 3](#) and [table 4](#):

	Single-byte encoding
	Second control-character block: single-byte encoding
	Subsequent byte of a multibyte representation
	First byte of a two-byte encoding
	First byte of a three-byte encoding
	First byte of a four-byte encoding
	First byte of a five-byte encoding
	Not a valid UTF-8MOD encoding

The following table 4 shows the byte allocation in UTF-EBCDIC representation (UTFE). This byte allocation is the result of converting UTF-8MOD to UTFE by means of the standard conversion from ISO8859-n to EBCDIC.DF.04.n.

The single-byte range of this UTFE encoding corresponds to the standard character set in BS2000/OSD (EBCDIC.DF.03IRV).

	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	00	01	02	03	85	09	86	7F	87	8D	8E	0B	0C	0D	0E	0F
-1	10	11	12	13	8F	92	08	97	18	19	9C	9D	1C	1D	1E	1F
-2	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
-3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
-4	20	A0	800- BFF	1000- 13FF	E0	400- 7FF	C00- FFF	1400- 17FF	1C00- 1FFF	8000- FFFF	60	2E	3C	28	2B	7C
-5	26	2400- 27FF	2800- 2BFF	2C00- 2FFF	2000- 23FF	3400- 37FF	3800- 3BFF	3C00- 3FFF	3000- 33FF	3E0- 3FF	21	24	2A	29	3B	9F
-6	2D	2F	C2	C4	C0	C1	C3	A0-BF	E0-FF	220- 23F	5E	2C	25	5F	3E	3F
-7	100000 - 10FFFF	120- 13F	140- 15F	160- 17F	100- 11F	1A0- 1BF	1C0- 1DF	1E0- 1FF	180- 19F	A8	3A	23	40	27	3D	22
-8	300- 31F	61	62	63	64	65	66	67	68	69	AB	BB	4000- 7FFF	FD	FE	B1
-9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	1800- 1BFF	B8	C0-DF	A4
-A	B5	AF	73	74	75	76	77	78	79	7A	A1	BF	200- 21F	3A0- 3BF	3C0- 3BF	AE
-B	A2	A3	A5	B7	A9	A7	B6	BC	BD	BE	AC	5B	5C	5D	B4	2E0- 2FF
-C	F9	41	42	43	44	45	46	47	48	49	AD	20000 - 27FFF	38000 - 3FFFF	10000 - 17FFF	18000 - 1FFFF	30000 - 37FFF
-D	A6	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	360- 37F	FA	FF
-E	320- 33F	40000- FFFFF	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
-F	30	1F	32	33	34	35	36	37	38	39	B3	7B	380- 39F	7D	340- 35F	7E

Table 4: Byte allocation in UTFE representation

3.3 UTF-16

In UTF-16, all Unicode characters between U+0000 and U+FFFF are encoded by 2 bytes. The characters in this range are often also known as the 2-byte Universal Character Set (UCS-2). All characters above U+FFFF are represented by 4 bytes, so-called surrogate pairs.

To uniquely identify these surrogate pairs, the range between U+D800 and U+DFFF has been reserved for them. The first 16 bits of such a pair always begin with x'110110' and the second with x'110111'. The remaining bits of the character to be represented are distributed over the remaining positions as in UTF-8.

Therefore, characters which are encoded with UTF-16, like those in UTF-8, do not have a fixed length. In practice, however, 2 bytes are sufficient. Microsoft Windows and Java, for example, support Unicode as 2-byte characters, i.e. UTF-16 without surrogate pairs (UCS-2).

With this encoding you must also bear in mind how the processor arranges the bytes: With Big Endian the most significant byte is located at the lowest memory address, while with Little Endian the least significant byte is at the lowest memory address:

Little Endian is used, for example, by all Intel systems, Big Endian, for example, on SPARC, by TCP/IP or the Java Virtual Machine.

UTF-16 doubles the text length for all common non-East-Asian languages.

Example

Unicode character	Big Endian	Little Endian
U+ 20AC (euro symbol)	00100000 10101100	10101100 00100000

3.4 UTF-32

Every character of the Unicode standard is encoded directly as a 32-bit unit. This also quadruples the memory requirement for European languages. With this encoding too, the processor attribute Big Endian or Little Endian must be taken into account, see also [section "UTF-16"](#).

Because of its high memory requirements, UTF-32 does not yet play a very prominent role.

3.5 Normalization

The encoding of a character in Unicode is not unique, i.e. there could be more than one encoding for a character.

A typical example is the German umlauts. For the “Ä” there are the following Unicode encodings: ‘u+00C4’ and the combination of “A” (‘u+0041’) and the umlaut symbol (‘u+0308’).

This characteristic of Unicode is very much a hindrance to programming and has therefore resulted in the normalization functions DECOMPOSE and COMPOSE.

The DECOMPOSE function dismantles each “composite” character into its individual components, i.e. the base character and the diacritical marks linked to it. The sequence of the linked diacritical marks is strictly defined.

The COMPOSE function maps all code points which together produce a character into the relevant code point.

The normalization process requires a great deal of computing power. Therefore BS2000/OSD, like the SQL standard, assumes that the data is present in normalized, i.e. compressed form.

If you are not sure whether the data exists in normalized form, you should perform a normalization. In BS2000/OSD the normalization functions are offered via the component XHCS, see also the [XHCS \(BS2000/OSD\)](#) manual. If the contents of a file are to be converted into “composite” form, this can be done using the BS2000/OSD utility PERCON, see also the [PERCON V2.9A \(BS2000/OSD\)](#) manual. Normalization in BS2000/OSD is limited to the range of code points ‘u+0000’ through ‘u+2FFF’.

Example

The Unicode code point ‘u+1ED6’ corresponds to the Latin upper-case letter “O” with circumflex and tilde. This character can be produced by means of three Unicode code points: ‘u+00D4’ for “Ö” and ‘u+0303’ for tilde, or ‘u+004F’ for “O” and ‘u+302’ for circumflex and ‘u+0303’ for tilde. The Unicode code-point sequence ‘u+00D5’ for the “Ö” with tilde and ‘u+302’ for circumflex also produces this character. The only rule is that the base character must come before the diacritical marks linked to it.

This means that the result of applying the DECOMPOSE function to the Latin upper-case letter “O” with circumflex and tilde is the Unicode code-point sequence ‘u+004F’, ‘u+0302’, ‘u+0303’, while the result of the COMPOSE function is ‘u+1ED6’.

3.6 Sort sequence

In most programs, character strings are compared in binary form. The binary sort sequence of the characters depends on their encoding:

- For all ISO8859 and Unicode encodings, the following applies:
Numbers (1-9) < Latin upper-case letters (A-Z) < Latin lower-case letters (a-z)
- For EBCDIC and UTFE, the following applies:
Latin lower-case letters (a-z) < Latin upper-case letters (A-Z) < numbers (1-9).



If records contain data in both UTF-16 code and EBCDIC code, make sure that the correct algorithm is used for sorting.

The Unicode standard describes a linguistic sorting algorithm, which performs comparisons at several levels.

For this purpose, each character is assigned a collation element which describes the weight (priority) of the character within the individual comparison level. Within each level, the sequence is defined by numbers.

The so-called sort key of a string is formed by combining the numbers of each level into a string. If the value on one level of a collation element is binary zero, this element is not used to form the sort key on this level. Two strings are compared, level by level, via the sort key. The first difference determines the result of the comparison.

In BS2000/OSD, sorting as far as level 3 is supported. The meanings of the individual levels are shown in the following table.

Comparison level	Description	Example
Level 1	Base character	role < roles < rule
Level 2	Accents	role < rôle < roles
Level 3	Upper/lower case	role < Role < rôle

At the highest level (level 1), the base characters are assigned valences, without the influence of the subsequent characters and diacritical additions. At level 2, base characters with diacritical additions, e.g. accents, tilde, are distinguished. If the strings being compared are completely the same at level 1, differences at level 2 are brought to bear. If levels 1 and 2 do not produce a difference, at level 3 upper- and lower-case letters are also distinguished. Comparison is always from left to right.



The values for the collation elements (Unicode Default Collation Table), which are published on the Unicode Consortium website, may change. For this table, visit <http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt>.

In BS2000/OSD you can obtain the collation element via XHCS, see also the [XHCS \(BS2000/OSD\)](#) manual.

4 Unicode in BS2000/OSD

4.1 Basics of Unicode support in BS2000/OSD

Application scenario

With the Unicode support in BS2000/OSD, the available code sets in BS2000/OSD systems have been extended to include additional characters. The programming and runtime environment you require to allow you to add Unicode data fields to your existing applications is provided. It is assumed that the number of fields that have to be converted to Unicode or additionally inserted is small. These are primarily name and address fields.

You need only modify applications or parts of applications which also use the extended functionality.

Unicode is supported in BS2000/OSD on the basis of the existing products. Unicode-based characters are only permitted for the texts to be processed or managed, i.e. for the field or container contents, but not for field or container names. The product-specific rules for commands and object names still remain.

Concept of the Coded Character Set Name (CCSN)

To support different character sets and encodings, the concept of Coded Character Sets (CCS) exists. A CCS defines a character set and the encoding of these characters in the file. XHCS is the central source of information for the CCSs available in BS2000/OSD, see also [page 36](#). The CCS name serves to identify the various character sets and encodings.

During the conversion from a 7-bit to an 8-bit character set and afterwards, it is necessary to distinguish converted data from non-converted data.

Therefore in BS2000/OSD-BC V7.0 you can control the default value of the CCS name of a file as follows:

- In principle, an explicit specification of the CCS name always takes precedence.
- When a new file is created, the CCS name of the user entry of the public volume set which receives it is taken over as the CCS name of the file if it is not EDF03IRV. If it is EDF03IRV, the file is given - as before - the CCS name *NONE. LMS behaves in the same way with regard to new elements; these are given the CCS name of the library if no value is specified for the CCS name.
- If you are working with the system-standard character set, the behavior remains the same as before.
- If you are working with an 8-bit character set, the use of a code at file level and library-element level is clearly specified.
- If you copy, save or restore a file, the file attribute CCS name is always transported along with it.

For an overview of the CCSN default values, see the [table "Default values for CCSN" on page 61](#).

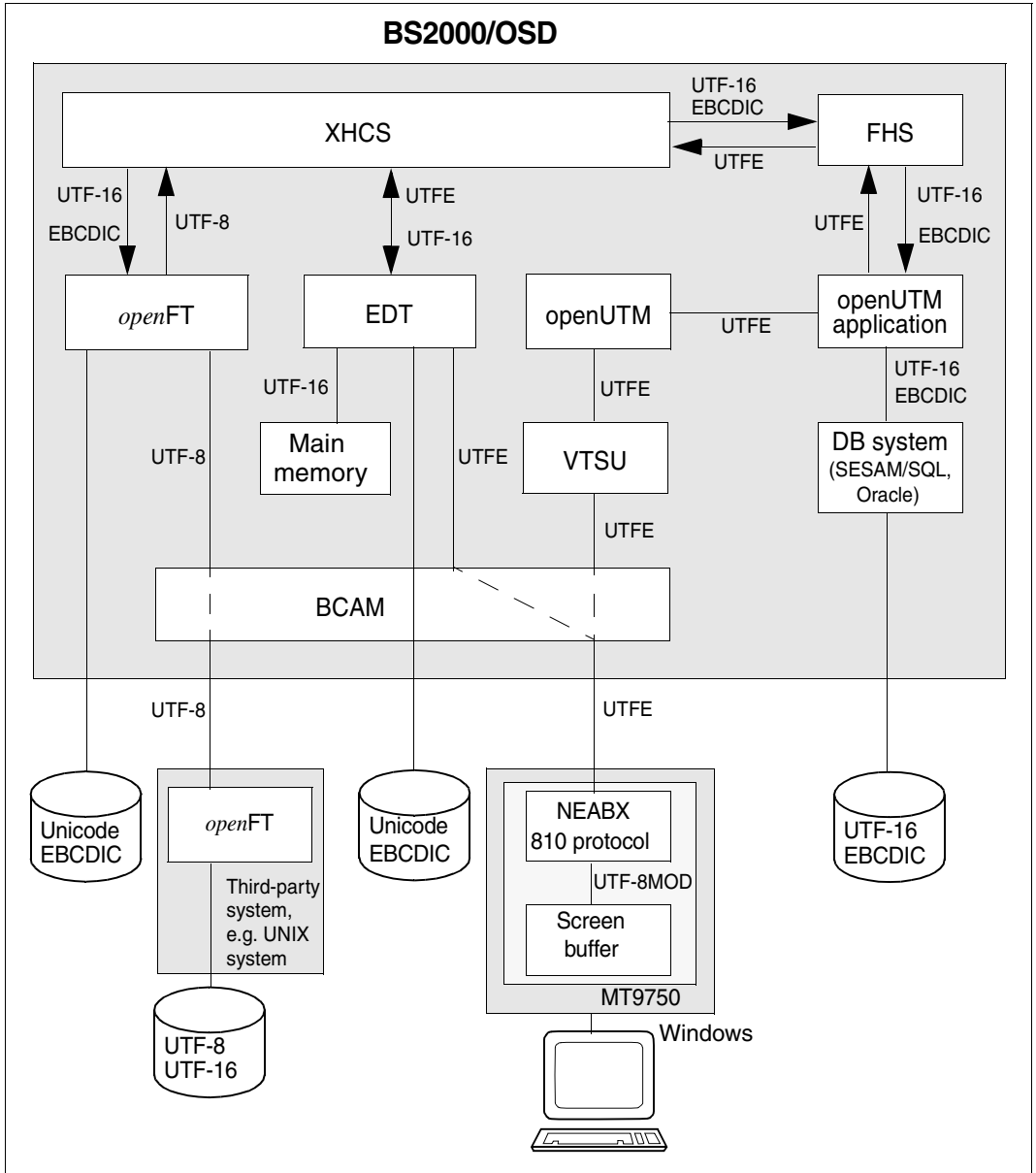
Further principles of the Unicode embedding in BS2000/OSD

In addition, the following considerations were taken as a basis for the embedding of Unicode in BS2000/OSD:

- Within applications, Unicode characters are encoded in UTF-16, at command level in UTFE.
- The Unicode character pool is limited to UCS-2, i.e. surrogate pairs (see also [section "UTF-16" on page 22](#)) are not supported.
- XHCS offers the services and the interface for encoding tasks and provides the conversions between the encodings (EBCDIC, Unicode).
- The area of use is assumed to be central Europe:
 - Conversion functions are only offered from the European language area.
 - Sorting and normalization are only available up to Unicode code point U+2FFF. As of U+3000, the Chinese, Japanese and Korean characters begin.
 - Only glyphs for European languages are available.
- The size of all BS2000/OSD and database containers remains the same. Because a UTF-16 character occupies two bytes of memory, in a SAM record, for example, depending on the file organization, a maximum of around 16,000 characters can be stored and not around 32,000, as before.
- Private disks are not supported, as they do not allow storage of the CCSN.

4.2 Overview of the affected interfaces

The following diagram shows Unicode in the BS2000/OSD system environment in the form of the Unicode encodings on the individual interfaces. For the sake of greater clarity, some of the technical details have been omitted.



Key:

EBCDIC	Any EBCDIC encoding, e.g. EDF03IRV, EDF041 etc.
Unicode	Possible Unicode encodings UTF-8, UTFE, UTF-16

The Unicode encodings on the different interfaces in BS2000/OSD are a result of the different requirements: UTF-8 is the preferred Unicode encoding in networks. Therefore the file-transfer program *openFT* uses UTF-8 encoding to transport data in heterogeneous environments. This means that *openFT* converts the data to be sent into UTF-8 and transfers it in non-transparent mode to the *openFT* partner in the destination system, which in turn converts the data into the desired target format. For conversion in BS2000/OSD, *openFT* uses the system component eXtended Host Code Support (XHCS).

UTFE corresponds to EDF03IRV in its single-byte encoding. In other words, parsers whose syntax elements do not use any characters outside the EDF03IRV character pool, can also process a UTFE string. This property is utilized at the BS2000 command level and, for example, in the EDT file editor when parsing the EDT statement.

The following sequence can be processed correctly in BS2000/OSD:

```

/MODIFY-TERMIMAL-OPTIONS CODED-CHARACTER-SET=UTFE           (1)
...
/START-EDTU                                                  (2)
@COPY F=UNICODE-FILE                                       (3)
@ON&CA 'Dolina Kukul' TO 'Долина Кукол'                    (4)
...

```

Explanation:

- (1) switches the input to UTFE.
- (2) Calls EDT. The UTFE encoding does not differ from EDF03IRV, i.e. the BS2000 command processor can interpret the command correctly.
- (3) Opens the file UNICODE-FILE and reads it into the main memory. File names which do not comply with the BS2000 file-naming conventions are rejected. While the data is read from the file into the main memory, it is converted to UTF-16 and the CCSN of the file is evaluated. For internal processing of Unicode data, EDT uses the Unicode encoding UTF-16, because this encoding is a fixed-length encoding in the range supported by BS2000/OSD (UCS-2), which assigns exactly 2 bytes to each character. This is more suitable for processing than the variable-length Unicode encoding UTFE.
- (4) The EDT command interpreter also recognizes the replace command. To execute the command, however, both strings 'Dolina Kukul' and 'Долина Кукол' are converted from UTFE to UTF-16. The data is then processed in UTF-16.

For the VTSU, UTFE is only another 8-bit EBCDIC character set. All VTSU control characters have the same encoding as in EDF03IRV. With the terminal setting UTFE, on the RDATA interface, only the characters from a-z are converted to A-Z, i.e. from lower to upper case, by default.

With version 8.3A, FHS lets you display both UTF-16 and EBCDIC and numeric fields in a mask. The COBOL application fills out the fields according to the addressing aids created by IFG. The entire mask with field contents is converted by FHS to UTFE and put in the output buffer. openUTM sends this buffer via VTSU to the emulation.

In the transport layer on the client side, the incoming data stream is converted from UTFE to UTF-8MOD. This conversion is equivalent to the conversion of EBCDIC to ISO8859 in EBCDIC messages. The emulation MT9750 transfers this data stream to its screen buffer and converts it into UTF-16 (Little Endian) for further processing.

In COBOL2000 version 1.4 and higher, the processing of Unicode data is possible with the help of the class 'national'. A national character in COBOL has the encoding UTF-16. Data fields with the class 'national' can be stored in a database table in columns with the data type NATIONAL CHARACTER (NCHAR) or NATIONAL CHARACTER VARYING (NVCHAR).

For further details, see sections [“Representing and processing Unicode characters with COBOL” on page 37](#) and [“Storing, searching for and managing Unicode data in databases” on page 39](#).

Data interchange between different systems

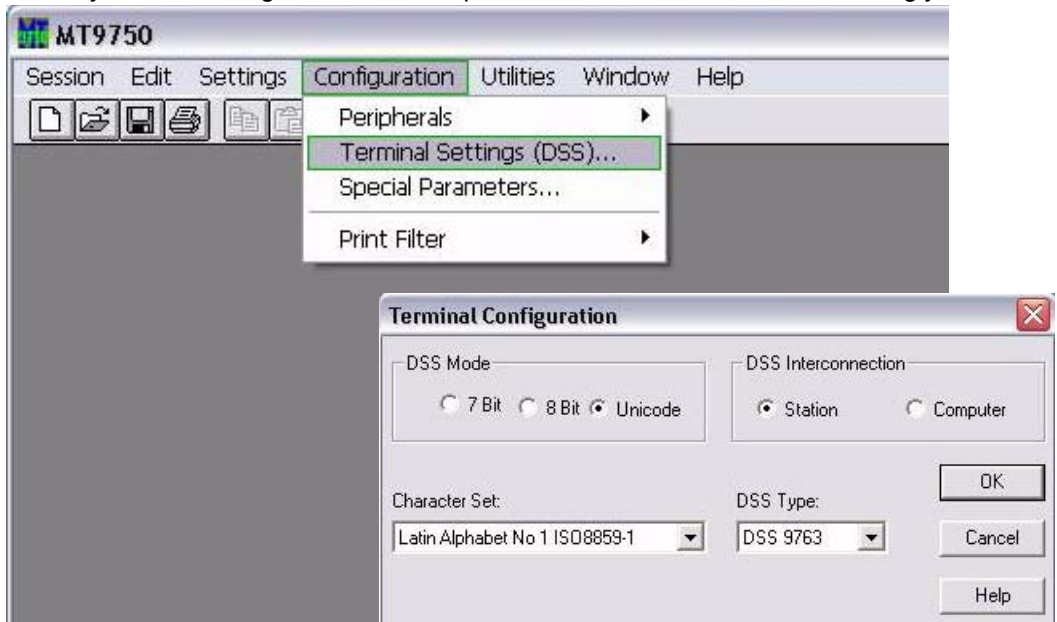
If UTF-16 character strings are sent from an application on a system A to a second application on a system B, e.g. via the UPIC interface, no conversion is necessary if the UTF-16 encoding is the same on both sides (Little Endian or Big Endian). Otherwise, the two bytes per UTF-16 character must be exchanged on one side.



The CPI-C calls *Convert_Incoming* and *Convert_Outgoing* of UPIC only provide conversion from EBCDIC to ASCII or vice versa. Therefore they cannot be used.

4.3 Configuring the terminal emulation MT9750

The terminal emulation MT9750 V7.0 allows you to send Unicode data to BS2000/OSD. To do this you must configure the session parameters of the emulation accordingly.



- ▶ On the “Configuration” menu, select “Terminal Settings (DSS)...”. The “Terminal Configuration” dialog box opens.
- ▶ For the terminal mode (“DSS Mode”) select the option “Unicode”.
- ▶ For the terminal type (“DSS Type”) select the entry “DSS9763”.

When a connection is established between the emulation and the BS2000/OSD system, the VTSU is notified that the terminal supports not only the EBCDIC character sets EDF03IRV, EDF041, EDF042 etc. but also the character set UTFE (Unicode). This is necessary to make the BS2000 command `MODIFY-TERMINAL-OPTIONS CCS = UTFE` possible. It is not possible to specify UTF-16 here because the BS2000 command processor cannot interpret UTF-16.

The BS2000 command `SHOW-TERMINAL-ATTRIBUTES *CAPABILITIES` outputs the information received by the VTSU from the emulation.

The numbers after the string `CHARACTER-SET-x =` are the so-called ISO numbers. They describe which ISO8859-n character sets the data display terminal supports. For UTFE the pseudo-ISO number 240 was chosen.

*Example: Output of /SHOW-TERMINAL-ATTRIBUTES *CAPABILITIES*

COLOUR-SUPPORT	= 8	HARDWARE-INFOLINE	= YES
LINE-MODE	= YES	EXTENDED-LINE-MODE	= YES
PHYSICAL-MODE	= YES	FORM-MODE	= YES
PROTOCOL-TYPE	= 810	EXTEND-FIELD-ATTRIB	= YES
STATUS-REQUEST	= YES	ENCRYPTION-SUPPORT	= NO
DOORS-SUPPORT	= NO	DESK2000-SUPPORT	= NO
NUMBER-OF-8-BIT-CHARACTER-SET-SUPPORTED	= 7		
CHARACTER-SET-1	= 1	CHARACTER-SET-2	= 2
CHARACTER-SET-3	= 5	CHARACTER-SET-4	= 7
CHARACTER-SET-5	= 9	CHARACTER-SET-6	= 15
CHARACTER-SET-7	= 240	CHARACTER-SET-8	= NO
CHARACTER-SET-9	= NO	CHARACTER-SET-10	= NO
CHARACTER-SET-11	= NO	CHARACTER-SET-12	= NO
CHARACTER-SET-13	= NO	CHARACTER-SET-14	= NO
CHARACTER-SET-15	= NO	CHARACTER-SET-16	= NO

In this example the terminal supports the character sets EDF041, EDF042, EDF045, EDF047, EDF049, EDF04F and UTFE.

If an application wants to send/receive Unicode data to/from the emulation, the VTSU must be notified. This can be done on the one hand via the BS2000 command MODIFY-TERMINAL-OPTIONS (see above) or, on the other, in the application via the VTSU-B control block.

The encoding of all data that is sent or received with a call must be present in UTFE.

The setting in the VTSU-B control block takes precedence over the value set with MODIFY-TERMINAL-OPTIONS. However, it only applies for the current call. The next call can be made with a different valid character set. But changing the character set results in the old screen content being deleted.

UTFE, like UTF-8, is a variable encoding of the Unicode code points. This means that, depending on the code point, between one and a maximum of five bytes are required for encoding. This property makes programming difficult. It is therefore normal to work with UCS-2 within programs. UCS-2 is the non-variable part of UTF-16, i.e. each character requires exactly two bytes, see also [section "What is Unicode?" on page 9](#). This is provided that the character pool of the BMP is sufficient.

The necessary conversion functions are provided by the system component XHCS, see also [section "Character handling in BS2000/OSD \(XHCS\)" on page 36](#).

5 Unicode adjustments in BS2000 applications

This chapter describes the Unicode adjustments in the BS2000 software products for program development and testing, data storage, printing and Web integration, which the customer needs in order to adapt their BS2000 applications to Unicode:

- XHCS, the software for handling characters in BS2000/OSD
- COBOL language for displaying and processing Unicode characters
- AID, the Advanced Interactive Debugger
- Databases SESAM/SQL and Oracle 10g for BS2000/OSD
- IFG and FHS for generating and handling formats
- RSO for controlling the output of print jobs
- *WebTransactions* for Web integration of Unicode-capable applications

For more detailed information on implementing the Unicode support, see the relevant product-specific manuals and the chapter [“Related publications” on page 97](#).

5.1 Character handling in BS2000/OSD (XHCS)

In BS2000/OSD, **EX**tended **H**ost **C**ode **S**upport (XHCS) manages the character sets, provides the services and interface for encoding tasks, and provides the conversions between the encodings (EBCDIC, Unicode).

The various software programs do not need to store information on character sets permanently, as the XHCS interfaces are available to any user program.

The following list outlines the key functions of XHCS in connection with Unicode support. XHCS

- Converts from and to UTF-16, UTF-8 and UTFE as well as from and to ISO and EBCDIC and maps the case functions (TOUPPER, TOWER).
- Outputs information on code compatibility.
- Normalizes input strings.
- Provides the product SORT for sorting the required sort tables for the European languages, see also [section “Sort sequence” on page 24](#).
- Supports the definition of additional Unicode characters. You can also obtain collation elements in BS2000/OSD via XHCS.

For further details of XHCS, see the [XHCS \(BS2000/OSD\)](#) manual.

5.2 Representing and processing Unicode characters with COBOL

Basic paradigms

- COBOL programs are still written in EBCDIC.
- Application data is still largely encoded in EBCDIC. The Unicode character set is only used where necessary.
- The individual application must program the use of Unicode data.
- Restrictions remain the same in terms of bytes.

Implementation of UTF-16 support

COBOL2000 V1.4 supports UTF-16 by implementing the key language elements for the data type 'national' from the COBOL standard ISO1989:2002.

The support includes:

- PICTURE picture characters N and USAGE NATIONAL as well as GROUP-USAGE NATIONAL
- National literals and national figurative constants
- Implicit conversion of EBCDIC to UTF-16 in transfers (MOVE) and in conditions
- Explicit conversion with the functions NATIONAL-OF and DISPLAY-OF
- Substitute characters and exception handling during conversions
- Extension of the functionality of existing language elements to support national data (in particular conditions, INITIALIZE, INSPECT, STRING, UNSTRING, SORT, MERGE and the functions NUMVAL, LOWER-CASE, UPPER-CASE, LENGTH, BYTE-LENGTH and REVERSE).

Data of the class 'national' is largely defined and used in the same way as the EBCDIC data of the class 'alphanumeric'.

Detailed information on national data and conversion between EBCDIC and UTF-16 representation can be found in the [COBOL2000 \(BS2000/OSD\)](#) language description.

5.3 Advanced Interactive Debugger (AID)

To support Unicode, the Advanced Interactive Debugger (AID) offers:

- The data type %UTF16
This data type is equivalent to the data type 'national', which COBOL2000 offers as part of its Unicode support, see [page 37](#). With this data type, every character has a two-byte encryption.
- Conversion functions from single-byte EBCDIC to UTF-16 and vice versa: functions %UTF16() and %C().
- The option of entering strings in UTFE format (U'..').
- The option of displaying the current code settings and available character sets with %SHOW %CCSN.

For further details, see the AID manual "[Debugging of COBOL Programs](#)".

5.4 Storing, searching for and managing Unicode data in databases

The database systems SESAM/SQL V5.0 and Oracle 10g for BS2000 allow you to store, search for and manage Unicode strings.

5.4.1 Unicode concept in SESAM/SQL

The concept of Unicode support in SESAM/SQL allows the use of Unicode characters in the columns of tables and takes into account coded character sets in databases, in input/output files and for user programs.

The SESAM/SQL database contains in the catalog the name of a coded character set, which specifies how character data in the database is interpreted.

This concept affects the SQL language description, the utility functions and the SESAM/SQL user programs, as described in the following.

For basic information on Unicode support in SESAM/SQL, see the [SESAM/SQL-Server \(BS2000/OSD\) Core Manual](#).

SQL language description

New data types NATIONAL CHARACTER and NATIONAL CHARACTER VARYING

In SESAM/SQL databases you can define table columns of the data type NATIONAL CHARACTER and NATIONAL CHARACTER VARYING and save Unicode data in them in UTF-16 format. Metadata such as names of tables, columns and views, for example, is still specified in EBCDIC.

The byte lengths of the character data types are unchanged. Because a UTF-16 character occupies two bytes of memory, the following maximum numbers of characters result for Unicode:

NCHAR	128 characters
NVARCHAR	16,000 characters

Unicode data types are also supported by the database functions for recovering and modifying data. In addition, host variables and SQL literals in UTF-16 format can be used.

New function TRANSLATE

Via the SQL language, SESAM/SQL offers the conversion function TRANSLATE for converting EBCDIC data into Unicode data and vice versa (N[VAR]CHAR <-> [VAR]CHAR).

For the conversion, SESAM/SQL uses the conversion functions provided by XHCS.

Utilities

Coded character sets are taken into account in the following utility functions:

- **CREATE CATALOG**
Creates a database. During database creation a coded (EBCDIC) character set (parameter `CODE_TABLE`) can be defined.
- **ALTER CATALOG**
Allows you to change the coded character set of a database.
- **EXPORT TABLE**
Allows you to export a table from a database into an export file. The export file is created with the Coded Character Set Name (CCSN) of the database.
- **IMPORT TABLE**
Allows you to import a table from an export file into a database. The CCSN of the export file is checked against the CCSN of the database.
- **UNLOAD**
Allows you to unload data from a table into a file. In delimiter format, the file is created optionally with the CCSN of the database or with the CCSN UTFE. With other formats, the file is created with the CCSN of the database. During unloading, the data is converted if necessary.
- **LOAD**
Allows you to load data from a file into a table. In delimiter format, a file with the CCSN UTFE or a CCSN of type EBCDIC can also be processed. A file CCSN of type EBCDIC is checked against the CCSN of the database. During loading, the data is converted if necessary.

SESAM/SQL applications

SESAM/SQL applications connect to the database via the so-called Database Handler (DBH). Depending on whether the DBH is independent or linked-in, you must specify the character set differently. With the new connection-module parameter `CCSN` (Coded Character Set Name), you specify for the application the character set with which the application interprets character data.

With a linked-in DBH, this is done via the `DBH` option.

The SESAM/SQL application can work with the database if the CCSN of the database and the CCSN of the application have identical values, or if no coded character set is used for the database.

Compatibility

“Old” databases can be operated unchanged without taking into account coded character sets. However, SESAM cannot then convert from and to UTF-16. Statements which require such conversions are rejected.

Output files which contain Unicode data cannot be used as input files in older SESAM/SQL versions.

5.4.2 Unicode concept in Oracle

Oracle 10g supports a variety of different character sets, including several Unicode character sets.

A list of all character sets supported by Oracle can be found in the manual "[Oracle Database Globalization Support Guide Part Number B14225-02](#)".



The UTFE character set listed there corresponds to the UTF-EBCDIC character set defined by IBM and not the BS2000 character set UTFE.

The Oracle character sets supported in BS2000/OSD are listed in the manual "[Oracle User's Guide for Fujitsu Siemens Computers BS2000/OSD](#)".

A distinction must be made here between the character set on the application side, the character set for the database, and the "national" character set for columns with the Unicode data type. If necessary, Oracle converts between these character sets using its own tables, regardless of the character sets of the operating system.

In Oracle in BS2000/OSD, you cannot specify a Unicode character set as the character set for the database or the character set on the application side.

The concept of Unicode support in Oracle in BS2000/OSD does, however, allow the use of Unicode characters in columns with the Unicode data type.

For basic information on Unicode support in Oracle, see the manual "[Oracle Database Globalization Support Guide Part Number B14225-02](#)".

SQL functionality

In Oracle databases you can store Unicode data in table columns of certain data types.

Specifically, these data types are as follows:

NCHAR	Fixed-length text data, maximum 2000 bytes
NVARCHAR2	Variable-length text data, maximum 4000 bytes
NCLOB	Character Large Object, maximum 4 GB, stored in UTF-16

You can define the character set of NCHAR and NVARCHAR2 data with CREATE DATABASE with either UTF-8 or AL16UTF-16, the UTF-16 character set of Oracle.

You can edit the Unicode data in these columns via different interfaces, for example SQL and PL/SQL for storing, recovering and modifying data.

You can also use host variables and SQL literals in UTF-16 format.

Via SQL, Oracle offers the conversion functions TRANSLATE, TO_CHAR and TO_NCHAR, for converting EBCDIC data into Unicode data and vice versa.

There are also the functions NCHR() and UNISTR() for creating individual Unicode characters and Unicode literals.

Utilities

Export/Import

Unicode data of types NCHAR, NVARCHAR2 and NCLOB is written unchanged to a binary file during export and can thus also be imported into other databases.

*SQL*Loader*

The SQL*Loader allows you to load Unicode data from an input file into the database.

In the SQL*Loader control file you can specify via the CHARACTERSET parameter that the data in the input file should be in UTF-16 format.

Oracle applications

Oracle applications can process Unicode data in the database via the interfaces Pro*Cobol, Pro*C, OCI, JDBC and ODBC.

Compatibility

Oracle has supported Unicode data in NCHAR data types since version 9i.

In "old" 8.1.7 Oracle databases, for columns of type NCHAR, other character sets may have been defined as Unicode. In this case, after upgrading the database from 8.1.7 to 10g, you must also upgrade the NCHAR columns, as described in the manual "[Oracle Database Upgrade Guide Part Number B14238-01](#)".

5.5 Support for Unicode fields in formats

As of version 8.3A of the Format Handling System (FHS), formatted messages between user program and terminal may also contain Unicode characters. This is enabled by the so-called Unicode formats, which can be generated with the Interactive Format Generator (IFG) as of version 8.3.

A Unicode format means a format which contains at least one field that was given the attribute UNICOD during the format definition in IFG.

Fields with the attribute UNICOD have the following properties:

- The user can enter any character from the BMP in this field.
- The contents of the field are stored in the addressing aid of the user program. Each character occupies two bytes.
- The encoding of the character is UTF-16.

Terminal emulation and formats

On the user interface, the terminal emulation supports two modes:

- Either the whole format is defined in Unicode mode, i.e. the user can enter any Unicode character in any input field.
- Or the format is not defined in Unicode mode, i.e. input is limited to the set 7-bit or 8-bit (ISO8859-x) character set.

The emulation does not support a mixture of these two modes in one mask, i.e. it cannot within a mask limit the input for one field to ISO8859-1 and at the same time allow the input of Unicode characters for another field in the same mask.

Checks which have previously been performed in mode 2 by the terminal emulation must be relocated to the application. FHS takes over these checks for the so-called # formats.

Processing # formats in FHS

With Unicode formats, FHS analyzes the user's UTFE input string, allocates the substrings to the individual fields of the format and converts them according to the field definitions.

- If the field is assigned the attribute UNICOD, FHS converts it to UTF-16 and transfers it to the point in the user program specified by the addressing aid of the format.
- If the field is not a UNICOD field, FHS checks whether all entered characters are compatible with the basic character set of the format – defined during generation in IFG – or with the character set that was defined via USER/LTERM in openUTM. The

currently valid character set is determined as with the 8-bit formats. If FHS does not detect any errors, the characters are converted to EBCDIC and transferred to the point in the user program which defines the addressing aid of the format.

If a character cannot be mapped to the target character set, FHS initiates the 'field validation check failures', i.e. FHS

- either puts a return code in the addressing aid or
- outputs a standard error message.

For further information, see the [FHS \(BS2000/OSD\)](#) manual.

Defining individual UNICODE fields with IFG

In IFG the attribute UNICODE can be assigned in the display properties of the fields (mask 0305), provided the relevant field is a pure text field. In the display properties of the format (mask 030A), the option "Requires UNICODE support" is set to "YES" as soon as the format contains a UNICODE field.

In IFG it is not possible to enter Unicode characters. Therefore, it is also not possible to define any constant texts or selection fields in IFG which contain Unicode characters.

Addressing aids can be generated for Unicode formats for the languages COBOL and Assembler using IFG.

For further information, see the [IFG \(BS2000/OSD\)](#) manual.

5.6 Outputting print jobs with Unicode data

For the output of Unicode characters, APF-IPDS printers and UTF-8-capable RSO printers are supported.

5.6.1 Central printers (AFP-IPDS)

The print file contains the net data stream. A second file, the page definition, defines how this net data is interpreted and printed. It is therefore possible to switch from UNICODE and EBCDIC fields.

If a text file or certain lines or parts of lines are encoded in Unicode, you must create or appropriately adjust a page definition and specify the Unicode character set in the print command.

5.6.2 Decentralized printers (RSO)

Remote Spool Output (RSO) controls the output of print jobs to decentralized printers.

The EBCDIC-based mechanism of RSO can also process UTFE data, because the single-byte range of the UTFE encoding corresponds to the character set EBCDIC.DF.03IRV (EDF03IRV), see also [section “UTF-EBCDIC \(UTFE\)” on page 17](#).

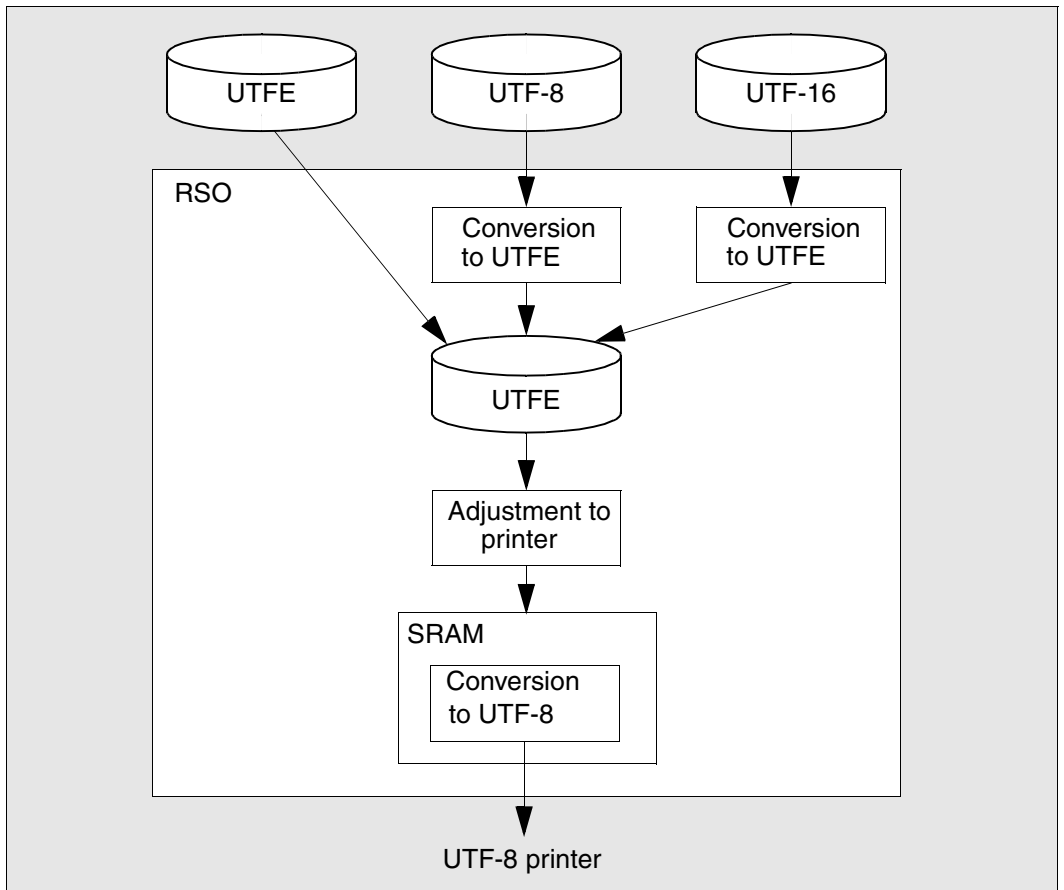


Figure 1: Processing of Unicode files

With LAN-to-host printing, i.e. with output of files from a PC or Unix-based systems to RSO printers, UTF-8 files must be converted to UTFE before they can be processed. Likewise, UTF-16 data must be converted to UTFE before processing. XHCS provides the basics for conversion from one encoding to another, see also the [XHCS \(BS2000/OSD\)](#) manual.

If a printer supports ASCII and UTF-8, e.g. the Unicode-capable UTF-8 printer PRINT-RONIX P7000, RSO can work in both ASCII mode (via EBCDIC) and UTF-8 mode (via UTFE), depending on the encoding of the incoming file. Because RSO evaluates the Coded Character Set name of the file (CCS-NAME), only one character set per file is possible.

For further information, see the “[RSO \(BS2000/OSD\)](#)” manual.

5.7 Web integration of Unicode-capable applications (*WebTransactions*)

As part of the Unicode support, *WebTransactions* generates UTF-8-encoded data. This data is forwarded to the browser and back again in order to support Unicode-capable host applications.

The templates inform the browser which character encoding it is to use to display the data and to send the response

- either via the attribute `charset` in the HTTP header `Content-Type` or
- via the HTML tag
`<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />`.

BizTransactions does not support Unicode.

WebTransactions for OSD

As part of the Unicode support from BS2000/OSD, the terminal protocol 9750 supported by *WebTransactions* for OSD becomes Unicode-capable.

The Unicode support must be explicitly activated, otherwise *WebTransactions* for OSD behaves compatibly with the previous versions.

For this purpose a new terminal type was introduced:

The system-object attribute `TERMINAL_TYPE` now also recognizes the value `9763-UNICODE`.

If you set this value, the emulation integrated in the host adapter reveals itself to be Unicode-capable during connection establishment to BS2000/OSD. The relevant BS2000/OSD application then has the option of using this capability.

When the host application switches to Unicode mode,

- the field contents from/to BS2000/OSD are interpreted as UTF-EBCDIC,
- the corresponding contents of the host objects are output as UTF-8-encoded during evaluation, and interpreted as UTF-8-encoded during assignment.

The host adapter indicates this in the system-object attribute `HOST_CHARSET` with the new value `UTF-8`.

Because all templates generated by *WebTransactions* for this host adapter assign the system-object attribute `HOST_CHARSET` to the global system-object attribute `CHARSET`, the content type in the HTTP header is appropriately set such that the browser can interpret the data correctly.

For further information, see the *WebTransactions* manual [“Connecting to OSD Applications”](#).

WebTransactions for openUTM

As part of the Unicode support from BS2000/OSD, the host adapter used by *WebTransactions* for openUTM can process data on the UPIC interface as well as Unicode characters.

The Interactive Format Generator (IFG) used to formulate the screen masks enables individual fields to be assigned the UNICODE attribute (see also [page 43](#)). With the OSD program IFG2FLD, the format descriptions are exported into a format which can be read by *WebLab* – the format-description source.

IFG2FLD calculates the offsets of the individual fields in the UPIC buffer depending on the UNICODE attribute. This format-description source is converted by the integrated generator in *WebLab* into format-specific FLD files, taking into account the Unicode attribute. The FLD files must be newly generated for all formats containing Unicode fields. Previously generated templates can remain unchanged if the switch to Unicode fields was the only change in the format. You may have to insert the assignment of the value UTF-8 to the system-object attribute CHARSET at a central point or in these format-specific templates.

This new Unicode attribute is also available to the *WebTransactions* applications as a host-object attribute: It can take the values 'Y' and 'N'.

- For all fields flagged with Unicode=='Y', the host data in the UPIC buffer is regarded as UTF-16 data and converted into UTF-8 data for the browser, or the UTF-8 browser data is converted into UTF-16 host data.
- The ASCII-EBCDIC conversion, which is dependent on the system-object attribute HOST_CHAR_CODE, is only performed if Unicode=='N'.

In addition, the attribute of the same name on the host control object WT_HOST_MESSAGE.Unicode indicates whether the active message contains at least one field with Unicode=='Y'.

Depending on this attribute, the template can set the global system-object attribute CHARSET to the value UTF-8. The templates generated by *WebTransactions* automatically contain this assignment. With existing templates which are not to be regenerated, you may have to insert this manually.

For further information, see the *WebTransactions* manual "[Connecting to openUTM Applications via UPIC](#)".

6 Unicode adjustments for file processing

This chapter explains the Unicode support in the following BS2000/OSD file-processing programs:

- EDT for editing Unicode files
- PERCON for conversion and normalization
- SORT for sorting fields
- *openFT* for file transfer

Detailed information on implementing the Unicode support can be found in the respective product-specific manuals, see also [“Related publications” on page 97](#).

6.1 Creating and editing Unicode files (EDT)

The file editor (EDT) allows you to create and edit Unicode files in the character sets UTF-8, UTF-16 or UTFE.

EDT V17.0 can be operated in a V16.6-compatible 'compatibility' mode and a Unicode mode.

- In Unicode mode, EDT V17.0A can edit Unicode-encoded files.

Users who want to edit Unicode-encoded files are offered easy-to-use support. This includes the option of editing differently encoded files in different EDT work files simultaneously, and removing the limit on the record length (previously 256 characters). EDT can process records of up to 32,768 bytes when writing to a file.

The consequence of the Unicode representation in the work files is that all interfaces on which the user has direct access to the records of the work file cannot remain compatible. This applies to the L-mode subprogram interface, the @RUN interface, and the locate mode of the IEDTGLE interface. These interfaces can therefore no longer be used if you want to use the new functions.

- The compatibility mode offers the full functionality of EDT V16.6B, including the old L-mode subprogram interface. A new @MODE statement in compatibility mode allows you to switch to Unicode mode.

For further information, see the [EDT \(BS2000/OSD\)](#) manual.

6.2 Converting and normalizing Unicode files (PERCON)

The Peripheral Converter (PERCON) allows you to convert SAM files or parts of them which are not Unicode-encoded into Unicode format and vice versa. Conversion occurs whenever the input and output files have been assigned different Coded Character Set (CCS) names.

Because Unicode data can be present in non-normalized form, PERCON additionally offers the option of converting this data into the composite character representation, see also [section “Normalization” on page 23](#).

For further information, see the [PERCON V2.9A \(BS2000/OSD\)](#) manual.

6.3 Sorting Unicode fields (SORT)

SORT allows you to sort fields which contain Unicode characters according to the Unicode Default Collation Table. This table contains the values for the collation elements, see also the website of the Unicode Consortium at <http://www.unicode.org/Public/UCA/4.0/allkeys-4.0.0.txt>.

Sorting is currently possible for the Unicode character set UTF-16, in which each character is represented by two bytes (without surrogate pairs).

Every UTF-16 character is assigned a so-called collation element, which defines the sequence in which the UTF-16 characters are sorted, see also [section “Sort sequence” on page 24](#). The collation elements themselves are defined by means of a table supplied by XHCS, see also the [XHCS \(BS2000/OSD\)](#) manual. This table contains a weight of the character at different levels. For more information on the different levels, see the [section “Sort sequence” on page 24](#).

For further information on how SORT works, see the [SORT V7.9A \(BS2000/OSD\)](#) manual.

6.4 Transferring Unicode files (*openFT*)

If you transfer files with *openFT* partners as of V10, you can assign the coded character sets to be used locally and remotely for the data conversion in the request. Unicode files can also be transferred with these partner systems.

You define the coded character sets to be used either by specifying a parameter in the TRANSFER-FILE command or with DMS files in the file catalog.

During file transfer within BS2000/OSD, XHCS is used for the code conversion.

For file transfer to other systems, the standard code tables for the conversion are implemented in *openFT* and can be set via operating parameters in *openFT*.

For further information on file transfer with *openFT*, see the manuals on [openFT V10.0A for BS2000/OSD](#).

7 Tips and tricks

7.1 Tips on SESAM/SQL

If you are editing Unicode files with SESAM/SQL, you should bear in mind the following:

CCSN of the database / CCSN of the application

Using the statement `ALTER CATALOG ALTER CODE_TABLE`, you can alter the CCSN (code table) of the database. Be aware that an application can only access a database correctly via the DBH if the CCSN parameter in the configuration file of the application matches the CCSN of the database. In the case of an application with linked-in-DBH, the parameter `CODED-CHARACTER-SET` of the DBH option `LINKED-IN-ATTRIBUTES` must be identical with the CCSN of the database.

If the values are not identical, access to the database is only possible if the `CCSN _NONE_` is defined for the database.

Furthermore, it is always possible to access the metadata of the database by means of the Utility Monitor via the SNF and INF masks. In the SNF.1 mask, the CCSN currently defined for the database is output. In the CNF mask, the CCSN entered in the configuration file of the application (or DBH option file with linked-in DBH) is output. You can use these two masks to check whether the CCSNs match.

CCSN of the terminal

In the SQL mask of the Utility Monitor, records returned by a `SELECT` statement are output.

Alphanumeric values of the data type `CHARACTER (VARYING)` are taken over as they stand in the database, while numeric and time-data types are converted to `CHARACTER`. `NATIONAL` values of the data type `N[VAR]CHAR` are converted to `CHARACTER` according to the CCSN of the database. For non-convertible values, the substitute character “period (.)” is output.

The screen shows the characters in the CCSN that is set for the terminal. The setting can be checked with the BS2000 command `SHOW-TERMINAL-OPTIONS` and, if necessary, modified with the `MODIFY-TERMINAL-OPTIONS` command.

Caution: This command is only permitted as long as no program is loaded.

If the CCSN set for the terminal differs from the CCSN of the database, some of the characters will not be displayed correctly on the screen.

Unloading in delimiter format with UTFE

During unloading in delimiter format with UTFE, the values of the columns are converted to UTFE and output to the unload file. Bear in mind that the encoding in UTFE can be between one and five bytes long.

7.2 Tips on LMS

If you are using LMS and Unicode files, you should bear in mind the following:

EXTRACT-ELEMENT in ISAM file

LMS extracts library elements by default as ISAM files with an 8-byte EBCDIC ISAM key. This can be interpreted by EDT as a line number. If the element in question has an ISO (ASCII) character set, LMS generates a file which contains both EBCDIC and ASCII characters. It may then not be able to edit this file satisfactorily with other products, e.g. EDT. UTF16 is even more problematic, as then the ISAM key consists of single-byte characters and the data of two-byte characters.

You can, however, extract the elements by specifying `ACCESS-METHOD = *SAM` or add an ISAM file created by EDT to a library with `//ADD-ELEMENT`. By default the ISAM key is then removed with a warning (also with a 16-byte key).

You can, however, also have the key saved in the element with `SOURCE-ATTRIBUTES = *KEEP` and later extract the element unchanged with the key and edit it further with EDT V17.

EDIT-ELEMENT

With the LMS statement EDIT-ELEMENT you can edit elements with EDT. LMS uses the EDT subprogram interface in compatibility mode (EDT-UP V16) for this. Elements in a Unicode character set cannot be edited with EDIT-ELEMENT, as the EDT-UP V16 interface cannot handle Unicode. LMS passes the statement @CODENAME UTF8/16/E to the EDT-UP interface. This is rejected by EDT V17 with a return code.

You can, however, edit the element from within EDT with @OPEN LIBRARY=... .

ADD-ELEMENT and EXTRACT-ELEMENT from the EDT statement line (@USE)

The EDT statement @USE allows you to execute certain LMS statements in the EDT statement line. These include the statements ADD-ELEMENT and EXTRACT-ELEMENT.

- ADD-ELEMENT (read from EDT work file)

You have created Unicode data in an EDT work file and now you want to write this data from the EDT statement line into a library element with ADD-ELEMENT. EDT passes the ADD-ELEMENT statement to LMS, and LMS now attempts to read the data using the EDT-UP function IEDTGET. EDT rejects this operation with a return code because LMS wants to read the data via the EDT-UP V16 interface.

- EXTRACT-ELEMENT (write to EDT work file)

You want to write a Unicode element to an EDT work file from within the EDT statement line using the LMS statement EXTRACT-ELEMENT. EDT rejects this operation as with EDIT-ELEMENT, because here too LMS attempts to pass a @CODENAME UTF8/16/E statement to EDT via the EDT-UP V16 interface.

Unicode elements can, however, be written to or read from a library with the EDT functions @WRITE LIBRARY=... and @COPY LIBRARY=... .

8 Appendix

8.1 Unicode products in BS2000/OSD: overview and dependencies

BS2000/OSD-BC as of version 6.0B is required for all Unicode products.

Product	Version	Requires (in addition to BS2000/OSD-BC V6.0B or higher)
BS2000/OSD-BC	6.0B (2nd correction package 2006) contains – CRTE-BASYS 1.6 – CRTE-MSG 1.6 – SPOOL 4.8 – SPSERVE 2.9 – SYSFILE 15.0B	–
	7.0 contains – CRTE-BASYS 1.6 – CRTE-MSG 1.6 – SPOOL 4.8 – SPSERVE 2.9 – SYSFILE 16.0	–
AID	3.2	<i>openNet Server 3.2, MT9750 7.0</i>
COBOL2000	1.4	XHCS 2.0, CRTE 2.6
CRTE	2.6	–
CRTE-BASYS	1.6	–
CRTE-MSG	1.6	–
EDT	17.0	<i>openNet Server 3.2</i>

Table 5: Unicode products: overview and dependencies

Product	Version	Requires (in addition to BS2000/OSD-BC V6.0B or higher)
ESQL-COBOL (COBOL SQL-Precompiler)	3.0	SESAM 5.0, COBOL2000 1.4
FHS	8.3	<i>openNet</i> Server 3.2, MT9750 7.0, IFG 8.3
IFG	8.3	FHS 8.3, COBOL2000 1.4
MT9750	7.0	<i>openNet</i> Server 3.2
<i>openFT</i>	10.0	–
<i>openNet</i> Server	3.2 contains – XHCS 2.0 – VTSU-B 13.2	–
ORACLE	10g	–
PERCON	2.9	XHCS 2.0
RSO	3.5	XHCS 2.0
SESAM/SQL-Server	5.0	COBOL2000 1.4, ESQL-COBOL 3.0, XHCS 2.0
SORT	7.9	XHCS 2.0
SYSFILE	15.0B for OSD 6.0B	–
	16.0 for OSD 7.0	–
VTSU-B	13.2	MT9750 7.0
<i>WebTransactions</i>	7.1	<i>openNet</i> Server 3.2
XHCS	2.0	–

Table 5: Unicode products: overview and dependencies

8.2 Extended BS2000 macros

GCCSN – display CCS name for command and file input

The STREAM parameter for outputting the Coded Character Set name for the BS2000 system files is new.

GCCSN
STREAM = SYSDTA/SYSCMD/SYSOUT/SYSLST/SYSLST(1)..(99) , ...

STREAM=

Name of the system file for which the CCS name is to be output.

Note

If SYSOUT/SYSLST was allocated to a file or a library element, the CCS name cannot change between the opening and closing of the file or library element.

If, however, SYSOUT was allocated to a terminal, you can temporarily change the CCS name dynamically in your program via the VTSUC-B for the current WROUT request.

This change only applies for the current request and cannot therefore be determined via the GCCSN macro.

WROUT – pass record to SYSOUT

The ASSIGN parameter for specifying whether changes to the SYSOUT allocation are to be displayed is new.

WROUT
record,error, , ... ,[ASSIGN = NO / YES]

ASSIGN=

Defines whether changes to the SYSOUT allocation are to be displayed. The user program is notified of the initial default allocation and of every subsequent allocation for SYSOUT via the error address.

The write operation does not occur if a change is detected.

This operand is only permitted for a 31-bit interface.

NO

Changes to the SYSOUT allocation are not displayed.

YES

Changes to the SYSOUT allocation are displayed.

Note

The SYSOUT allocation is displayed in an output field of the parameter list and is supplied from the SYSFIL processing.

The user program thus has the option of reacting to the changed basic conditions and is able to perform any necessary conversions of the output string in order to then re-initiate the write operation with the corrected output string.

TSTAT – query properties of data terminal /**DCSTA – generate operand table for data-terminal properties**

With the TSTAT macro you request information on the terminal in timesharing mode. DCSTA generates receiving fields or symbolic field names (DSECTs) for the information you receive by calling TSTAT.

In the BASIC section of the DSECT DCSTA in the STACSSx field, the value x'F0' is output for Unicode-capable terminal types.

VTSUCB – create VTSU parameters for input/output

If Unicode fields are to be output on the terminal, they must be converted to UTFE.

The CCS name in VTSUCB must then be supplied with UTFE. Other Unicode CCS names (UTF16 or UTF8) are not allowed.

8.3 Coded Character Set Names (CCSN): default values

Variable/Command/Program	Default value	Comments
CLASS-2 option HOSTCODE	EDF03IRV	
ADD-USER	From CLASS-2 option HOSTCODE	
MODIFY-USER-ATTR CCS=*STD	From CLASS-2 option HOSTCODE	
VTSU	TIAM/UTM/DCAM-PTERM8=N: – EDF03IRV TIAM/UTM/DCAM-PTERM8=Y: – 8-bit terminal: CCSN of HOME-PUB-SET – 7-bit terminal: CCSN=EDF03IRV	The parameters TIAM-PTERM8, UTM-PTERM8 and DCAM-PTERM8 are found in the VTSU parameter file under TSOS. They are valid throughout the system.
MODIFY-TERMINAL- OPTIONS CCS=*8-BIT-DEFAULT	8-bit terminal: – CCSN= CCSN of HOME-PUB-SET 7-bit terminal: – Error message	You can also specify a CCSN explicitly, but this must be supported by the terminal.
EDT	After loading of EDT – in DIALOG mode the CCSN matches the CCSN set in the VTSU – in BATCH mode the CCSN matches the CCSN of the allocated statement input file	EDT in Unicode mode: Each window can have data with a separate CCSN. If data is copied from one window to another, the data is converted. EDT in compatibility mode: In interactive mode, when a file is imported, the CCSN can change to that of the file, as long as there is no data in either of the EDT windows. This means that all EDT windows can only contain data with the same CCSN.

Table 6: Default values for CCSN

Variable/Command/Program	Default value	Comments
BS2000/OSD: Creating a file: CREATE-FILE CREATE-FILE-GROUP MODIFY-FILE-ATTRIBUTES MODIFY-FILE-GROUP- ATTRIBUTES	CCSN of the receiving Public Volume Set (PVS) = EDF03IRV -> CCSN of the file = *NONE CCSN of the receiving PVS != EDF03IRV -> The file receives the CCSN of the user entry of the PVS.	
LMS EDIT-ELEMENT ELEMENT=*NONE TO-ELEMENT=<output>	EDT determines the CCSN, see EDT.	
PLAM	CCS=*NONE CCS= CCS of the PLAM library	

Table 6: Default values for CCSN

Action	Command	Default value	Comment
Defining the system default character set	CLASS-2 option HOSTCODE=<ccsn>	EDF03IRV	Evaluation occurs with ADD- USER command, with XHCS CCSN=*SYSDEF
Setting up a user ID	ADD-USER	From CLASS-2 option HOSTCODE	Defines the CCS name in the user entry of the HOME PVS. Evaluation: XHCS CCSN=*USRDEF VTSU macro DCSTA STACURCH
Setting up a user entry for a user ID	ADD-USER	From CLASS-2 option HOSTCODE	Defines the CCS name in the user entry of the PVS.

Table 7: Actions and associated default values for CCSN

8.4 Useful code tables

8.4.1 Conversion from ISO8859 to EBCDIC (BS2000/OSD) and vice versa

ISO8859-n and EBCDIC.DF.04.n have the same pool of characters. Conversion from one character set to the other is based on the following conversion table:

Conversion from ISO8859 to EBCDIC (BS2000/OSD)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
1-	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2-	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4-	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5-	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	BB	BC	BD	6A	6D
6-	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7-	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	FB	4F	FD	FF	07
8-	20	21	22	23	24	04	06	08	28	29	2A	2B	2C	09	0A	14
9-	30	31	25	33	34	35	36	17	38	39	3A	3B	1A	1B	3E	5F
A-	41	AA	B0	B1	9F	B2	D0	B5	79	B4	9A	8A	BA	CA	AF	A1
B-	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
C-	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
D-	AC	69	ED	EE	EB	EF	EC	BF	80	E0	FE	DD	FC	AD	AE	59
E-	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
F-	8C	49	CD	CE	CB	CF	CC	E1	70	C0	DE	DB	DC	8D	8E	DF

Table 8: Conversion from ISO8859 to EBCDIC (BS2000/OSD)

Conversion from EBCDIC (BS2000) to ISO8859

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	85	09	86	7F	87	8D	8E	0B	0C	0D	0E	0F
1-	10	11	12	13	8F	0A	08	97	18	19	9C	9D	1C	1D	1E	1F
2-	80	81	82	83	84	92	17	1B	88	89	8A	8B	8C	05	06	07
3-	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4-	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	60	2E	3C	28	2B	7C
5-	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	21	24	2A	29	3B	9F
6-	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	5E	2C	25	5F	3E	3F
7-	F8	C9	CA	CB	C8	CD	CE	CF	CC	A8	3A	23	40	27	3D	22
8-	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
9-	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
A-	B5	AF	73	74	75	76	77	78	79	7A	A1	BF	D0	DD	DE	AE
B-	A2	A3	A5	B7	A9	A7	B6	BC	BD	BE	AC	5B	5C	5D	B4	D7
C-	F9	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
D-	A6	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	DB	FA	FF
E-	D9	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
F-	30	31	32	33	34	35	36	37	38	39	B3	7B	DC	7D	DA	7E

Table 9: Conversion from EBCDIC (BS2000) to ISO8859

Unused byte positions in ISO8859/ EBCDIC tables

Table	Unused ISO8859 position	Unused EBCDIC.DF.04 position
ISO8859-1	-	-
ISO8859-2	-	-
ISO8859-3	A5,AE,BE,C2,D0,E3,F0	B2,AF,B9,62,AC,46,8C
ISO8859-4	-	-
ISO8859-5	-	-
ISO8859-7	AE,D2	AF,ED
ISO8859-15	-	-

Table 10: Unused byte positions in ISO8859/EBCDIC tables

Contents of ISO and corresponding EBCDIC code tables

ISO table	EBCDIC BS2000/OSD	Contents
ISO/IEC 8859-1	EBCDIC.DF.04-1	Part 1: Latin alphabet No. 1
ISO/IEC 8859-2	EBCDIC.DF.04-2	Part 2: Latin alphabet No. 2
ISO/IEC 8859-3	EBCDIC.DF.04-3	Part 3: Latin alphabet No. 3
ISO/IEC 8859-4	EBCDIC.DF.04-4	Part 4: Latin alphabet No. 4
ISO/IEC 8859-5	EBCDIC.DF.04-5	Part 5: Latin/Cyrillic alphabet
ISO/IEC 8859-6		Part 6: Latin/Arabic alphabet
ISO/IEC 8859-7	EBCDIC.DF.04-7	Part 7: Latin/Greek alphabet
ISO/IEC 8859-8		Part 8: Latin/Hebrew alphabet
ISO/IEC 8859-9	EBCDIC.DF.04-9	Part 9: Latin alphabet No. 5
ISO/IEC 8859-10		Part 10: Latin alphabet No. 6
ISO/IEC 8859-11		Part 11: Latin/Thai alphabet
ISO/IEC 8859-13		Part 13: Latin alphabet No. 7
ISO/IEC 8859-14		Part 14: Latin alphabet No. 8 (Celtic)
ISO/IEC 8859-15	EBCDIC.DF.04-F	Part 15: Latin alphabet No. 9
ISO/IEC 8859-16		Part 16: Latin alphabet No. 10

Table 11: Contents of ISO and corresponding EBCDIC code tables

8.4.2 Unicode characters convertible to ISO8859.n

Unicode 7-bit characters convertible to ISO8859-n

All Unicode 7-bit characters listed in the following table can be converted into the following character sets:

- ISO8859.1 or EBCDIC.DF.04-1
- ISO8859.2 or EBCDIC.DF.04-2
- ISO8859.3 or EBCDIC.DF.04-3
- ISO8859.4 or EBCDIC.DF.04-4
- ISO8859.5 or EBCDIC.DF.04-5
- ISO8859.7 or EBCDIC.DF.04-7
- ISO8859.9 or EBCDIC.DF.04-9
- ISO8859.15 or EBCDIC.DF.04-15

Unicode Code Point	Character
u+0000	0
u+0001	START OF HEADING
u+0002	START OF TEXT
u+0003	END OF TEXT
u+0004	END OF TRANSMISSION
u+0005	ENQUIRY
u+0006	ACKNOWLEDGE
u+0007	BELL
u+0008	BACKSPACE
u+0009	HORIZONTAL TABULATION
u+000A	LINE FEED
u+000B	VERTICAL TABULATION
u+000C	FORM FEED
u+000D	CARRIAGE RETURN
u+000E	SHIFT OUT
u+000F	SHIFT IN
u+0010	DATA LINK ESCAPE
u+0011	DEVICE CONTROL ONE
u+0012	DEVICE CONTROL TWO
u+0013	DEVICE CONTROL THREE
u+0014	DEVICE CONTROL FOUR
u+0015	NEGATIVE ACKNOWLEDGE
u+0016	SYNCHRONOUS IDLE
u+0017	END OF TRANSMISSION BLOCK
u+0018	CANCEL
u+0019	END OF MEDIUM
u+001A	SUBSTITUTE
u+001B	ESCAPE
u+001C	FILE SEPARATOR
u+001D	GROUP SEPARATOR

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode Code Point	Character
u+001E	RECORD SEPARATOR
u+001F	UNIT SEPARATOR
u+0020	SPACE
u+0021	!
u+0022	"
u+0023	#
u+0024	\$
u+0025	%
u+0026	&
u+0027	'
u+0028	(
u+0029)
u+002A	*
u+002B	"+"
u+002C	,
u+002D	-
u+002E	.
u+002F	/
u+0030	0
u+0031	1
u+0032	2
u+0033	3
u+0034	4
u+0035	5
u+0036	6
u+0037	7
u+0038	8
u+0039	9
u+003A	:
u+003B	;

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode Code Point	Character
u+003C	<
u+003D	=
u+003E	>
u+003F	?
u+0040	@
u+0041	A
u+0042	B
u+0043	C
u+0044	D
u+0045	E
u+0046	F
u+0047	G
u+0048	H
u+0049	I
u+004A	J
u+004B	K
u+004C	L
u+004D	M
u+004E	N
u+004F	O
u+0050	P
u+0051	Q
u+0052	R
u+0053	S
u+0054	T
u+0055	U
u+0056	V
u+0057	W
u+0058	X
u+0059	Y

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode Code Point	Character
u+005A	Z
u+005B	[
u+005C	\
u+005D]
u+005E	^
u+005F	_
u+0060	`
u+0061	a
u+0062	b
u+0063	c
u+0064	d
u+0065	e
u+0066	f
u+0067	g
u+0068	h
u+0069	i
u+006A	j
u+006B	k
u+006C	l
u+006D	m
u+006E	n
u+006F	o
u+0070	p
u+0071	q
u+0072	r
u+0073	s
u+0074	t
u+0075	u
u+0076	v
u+0077	w

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode Code Point	Character
u+0078	x
u+0079	y
u+007A	z
u+007B	{
u+007C	
u+007D	}
u+007E	~
u+007F	<control> DELETE
u+0080	<control>
u+0081	<control>
u+0082	<control>= BREAK PERMITTED HERE
u+0083	<control>= NO BREAK HERE
u+0084	<control>
u+0085	<control>=NEXT LINE (NL)
u+0086	<control>= START OF SELECTED AREA
u+0087	<control>= END OF SELECTED AREA
u+0088	<control>= CHARACTER TABULATION SET
u+0089	<control>= CHARACTER TABULATION WITH JUSTIFICATION
u+008A	<control>= LINE TABULATION SET
u+008B	<control>= PARTIAL LINE FORWARD
u+008C	<control>= PARTIAL LINE BACKWARD
u+008D	<control>= REVERSE LINE FEED
u+008E	<control>= SINGLE SHIFT TWO
u+008F	<control>= SINGLE SHIFT THREE
u+0090	<control>= DEVICE CONTROL STRING
u+0091	<control>= PRIVATE USE ONE
u+0092	<control>= PRIVATE USE TWO
u+0093	<control>= SET TRANSMIT STATE
u+0094	<control>= CANCEL CHARACTER

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode Code Point	Character
u+0095	<control>= MESSAGE WAITING
u+0096	<control>= START OF GUARDED AREA
u+0097	<control>= END OF GUARDED AREA
u+0098	<control>= START OF STRING
u+0099	<control>
u+009A	<control>= SINGLE CHARACTER INTRODUCER
u+009B	<control>= CONTROL SEQUENCE INTRODUCER
u+009C	<control>= STRING TERMINATOR
u+009D	<control>= OPERATING SYSTEM COMMAND
u+009E	<control>= PRIVACY MESSAGE
u+009F	<control>= APPLICATION PROGRAM COMMAND

Table 12: Unicode 7-bit characters convertible to ISO8859-n

Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+00A0	1,2,3,4,5,7,9,15	NO-BREAK SPACE
u+00A1	1,9,15	ı
u+00A2	1,9,15	ϕ
u+00A3	1,3,7,9,15	£
u+00A4	1,2,3,5,9	α
u+00A5	1,9,15	¥
u+00A6	1,7,9	ı
u+00A7	1,2,3,4,5,7,9,15	§
u+00A8	1,2,3,4,7,9	¨
u+00A9	1,7,9,15	©
u+00AA	1,9,15	ª
u+00AB	1,7,9,15	«
u+00AC	1,7,9,15	¬
u+00AD	1,2,3,4,5,7,9,15	-
u+00AE	1,9,15	®
u+00AF	1,4,9,15	—
u+00B0	1,2,3,4,7,9,15	°
u+00B1	1,7,9,15	±
u+00B2	1,3,7,9,15	²
u+00B3	1,3,7,9,15	³
u+00B4	1,2,3,4,9	´
u+00B5	1,3,9,15	μ
u+00B6	1,9,15	¶
u+00B7	1,3,7,9,15	·
u+00B8	1,2,3,9	¸
u+00B9	1,9,15	¹
u+00BA	1,9,15	º
u+00BB	1,7,9,15	»
u+00BD	1,3,7,9	½
u+00BE	1,9	¾
u+00BF	1,9,15	¿

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+00C0	1,3,9,15	À
u+00C1	1,2,3,4,9,15	Á
u+00C2	1,2,3,4,9,15	Â
u+00C3	1,4,9,15	Ã
u+00C4	1,2,3,4,9,15	Ä
u+00C5	1,4,9,15	Å
u+00C6	1,4,9,15	Æ
u+00C7	1,2,3,9,15	Ç
u+00C8	1,3,9,15	È
u+00C9	1,2,3,4,9,15	É
u+00CA	1,3,9,15	Ê
u+00CB	1,2,3,4,9,15	Ë
u+00CC	1,3,9,15	Ì
u+00CD	1,2,3,4,9,15	Í
u+00CE	1,2,3,4,9,15	Î
u+00CF	1,3,9,15	Ï
u+00D0	1,15	Ð
u+00D1	1,3,9,15	Ñ
u+00D2	1,3,9,15	Ò
u+00D3	1,2,3,9,15	Ó
u+00D4	1,2,3,4,9,15	Ô
u+00D5	1,4,9,15	Õ
u+00D6	1,2,3,4,9,15	Ö
u+00D7	1,2,3,4,9,15	×
u+00D8	1,4,9,15	Ø
u+00D9	1,3,9,15	Ù
u+00DA	1,2,3,4,9,15	Ú
u+00DB	1,3,4,9,15	Û
u+00DC	1,2,3,4,9,15	Ü
u+00DD	1,2,15	Ý
u+00DE	1,15	Þ

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+00DF	1,2,3,4,9,15	ß
u+00E0	1,3,9,15	à
u+00E1	1,2,3,4,9,15	á
u+00E2	1,2,3,4,9,15	â
u+00E3	1,4,9,15	ã
u+00E4	1,2,3,4,9,15	ä
u+00E5	1,4,9,15	å
u+00E6	1,4,9,15	æ
u+00E7	1,2,3,9,15	ç
u+00E8	1,3,9,15	è
u+00E9	1,2,3,4,9,15	é
u+00EA	1,3,9,15	ê
u+00EB	1,2,3,4,9,15	ë
u+00EC	1,3,9,15	ì
u+00ED	1,2,3,4,9,15	í
u+00EE	1,2,3,4,9,15	î
u+00EF	1,3,9,15	ï
u+00F0	1,15	ð
u+00F1	1,3,9,15	ñ
u+00F2	1,3,9,15	ò
u+00F3	1,2,3,9,15	ó
u+00F4	1,2,3,4,9,15	ô
u+00F5	1,4,9,15	õ
u+00F6	1,2,3,4,9,15	ö
u+00F7	1,2,3,4,9,15	÷
u+00F8	1,4,9,15	ø
u+00F9	1,3,9,15	ù
u+00FA	1,2,3,4,9,15	ú
u+00FB	1,3,4,9,15	û
u+00FC	1,2,3,4,9,15	ü
u+00FD	1,2,15	ý

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+00FE	1,15	þ
u+00FF	1,9,15	ÿ
u+0100	4	Ā
u+0101	4	ā
u+0102	2	Ă
u+0103	2	ă
u+0104	2,4	Ą
u+0105	2,4	ą
u+0106	2	Ć
u+0107	2	ć
u+0108	3	Ĉ
u+0109	3	ĉ
u+010A	3	Č
u+010B	3	č
u+010C	2,4	Č
u+010D	2,3	č
u+010E	2	Ď
u+010F	2	ď
u+0110	2,4	Đ
u+0111	2	đ
u+0112	4	Ē
u+0113	4	ē
u+0116	4	Ĕ
u+0117	4	ė
u+0118	2,4	Ę
u+0119	2,4	ę
u+011A	2	Ė
u+011B	2	ė
u+011C	3	Ĝ
u+011D	3	ĝ
u+011E	3,9	Ğ

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+011F	3,9	ǧ
u+0120	3	Ĝ
u+0121	3	ğ
u+0122	4	Ğ
u+0123	4	ġ
u+0124	3	Ĥ
u+0125	3	ĥ
u+0126	3	Ħ
u+0127	3	ħ
u+0128	4	İ
u+0129	4	ı
u+012A	4	Ī
u+012B	4	ī
u+012E	4	Ĵ
u+012F	4	ĵ
u+0130	3,9	İ
u+0131	3,9	ı
u+0134	3	Ĵ
u+0135	3	ĵ
u+0136	4	Ƙ
u+0137	4	ƙ
u+0138	4	κ
u+0139	2	Ĺ
u+013A	2	ĺ
u+013B	4	Ł
u+013C	4	ł
u+013D	2	Ľ
u+013E	2	ľ
u+0141	2	Ł
u+0142	2	ł
u+0143	2	Ń

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+0144	2	ń
u+0145	4	Ń
u+0146	4	ņ
u+0147	2	Ņ
u+0148	2	ň
u+014A	4	Ŋ
u+014B	4	ŋ
u+014C	4	Ō
u+014D	4	ō
u+0150	2	Ŏ
u+0151	2	ó
u+0152	15	Œ
u+0153	15	œ
u+0154	2	Ŕ
u+0155	2	ř
u+0156	4	Ṛ
u+0157	4	ṛ
u+0158	2	Ř
u+0159	2	ř
u+015A	2	Ś
u+015B	2	ś
u+015C	3	Ŝ
u+015D	3	ŝ
u+015E	2,3,9	Ş
u+015F	2,3,9	ş
u+0160	2,4,15	Š
u+0161	2,4,15	š
u+0162	2	Ţ
u+0163	2	ţ
u+0164	2	Ț
u+0165	2	ț

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+0166	4	ƒ
u+0167	4	‡
u+0168	4	Ū
u+0169	4	ū
u+016A	4	Ů
u+016B	4	ů
u+016C	3	Ů
u+016D	3	ů
u+016E	2	Ů
u+016F	2	ů
u+0170	2	Ů
u+0171	2	ů
u+0172	4	Ů
u+0173	4	ů
u+0178	15	Ÿ
u+0179	2	Ž
u+017A	2	ž
u+017B	2,3	Ž
u+017C	2,3	ž
u+017D	2,4,15	Ž
u+017E	2,4,15	ž
u+02C7	2,4	˘
u+02D8	2,3	˘
u+02D9	2,3,4	˙
u+02DB	2,4	˘
u+02DD	2	˘
u+037A	7	˘
u+0384	7	˘
u+0385	7	˘
u+0386	7	˘
u+0388	7	˘

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+0389	7	Ĥ
u+038A	7	Ħ
u+038C	7	Ŏ
u+038E	7	Ÿ
u+038F	7	Ω
u+0390	7	ĩ
u+0391	7	À
u+0392	7	Β
u+0393	7	Γ
u+0394	7	Δ
u+0395	7	Ε
u+0396	7	Ζ
u+0397	7	Η
u+0398	7	Θ
u+0399	7	Ι
u+039A	7	Κ
u+039B	7	Λ
u+039C	7	Μ
u+039D	7	Ν
u+039E	7	Ξ
u+039F	7	Υ
u+03A0	7	Π
u+03A1	7	Ρ
u+03A3	7	Σ
u+03A4	7	Τ
u+03A5	7	Υ
u+03A6	7	Φ
u+03A7	7	Χ
u+03A8	7	Ψ
u+03A9	7	Ω
u+03AA	7	Ϊ

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+03AB	7	ÿ
u+03AC	7	á
u+03AD	7	é
u+03AE	7	ñ
u+03AF	7	í
u+03B0	7	ü
u+03B1	7	α
u+03B2	7	β
u+03B3	7	γ
u+03B4	7	δ
u+03B5	7	ε
u+03B6	7	ζ
u+03B7	7	η
u+03B8	7	θ
u+03B9	7	ι
u+03BA	7	κ
u+03BB	7	λ
u+03BC	7	μ
u+03BD	7	ν
u+03BE	7	ξ
u+03BF	7	ο
u+03C0	7	π
u+03C1	7	ρ
u+03C2	7	ς
u+03C3	7	σ
u+03C4	7	τ
u+03C5	7	υ
u+03C6	7	φ
u+03C7	7	χ
u+03C8	7	ψ
u+03C9	7	ω

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+03CA	7	ï
u+03CB	7	ü
u+03CC	7	ó
u+03CD	7	ú
u+03CE	7	ώ
u+0401	5	Ě
u+0402	5	Ђ
u+0403	5	ѓ
u+0404	5	Є
u+0405	5	Š
u+0406	5	І
u+0407	5	ї
u+0408	5	Ј
u+0409	5	Љ
u+040A	5	Њ
u+040B	5	Ћ
u+040C	5	Ќ
u+040E	5	Ў
u+040F	5	Џ
u+0410	5	А
u+0411	5	Б
u+0412	5	В
u+0413	5	Г
u+0414	5	Д
u+0415	5	Е
u+0416	5	Ж
u+0417	5	З
u+0418	5	И
u+0419	5	Й
u+041A	5	К
u+041B	5	Л

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+041C	5	М
u+041D	5	Н
u+041E	5	О
u+041F	5	П
u+0420	5	Р
u+0421	5	С
u+0422	5	Т
u+0423	5	У
u+0424	5	Ф
u+0425	5	Х
u+0426	5	Ц
u+0427	5	Ч
u+0428	5	Ш
u+0429	5	Щ
u+042A	5	Ъ
u+042B	5	Ы
u+042C	5	Ь
u+042D	5	Э
u+042E	5	Ю
u+042F	5	Я
u+0430	5	а
u+0431	5	б
u+0432	5	в
u+0433	5	г
u+0434	5	д
u+0435	5	е
u+0436	5	ж
u+0437	5	з
u+0438	5	и
u+0439	5	й
u+043A	5	к

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+043B	5	л
u+043C	5	м
u+043D	5	н
u+043E	5	о
u+043F	5	п
u+0440	5	р
u+0441	5	с
u+0442	5	т
u+0443	5	у
u+0444	5	ф
u+0445	5	х
u+0446	5	ц
u+0447	5	ч
u+0448	5	ш
u+0449	5	щ
u+044A	5	ъ
u+044B	5	ы
u+044C	5	ь
u+044D	5	э
u+044E	5	ю
u+044F	5	я
u+0451	5	ё
u+0452	5	ђ
u+0453	5	ѓ
u+0454	5	е
u+0455	5	ѕ
u+0456	5	і
u+0457	5	ї
u+0458	5	ј
u+0459	5	љ
u+045A	5	њ

Table 13: Unicode characters convertible to ISO8859-n

Unicode Code Point	ISO8859-n / EBCDIC.DF.04.n	Character
u+045B	5	ħ
u+045C	5	ı
u+045E	5	Ÿ
u+045F	5	ı
u+2015	7	–
u+2018	7	‘

Table 13: Unicode characters convertible to ISO8859-n

Characters required for German “Meldewesen” (reporting system) with no equivalent in ISO8859-n

Unicode Code Point	Character
u+0114	Ě
u+0115	ě
u+012C	Ī
u+012D	ī
u+014E	Ŏ
u+014F	ö
u+0166	Ʀ
u+0167	Ƨ
u+0174	Ŵ
u+0175	ŵ
u+0176	Ŷ
u+0177	ŷ
u+019D	Ɔ
u+01A0	Ɔ
u+01A1	σ
u+01AF	Ŭ
u+01B0	ŭ
u+01CD	Ǻ
u+01CE	ǻ
u+01CF	Ǫ
u+01D0	ǫ
u+01D1	Ǿ
u+01D2	ǿ
u+01D3	ǰ
u+01D4	Ǳ
u+01E6	Ǧ
u+01E7	ǧ

Table 14: Characters required for German “Meldewesen” (reporting system) with no equivalent in ISO8859-n

Unicode Code Point	Character
u+01F4	Ĝ
u+01F5	ĝ
u+0212	Ř
u+0213	ř
u+0272	ƚ
u+1E20	Ĝ
u+1E21	ĝ
u+1E24	Ĥ
u+1E25	ĥ
u+1E30	Ķ
u+1E31	ķ
u+1E44	Ñ
u+1E45	ñ
u+1E60	Ŝ
u+1E61	ŝ
u+1E62	Ş
u+1E63	ş
u+1E84	Ŵ
u+1E85	ŵ
u+1E8E	Ŷ
u+1E8F	ŷ
u+1E90	Ž
u+1E91	ž
u+1E92	Ẑ
u+1E93	ẑ
u+1EA0	Ạ
u+1EA1	ạ

Table 14: Characters required for German “Meldewesen” (reporting system) with no equivalent in ISO8859-n

Unicode Code Point	Character
u+1EAA	Ă
u+1EAB	ă
u+1EBC	Ě
u+1EBD	ě
u+1EC4	Ě̃
u+1EC5	ě̃
u+1ECA	İ
u+1ECB	ı
u+1ECC	Ȯ
u+1ECD	ȯ
u+1ED6	Ŏ
u+1ED7	ŏ
u+1EE4	Ȫ
u+1EE5	ȫ
u+1EF2	Ỳ
u+1EF3	ỳ
u+1EF8	Ỳ̃
u+1EF9	ỳ̃
u+2264	≤

Table 14: Characters required for German “Meldewesen” (reporting system) with no equivalent in ISO8859-n

Glossary

Basic Multilingual Plane (BMP)

Corresponds to UCS-2.

UCS-2 can only represent the BMP characters, but characters from the BMP can be saved in UCS-2, UTF-8, etc.

Big Endian

Type of byte ordering by the processor in a particular encoding.

In Big Endian, the most significant byte is stored at the lowest memory address.

Byte Order Marks (BOM)

BOMs indicate whether a UTF-16 or UTF-32 string is in Little Endian or Big Endian format.

Code point

See Unicode code point.

Code space

The set of all code points. The code space of the Unicode standard V4 recognizes 1,114,112 code points. The code space is divided into planes, each encompassing 65,536 code points.

Collation element

Sorting element.

The Unicode standard describes a linguistic sort algorithm, which is based on the fact that each character is assigned a collation element. It consists of a series of up to three levels.

Diacritical mark

Mark which is linked to a base character or symbol, e.g. accent, tilde.

Little Endian

Type of byte ordering by the processor in a particular encoding.

In Little Endian the least significant byte is stored at the lowest memory address.

Normalization

Procedure for consistent representation of characters which can have several Unicode code points (e.g. diacritical marks).

Surrogate pairs

All characters above U+FFFF in the Unicode encoding UTF-16. These are represented by 4 bytes.

U+D800 - U+DBFF: High surrogate

U+DC00- U+DFFF: Low surrogate

A pair of high and low surrogates map the code points from U+10000 through U+10FFFF.

Unicode code point

In Unicode, each character is assigned a number, the so-called code point.

A Unicode code point is generally specified in the form U+n, where n consists of 4 to 6 hexadecimal numbers.

UCS-2 (2-byte Universal Character Set)

In UTF-16, all Unicode characters between U+0000 and U+FFFF are encoded with 2 bytes. The characters in this range are also known as the 2-byte Universal Character Set (UCS-2). The range of surrogates U+D800 through U+DFFF and the byte order marks U+FEFF and U+FFFE are excluded.

UTF-8

Unicode encoding, defined by the Unicode Consortium.

UTF-8 uses a variable number of bytes for encoding the Unicode characters.

The byte representation of the ASCII characters remains unchanged. With a character that is encoded with several bytes, none of the individual bytes represents a valid character.

UTF-8-Mod (modified UTF-8)

Preliminary stage of UTF-EBCDIC. The single-byte encoding of UTF-8 is extended to include the second control-character block (U+80 - 8+9F). The number of possible following bytes is reduced accordingly. With a subsequent conversion from UTF-8MOD to EBCDIC, the representation of the EDF03IRV characters remains unchanged.

A technical report from the Unicode Consortium proposes for systems which use EBCDIC a conversion of Unicode characters to EBCDIC.

UTF-16

Unicode encoding, defined by the Unicode Consortium.

In UTF-16, all Unicode characters between U+0000 and U+FFFF are encoded with 2 bytes. All characters above U+FFFF are represented by 4 bytes, so-called surrogate pairs.

UTF-32

Unicode encoding, defined by the Unicode Consortium.

Every character in the Unicode standard is encoded directly as a 32-bit unit.

UTF-EBCDIC (UTFE)

Unicode encoding for machines which use the EBCDIC character set.

Abbreviations

BMP	Basic Multilingual Plane
ISO	International Standardization Organization
UCS-2	Universal Character Set 2
UCS-4	Universal Character Set 4
UTF-8	UCS Transformation Format 8-bit or Unicode Transformation Format
UTF-8MOD	Modified UTF-8
UTF-16	UCS Transformation Format 16-bit
UTFE	In BS2000/OSD: UTF-EBCDIC

Tables

Table 1: Code space of the Unicode standard V4	13
Table 2: Byte allocation in UTF-8 representation	16
Table 3: Byte allocation in UTF-8MOD representation	19
Table 4: Byte allocation in UTFE representation	21
Table 5: Unicode products: overview and dependencies	57
Table 6: Default values for CCSN	61
Table 7: Actions and associated default values for CCSN	62
Table 8: Conversion from ISO8859 to EBCDIC (BS2000/OSD)	63
Table 9: Conversion from EBCDIC (BS2000) to ISO8859	64
Table 10: Unused byte positions in ISO8859/EBCDIC tables	64
Table 11: Contents of ISO and corresponding EBCDIC code tables	65
Table 12: Unicode 7-bit characters convertible to ISO8859-n	66
Table 13: Unicode characters convertible to ISO8859-n	72
Table 14: Characters required for German “Meldewesen” (reporting system) with no equivalent in ISO8859-n	85

Related publications

The manuals are available as online manuals, see <http://manuals.fujitsu-siemens.com>, or in printed form which must be paid and ordered separately at <http://FSC-manualshop.com>.

- [1] **XHCS** (BS2000/OSD)
8-bit Code and Unicode Support in BS2000/OSD
User Guide
- [2] **PERCON V2.9A** (BS2000/OSD)
User Guide
- [3] **COBOL2000** (BS2000/OSD)
COBOL Compiler
Reference Manual
- [4] **AID** (BS2000/OSD)
Advanced Interactive Debugger
Core Manual
- [5] **AID** (BS2000/OSD)
Debugging of COBOL Programs
- [6] **IFG** (BS2000/OSD)
Interactive Format Generator
User Guide
- [7] **FHS** (BS2000/OSD)
Format Handling System for openUTM, TIAM, DCAM
User Guide
- [8] **SESAM/SQL-Server** (BS2000/OSD)
Core Manual
User Guide
- [9] **SORT V7.9A** (BS2000/OSD)
User Guide

Related publications

- [10] **EDT (BS2000/OSD)**
Statements
User Guide
- [11] *openFT V10.0A for BS2000/OSD*
Enterprise File Transfer in the Open World
User Guide
- [12] *openFT V10.0A for BS2000/OSD*
Installation and Administration
System Administrator Guide
- [13] **RSO (BS2000/OSD)**
Remote Spool Output
- [14] *WebTransactions*
Concepts and Functions
- [15] *WebTransactions*
Connecting to OSD Applications
User Guide
- [16] *WebTransactions*
Connecting to openUTM Applications via UPIC
User Guide

Additional publications

Website of the Unicode Consortium:

<http://www.unicode.org/standard/WhatIsUnicode.html>

Oracle User's Guide for Fujitsu Siemens Computers BS2000/OSD

<http://www.oracle.com/technology/documentation/database10gr2.html>

Oracle Database Globalization Support Guide

Part Number B14225-02

Oracle Database Upgrade Guide

Part Number B14238-01

Index

formats (FHS)
processing 43

%C()
AID function 38
%SHOW %CCSN (AID) 38
%UTF16 (AID data type) 38
%UTF16()
AID function 38

A

Advanced Interactive Debugger (AID) 38
AFP-IPDS printer 45
AID 38
%SHOW%CCSN 38
data type %UTF16 38
AID (Advanced Interactive Debugger) 38
AID function
%C() 38
%UTF16() 38
AL16UTF-16 42
ALTER CATALOG (SESAM/SQL) 40
application scenario 27
applications
Web integration 47
ASCII compatibility 15

B

Basic Multilingual Plane 13
Big Endian 22
BMP 13
byte positions (EBCDIC table)
unused 64
byte positions (ISO8859 tables)
unused 64

C

CCSN
actions and associated default values 62
concept 27
default values 61
SESAM/SQL 40, 53
central printers 45
CJK scripts 9
COBOL
representation/processing of Unicode
characters 37
COBOL standard ISO1989 2002 37
COBOL2000 V1.4 37
code compatibility 36
code point 13
code space 13
code table
conversion from EBCDIC (BS2000/OSD) to
ISO8859 64
conversion from ISO8859 to EBCDIC
(BS2000/OSD) 63
Coded Character Set Name
actions and associated default values 62
concept 27
default values 61
PERCON 51
SESAM/SQL 40
coded character sets in utility functions (SESAM/
SQL) 40
collation element 24, 36, 51
compatibility
Oracle 10g 42
SESAM/SQL 41
compatibility mode
EDT 50

COMPOSE (function) 23
conversion
 between encodings 36
 EBCDIC (BS2000/OSD) to ISO8859 (code table) 64
 ISO8859 to EBCDIC (BS2000/OSD) (code table) 63
 Oracle 10g 41
 to Unicode 51
 Unicode characters to EBCDIC 17
conversion function
 TO_CHAR (Oracle 10g) 42
 TO_NCHAR (Oracle 10g) 42
 TRANSLATE (Oracle 10g) 42
 TRANSLATE (SESAM/SQL) 39
CREATE CATALOG (SESAM/SQL) 40

D

data class
 national 37
data type
 %UTF16 (AID) 38
DCSTA
 BS2000 macro 60
decentralized printers 45
DECOMPOSE (function) 23
definition
 additional Unicode characters 36
dependencies of Unicode products in BS2000/
 OSD 57
diacritical mark 23

E

EBCDIC and corresponding ISO code table
 contents 65
EBCDIC table
 unused byte positions 64
EBCDIC.DF.03IRV 10, 18, 45
EDF03IRV 18
EDT 50
 compatibility mode 50
 Unicode mode 50
encodings
 conversion 36

export
 of Unicode data (Oracle 10g) 42
EXPORT TABLE (SESAM/SQL) 40
EXtended Host Code Support 36
EXTRACT-ELEMENT 54

F

FHS 31, 43
 processing # formats 43
file editing
 with EDT 50
file transfer (Unicode) 52
format
 in Unicode mode 43
Format Handling System 43
formats
 defining Unicode fields 44
 Unicode characters 43
function
 COMPOSE 23
 DECOMPOSE 23

G

GCCSN
 BS2000 macro 59
German Meldewesen
 characters with no equivalent in ISO8859-
 n 85

I

IFG 31, 43, 44
import
 of Unicode data (Oracle 10g) 42
IMPORT TABLE (SESAM/SQL) 40
integrating Unicode-capable applications into the
 Web 47
Interactive Format Generator 43
interfaces (BS2000/OSD)
 Unicode encodings 29
ISO and corresponding EBCDIC code table
 contents 65
ISO8859 tables
 unused byte positions 64

L

level (Unicode code space) 13
 linguistic sort algorithm (Unicode standard) 24
 LINKED-IN-ATTRIBUTES 53
 Little Endian 22
 LOAD (SESAM/SQL) 40

M

mark
 diacritical 23
 MT9750 configuration 32

N

national
 data class 37
 NATIONAL CHARACTER (data type) 39
 NATIONAL CHARACTER VARYING (data type) 39
 NCHAR 39, 41
 NCHR (SQL function Oracle 10g) 42
 NCLOB 41
 normalization 23, 36, 51
 NVARCHAR 39
 NVARCHAR2 41

O

*open*FT 52
 Oracle 10g 41
 compatibility 42
 Oracle applications
 processing 42

P

PERCON 51
 Peripheral Converter (PERCON) 51
 plane (Unicode code space) 13
 printers
 central 45
 decentralized 45
 PRINTRONIX P7000 46
 processing
 Oracle applications 42

R

RDATA interface 31
 Remote Spool Output 45
 RSO 45

S

SESAM/SQL 39
 compatibility 41
 conversion function TRANSLATE 39
 Unicode concept 39
 SESAM/SQL applications 40
 SORT 51
 sort key 24
 sort sequence 24, 36
 Unicode standard 24
 sorting
 Unicode fields 51
 SQL language description
 Unicode support 39
 SQL*Loader 42
 surrogate pair 22

T

terminal emulation MT9750
 configuration 32
 TO_CHAR
 conversion function in Oracle 10g 42
 TO_NCHAR
 conversion function in Oracle 10g 42
 transfer
 Unicode files 52
 TRANSLATE
 conversion function in Oracle 10g 42
 conversion function in SESAM/SQL 39
 transport layer
 Unicode encoding 31
 TSTAT
 BS2000 macro 60

U

UCS-2 9, 22
 UCS-4 9
 UNICODE
 field attribute 43

Unicode

- conversion to [51](#)
 - defining fields in formats [44](#)
 - in formats [43](#)
- Unicode 7-bit characters
- convertible to ISO8859.x [65](#)
- Unicode characters
- convertible to ISO8859.n [72](#)
 - definition [36](#)
 - representation/processing (COBOL) [37](#)
- Unicode Default Collation Table [25, 51](#)
- Unicode encoding
- on BS2000/OSD interfaces [29](#)
 - transport layer [31](#)
- Unicode fields
- sorting [51](#)
- Unicode files
- transfer [52](#)
- Unicode format [43](#)
- Unicode in BS2000/OSD
- basic considerations [28](#)
- Unicode mode
- EDT [50](#)
- Unicode products in BS2000/OSD
- dependencies [57](#)
- Unicode support
- WebTransactions* for openUTM [48](#)
 - WebTransactions* for OSD [47](#)
- UNISTR (SQL function Oracle 10g) [42](#)
- UNLOAD (SESAM/SQL) [40](#)
- UTF-16 [10, 22](#)
- UTF-32 [22](#)
- UTF-8 [10, 15](#)
- ASCII compatibility [15](#)
- UTF-8MOD [17](#)
- byte allocation [19](#)
- UTFE [17](#)
- byte allocation [21](#)
 - Oracle 10g [41](#)
 - SESAM/SQL [54](#)
- UTF-EBCDIC [17](#)
- utility functions (SESAM/SQL)
- coded character sets [40](#)

V

- VTSU [31](#)
- VTSUCB
 - BS2000 macro [60](#)

W

- Web integration
 - of Unicode-capable applications [47](#)
- WebTransactions* [47](#)
- WebTransactions* for openUTM
 - Unicode support [48](#)
- WebTransactions* for OSD
 - Unicode support [47](#)
- WROUT
 - BS2000 macro [59](#)

X

- XHCS [36](#)



Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at

<http://ts.fujitsu.com/...>

and the user documentation at <http://manuals.ts.fujitsu.com>.

Copyright Fujitsu Technology Solutions, 2009

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter <http://de.ts.fujitsu.com/...>, und unter <http://manuals.ts.fujitsu.com> finden Sie die Benutzerdokumentation.

Copyright Fujitsu Technology Solutions, 2009