



iSeries

National Language Support APIs

Version 5 Release 3





@server

iSeries

National Language Support APIs

Version 5 Release 3

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 147.

Sixth Edition (August 2005)

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

National Language Support APIs	1
APIs	1
National Language Support-related APIs	1
UniNextCompChar()—Advance to Next Composite Character Sequence API	2
Parameters	2
Return Value	3
Error Conditions	3
UniQueryCompCharLen()—Composite Character Sequence Code Element Count API.	3
Parameters	4
Return Value	4
Error Conditions	4
Convert Case (QLGCNVCS, QlgConvertCase) API.	5
Authorities and Locks	5
Required Parameter Group	5
Format of Request Control Block	6
Field Descriptions	6
Error Messages	8
Convert Sort Sequence Table (QLGCNVSS) API.	9
Required Parameter Group	9
Error Messages	10
QlgCvtTextDescToDesc()—Convert Text Descriptor API	10
Authorities and Locks	11
Parameters	11
Return Value	12
Error Conditions.	12
Usage Notes	13
UniQueryCompChar()—Number of Composite Character Sequences API	15
Parameters	15
Return Value	15
Error Conditions.	15
Retrieve CCSID Data (QLGRTVCD)	16
Authorities and Locks	16
Required Parameter Group	16
Format of Receiver Variable	17
Field Descriptions	17
Error Messages	18
Retrieve CCSID Text (QLGRTVCT) API	18
Authorities and Locks	18
Required Parameter Group	18
Format of the Generated Information.	19
Field Descriptions	20
Error Messages	20
Retrieve Country or Region Identifiers (QLGRTVCI) API	21
Authorities and Locks	21
Required Parameter Group	21
Format of the Generated Information.	22
Field Descriptions	22
Error Messages	22
Retrieve Default CCSID (QLGRTVDC)	23
Authorities and Locks	23
Required Parameter Group	23
Error Messages	23
Retrieve Language IDs (QLGRTVLI) API	24
Required Parameter Group	24
RTVL0100 Format	24
Field Descriptions	25
Error Messages	25
Retrieve Language Information (QLGRLNGI) API	26
Authorities and Locks	26
Required Parameter Group	26
LNGI0100 Format	27
LNGI0110 Format	27
Field Descriptions	28
Format of the Returned Information	28
Field Descriptions	29
Error Messages	29
Retrieve Locale Information (QLGRTVLC, QlgRetrieveLocaleInformation) API	30
Authorizations and Locks	30
Required Parameters	30
Locale Category Keys	32
Locale Category Key Descriptions	32
Format of the Locale Information	33
Locale Category Formats	33
LC_CTYPE - Character Classification and Conversion Format	33
LC_COLLATE - Character Collation Format	45
LC_TIME - Date and Time Formatting Format.	52
LC_NUMERIC - Numeric and Non-Monetary Formatting Format	54
LC_MESSAGES - Affirmative and Negative Responses Format	55
LC_MONETARY - Monetary Related Numeric Formatting Format	55
LC_TOD - Time Zone Information Format	56
LC_ALL - All Locale Categories	57
Field Descriptions	57
Error Messages	66
Retrieve Sort Sequence Table (QLGRTVSS) API	67
Authorities and Locks	67
Required Parameter Group	67
RSST0100 Format	69
RSST0200 Format	69
Field Descriptions	70
Layout of a Returned UCS-2 Sort Sequence Table	71
Error Messages	72
Scan String for Mixed Data (QLGSCNMX) API	73
Required Parameter Group	73
Error Messages	74
Sort (QLGSORT) API	75
Authorities and Locks	77
Required Parameter Group	77
Optional Parameter Group	78
Format of Request Control Block	79
Format of Returned Records Feedback Information	80
Field Descriptions	80

Buffer Layout Examples	88	Authorities and Locks	119
Error Messages	90	Required Parameter Group	119
Sort Input/Output (QLGSRTIO) API	91	Optional Parameter Group	119
Authorities and Locks	92	Error Messages	120
Required Parameter Group	92	Character Data Representation Architecture	
Optional Parameter Group	93	(CDRA) APIs	121
Format of Request Control Block	93	Convert a Graphic Character String (CDRCVRT,	
Format of Output Data Information	93	QTQCVRT) API	123
Format of Returned Records Feedback		Required Parameter Group	124
Information	94	Feedback Codes and Conditions	125
Field Descriptions	94	Usage Notes	127
Error Messages	96	Get CCSID for Normalization (CDRGCCN,	
QlgTransformUCSData()—Transform UCS Data API	97	QTQGCCN) API	128
Parameters	97	Required Parameter Group	128
Authorities and Locks	98	Feedback Codes and Conditions	129
Return Value	98	Usage Notes	129
Usage Notes:	99	Get Control Function Definition (CDRGCTL,	
Error Messages	99	QTQGCTL) API	130
Truncate Character Data (QLGTRDTA) API	100	Required Parameter Group	131
Required Parameter Group	101	Feedback Codes and Conditions	132
Error Messages	102	Usage Notes	133
QtqValidateCCSID()—Validate CCSID API	102	Get Encoding Scheme, Character Set, and Code	
Parameters	103	Page Elements (CDRGESP, QTQGESP) API	133
Return Value	103	Required Parameter Group	133
Validate Language ID (QLGVLDID) API	103	Feedback Codes and Conditions	134
Required Parameter Group	103	Usage Notes	135
Error Messages	104	Get Related Default CCSID (CDRGRDC,	
Data Conversion APIs	104	QTQGRDC) API	135
iconv()—Code Conversion API	105	Required Parameter Group	135
Parameters	105	Feedback Codes and Conditions	136
Restrictions	106	Usage Notes	136
Return Value	107	Get Short Form (CCSID) from Specified ES and	
Error Conditions	107	(CS, CP) Pair (CDRSCSP, QTQSCSP) API	137
Usage Notes	108	Required Parameter Group	138
Related Information	108	Feedback Codes and Conditions	138
iconv_open()—Code Conversion Allocation API	109	Usage Notes	139
Parameters	109	Get Short Form with Maximal CS for Specified ES	
Format of fromcode and tocode	109	and CP (CDRSMXC, QTQSMXC) API	140
Field Descriptions	110	Required Parameters	140
Return Value	112	Feedback Codes and Conditions	141
Error Conditions	112	Usage Notes	142
Usage Notes	112	Get Short Form with Maximal CS for Specified ES	
Related Information	112	and CP (QTQSMXC2) API	142
QtqIconvOpen()—Code Conversion Allocation API	113	Required Parameters	143
Parameters	113	Feedback Codes and Conditions	144
Format of QtqCode_T	113	Usage Notes	144
Field Descriptions	114	Resolve Client Server CCSID (QTQRCSC) API	145
Return Value	115	Required Parameters	145
Error Conditions	115	Feedback Codes and Conditions	146
Related Information	116	Usage Notes	146
iconv_close()—Code Conversion Deallocation API	116	Appendix. Notices	147
Parameters	117	Trademarks	148
Return Value	117	Terms and conditions for downloading and	
Error Conditions	117	printing publications	149
Usage Notes	117	Code disclaimer information	150
Related Information	117		
Convert Data (QDCXLATE) API	118		

National Language Support APIs

The national language support APIs include:

- “National Language Support-related APIs”
- “Data Conversion APIs” on page 104
- “Character Data Representation Architecture (CDRA) APIs” on page 121

APIs by category

APIs

These are the APIs for this category.

National Language Support-related APIs

The national language support-related APIs work with the national language support (NLS) functions.

The NLS-related APIs that work with UCS2 (Universal Multiple-Octet Coded Character Set with 16 bits per character) character sets are:

- Advance to Next Composite Character Sequence (UniNextCompChar()) API
- Composite Character Sequence Code Element Count (UniQueryCompCharLen()) API
- Convert Case (QLGCNVCS, QlgConvertCase) API
- Number of Composite Character Sequences (UniQueryCompChar()) API
- Retrieve Locale Information (QLGRTVLC, QlgRetrieveLocaleInformation) API
- Transform UCS Data (QlgTransformUCSData()) API

The NLS-related APIs are:

- “UniNextCompChar()—Advance to Next Composite Character Sequence API” on page 2 (UniNextCompChar()) locates the next non-combining character in a string.
- “UniQueryCompCharLen()—Composite Character Sequence Code Element Count API” on page 3 (UniQueryCompCharLen()) computes the number of code elements in a composite character sequence.
- “Convert Case (QLGCNVCS, QlgConvertCase) API” on page 5 (QLGCNVCS, QlgConvertCase) provides case conversion to either uppercase or lowercase.
- “Convert Sort Sequence Table (QLGCNVSS) API” on page 9 (QLGCNVSS) converts a sort sequence table from one coded character set identifier (CCSID) to another.
- “QlgCvtTextDescToDesc()—Convert Text Descriptor API” on page 10 (QlgCvtTextDescToDesc) converts a descriptor of text from one type (CCSID, for example) to another type (IANA name, for example).
- “UniQueryCompChar()—Number of Composite Character Sequences API” on page 15 (UniQueryCompChar()) computes the number of composite character sequences in a code element array.
- “Retrieve CCSID Data (QLGRTVCD)” on page 16 (QLGRTVCD) retrieves different subsets of CCSIDs based on the selection type.
- “Retrieve CCSID Text (QLGRTVCT) API” on page 18 (QLGRTVCT) retrieves different subsets of CCSIDs’ values and their descriptions, if available.
- “Retrieve Country or Region Identifiers (QLGRTVCI) API” on page 21 (QLGRTVCI) retrieves a list of country or region identifiers and their descriptions.

- “Retrieve Default CCSID (QLGRTVDC)” on page 23 (QLGRTVDC) retrieves the default CCSID given a language ID.
- “Retrieve Language IDs (QLGRTVLI) API” on page 24 (QLGRTVLI) retrieves a list of language identifiers.
- “Retrieve Language Information (QLGRLNGI) API” on page 26 (QLGRLNGI) returns a selected national language version (NLV) based on the specified product, option, and language identifier.
- “Retrieve Locale Information (QLGRTVLC, QlgRetrieveLocaleInformation) API” on page 30 (QLGRTVLC, QlgRetrieveLocaleInformation) retrieves one or all categories of a locale.
- “Retrieve Sort Sequence Table (QLGRTVSS) API” on page 67 (QLGRTVSS) retrieves a specified sort sequence table.
- “Scan String for Mixed Data (QLGSCNMX) API” on page 73 (QLGSCNMX) tests a mixed input string for double-byte characters.
- “Sort (QLGSORT) API” on page 75 (QLGSORT) provides a generalized sort function.
- “Sort Input/Output (QLGSRTIO) API” on page 91 (QLGSRTIO) provides a set of records to be sorted or returns a set of records that have already been sorted.
- “QlgTransformUCSData()—Transform UCS Data API” on page 97 (QlgTransformUCSData()) transforms, through a formula as compared to a mapping, data from one form of Unicode to another.
- “Truncate Character Data (QLGTRDTA) API” on page 100 (QLGTRDTA) truncates a CCSID-tagged string of character data to a specified length.
- “QtqValidateCCSID()—Validate CCSID API” on page 102 (QtqValidateCCSID()) determines whether the specified CCSID is supported by the iSeries.
- “Validate Language ID (QLGVLI) API” on page 103 (QLGVLI) ensures that a language identifier is supported.

Top | “National Language Support APIs,” on page 1 | APIs by category

UniNextCompChar()—Advance to Next Composite Character Sequence API

Syntax

```
#include <UniChar.h>

int UniNextCompChar(void *reserved,
                   UniChar *ucs,
                   UniChar **next_sequence);
```

Service Program: QTQULSFN

Default Public Authority: *USE

Threadsafe: No

The **UniNextCompChar**—Advance to Next Composite Character Sequence function locates the next non-combining character in the string *ucs*.

Parameters

reserved

INPUT; PTR(SPP)

This parameter must be set to null.

ucs INPUT; PTR(SPP)

The composite character sequence to be analyzed.

next_sequence

OUTPUT; PTR(SPP)

This is set to the next non-combining character.

Return Value

zero **UniNextCompChar** function was successful. The value returned in *next_sequence* is a pointer to the next non-combining character in *ucs*.

non-zero

UniNextCompChar was not successful and *next_sequence* is set to the NULL pointer.

Error Conditions

If **UniNextCompChar** is not successful, the return value indicates one of the following errors:

UNI_BADOBJ

UniNextCompChar function detected that the *reserved* parameter is not null.

UNI_INVALID

UniNextCompChar function detected that *ucs* is a null pointer.

UNI_NODATA

UniNextCompChar function detected that *ucs* points to a null code element.

UNI_COMP_FOUND

UniNextCompChar function detected that *ucs* points to a combining character. The caller should advance *ucs* by one **UniChar** and call the function again until this error no longer occurs.

UNI_RANGE

UniQueryCompCharLen function detected that the value to be set in *element_count* was too large to fit.

API introduced: V3R6

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

UniQueryCompCharLen()—Composite Character Sequence Code Element Count API

Syntax

```
#include <UniChar.h>

int UniQueryCompCharLen(void *reserved,
                        const UniChar *ucs,
                        size_t *element_count);
```

Service Program: QTQULSFN

Default Public Authority: *USE

Threadsafe: No

The **UniQueryCompCharLen**—Composite Character Sequence Code Element Count function computes the number of code elements in the composite character sequence pointed to by *ucs*.

The function sets the number of code elements. This includes the non-combining character and the zero or more following combining characters up to, but not including, the next non-combining character or null code element into *element_count*.

Parameters

reserved

INPUT; PTR(SPP)

This parameter must be set to null.

ucs INPUT; PTR(SPP)

The composite character sequence to be analyzed.

element_count

OUTPUT; PTR(SPP)

The number of code elements found in *ucs*. This is set to zero if the return value is not zero.

Return Value

zero **UniQueryCompCharLen** function was successful. The value returned in *element_count* is the number of code elements in *ucs*.

non-zero

UniQueryCompCharLen was not successful and *element_count* is set to zero.

Error Conditions

If **UniQueryCompCharLen** is not successful, the return value indicates one of the following errors:

UNI_BADOBJ

UniQueryCompCharLen function detected that the *reserved* parameter is not null.

UNI_INVALID

UniQueryCompCharLen function detected that *ucs* is a null pointer.

UNI_NODATA

UniQueryCompCharLen function detected that *ucs* points to a null code element.

UNI_COMP_FOUND

UniQueryCompCharLen function detected that *ucs* points to a combining character.

UNI_RANGE

UniQueryCompCharLen function detected that the value to be set in *element_count* was too large to fit.

API introduced: V3R6

Top | “National Language Support APIs,” on page 1 | APIs by category

Convert Case (QLGCNVCS, QlgConvertCase) API

Required Parameter Group:	
1	Request control block
Input	Char(*)
2	Input data
Input	Char(*)
3	Output data
Output	Char(*)
4	Length of data
Input	Binary(4)
5	Error code
I/O	Char(*)
	Service Program: QLGCASE
Default Public Authority: *USE	
Threadsafe: No	

The Convert Case (OPM, QLGCNVCS; ILE, QlgConvertCase) API provides a case conversion function that can be directly called by any application program. This API can be used to convert character data to either uppercase or lowercase.

This API supports conversion for single-byte, mixed-byte, and UCS2 (Universal Multiple-Octet Coded Character Set with 16 bits per character) character sets. For the mixed-byte character set data, only the single-byte portion of the data is converted. This API does not convert double-byte character data from any double-byte character set (DBCS) or from a mixed-byte character set.

This API can base case conversion on a CCSID, whereas the Convert Data (QDCXLATE) API uses only table objects.

Authorities and Locks

Table Authority

*USE

Table Library

*USE

Table Lock

*SHRNUP

Required Parameter Group

Request control block

INPUT; CHAR(*)

The information that defines the case conversion alternatives. Refer to "Format of Request Control Block" on page 6 for details.

Input data

INPUT; CHAR(*)

The input data being converted.

Output data

OUTPUT; CHAR(*)

The converted output data that is returned to the calling program. The storage allocated for this output buffer parameter must be at least as long as the input data buffer or unpredictable results may occur.

Length of data

INPUT; BINARY(4)

The length of the data being converted in the input data parameter. Any data that extends beyond the specified length is ignored. The specified length must be a value greater than 0 and less than 16 773 104 bytes.

The length of the storage allocated for the output data is assumed to be the same size as the length of the input data.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Format of Request Control Block

The following table shows the layout of the request control block. For a detailed description of each field, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Type of request
Note: The rest of the layout when the request is 1 (CCSID format)			
4	4	BINARY(4)	CCSID of input data
8	8	BINARY(4)	Case request
12	C	CHAR(10)	Reserved
Note: The rest of the layout when the request is 2 (table object format)			
4	4	BINARY(4)	DBCS indicator
8	8	CHAR(20)	Qualified table name
Note: The rest of the layout when the request is 3 (user-defined format)			
4	4	BINARY(4)	DBCS indicator
8	8	BINARY(4)	Reserved
12	C	BINARY(4)	Length of user-defined table
16	10	CHAR(*)	User-defined case conversion table

Field Descriptions

Case request. A request that specifies the type of case conversion to be performed. Valid values are:

- | | |
|---|--------------------------------------|
| 0 | Convert the input data to uppercase. |
| 1 | Convert the input data to lowercase. |

CCSID of input data. The coded character set identifier (CCSID) of the input data to be converted. Case conversion is performed based on this CCSID. For a list of valid CCSIDs, see the Globalization topic in the iSeries Information Center. The valid values are:

0	The CCSID of the job is used to determine the CCSID of the data to be converted. If the job CCSID is 65535, the CCSID from the default CCSID (DFTCCSID) job attribute is used.
1-65533	A valid CCSID in this range.

DBCS indicator. The indicator that specifies whether or not double-byte data exists in the input data.

0	Input data does not contain any double-byte values.
1	Input data may contain mixed values where the double-byte data is surrounded by shift-out and shift-in characters. Any double-byte data found between a shift-out and shift-in character is ignored and returned to the output data buffer as is.

When the DBCS indicator is 0 but the input data contains double-byte values, this API processes the double-byte data as single-byte data. This also pertains to the shift-in and shift-out characters. It is recommended that an indicator of 0 be used when processing input data that contains only single-byte data because the performance is faster.

Length of user-defined table. The length (in bytes) of the user-defined case conversion table. This length must be 256.

Qualified table name. The table used to convert the data. The first 10 characters contain the table name. The second 10 characters contain the name of the library in which the table resides.

The system-supplied tables only provide uppercasing of the data. For a list of valid case conversion tables supplied by the system, see the Globalization topic in the iSeries Information Center.

You can use the following special values for the library name:

*LIBL	The library list.
*CURLIB	The job's current library.

Reserved. A reserved field that must be set to hexadecimal zeros.

Type of request. The input information format of the case conversion being requested. The layout of the request control block parameter is based on one of the following values:

1	CCSID format
2	Table object format
3	User-defined format

User-defined case conversion table. The user-defined case conversion table used to convert the input data.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3BC7 E	CCSID &1 outside of valid range.
CPF3BDE E	CCSID &1 not supported by API.
CPF3BE5 E	Value &1 for case request not valid.
CPF3BE8 E	Value &1 for DBCS indicator not valid.
CPF3BE9 E	Value &1 for reserved field not valid.
CPF3BEA E	User-defined table length &1 not valid.
CPF3BEB E	Type of request &1 not valid.
CPF3BEC E	Value &1 for reserved field not valid.
CPF3C12 E	Length of data is not valid.
CPF3C1E E	Required parameter &1 omitted.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid. (This message applies only to the QLGCVCS API, not the QlgConvertCase API.)
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9801 E	Object &2 in library &3 not found.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF9805 E	Object &2 in library &3 destroyed.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9809 E	Library &1 cannot be accessed.
CPF9810 E	Library &1 not found.
CPF9830 E	Cannot assign library &1.

API introduced: V3R1

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Convert Sort Sequence Table (QLGCNVSS) API

Required Parameter Group:	
1	Converted sort sequence table
Output	Char(256)
2	Type of returned sort sequence table
Output	Char(1)
3	Substitution values encountered
Output	Char(1)
4	Source sort sequence table
Input	Char(256)
5	CCSID of source table
Input	Binary(4)
6	CCSID of converted table
Input	Binary(4)
7	Error code
I/O	Char(*)
	Default Public Authority: *EXCLUDE
Threadsafe: No	

The Convert Sort Sequence Table (QLGCNVSS) API converts a 256-byte sort sequence table from one coded character set identifier (CCSID) encoding to another.

Required Parameter Group

Converted sort sequence table

OUTPUT; CHAR(256)

The converted sort sequence table.

Type of returned sort sequence table

OUTPUT; CHAR(1)

The type of sort sequence table returned. Possible values are:

- | | |
|---|------------------------|
| 1 | A shared-weight table. |
| 2 | A unique-weight table. |

Substitution values encountered

OUTPUT; CHAR(1)

Whether substitution values were involved during the conversion from the source CCSID of the table to the requested CCSID. Possible values are:

- | | |
|---|---------------------------------|
| 0 | No substitutions were involved. |
| 1 | Substitutions were involved. |

Source sort sequence table

INPUT; CHAR(256)

The source sort sequence table to be converted.

CCSID of source table

INPUT; BINARY(4)

The CCSID of the source table. The valid range is 1 to 65533 and 65535.

CCSID of converted table

INPUT; BINARY(4)

The CCSID to which the source table is to be converted. The valid range is 1 to 65533 and 65535.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3BC7 E	CCSID &1 outside of valid range.
CPF3BC8 E	Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E	Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "National Language Support APIs," on page 1 | APIs by category

QlgCvtTextDescToDesc()—Convert Text Descriptor API

Syntax

```
int QlgCvtTextDescToDesc(int InType,
                        int OutType,
                        char *InDescriptor,
                        int InDescSize,
                        char *OutDescriptor,
                        int OutDescSize,
                        int JobCCSID);
```

Service Program: QLGUSR

Default Public Authority: *USE

Threadsafe: Yes

The **QlgCvtTextDescToDesc()** function converts a descriptor of text from one type (CCSID, for example) to another type (IANA name, for example). (IANA is the Internet Assigned Number Authority.)

An example of a use of this support would be to convert an IANA name of ISO-8859-1 to an OS/400 CCSID of 819. For more information see the Usage Notes.

Authorities and Locks

API Public Authority
*USE

Parameters

InType

INPUT

Type of descriptor provided.

0	CCSID
1	OS/400 supported CCSID
2	short text description
3	AIX 3.1
4	AIX 4.1
5	Windows 3.1
6	Windows 95
7	Windows NT
8	OS/2 Version 3
9	OS/2 Version 4
10	MVS ^(TM) description
11	IANA string
12	Developer Kit for Java
» 13	iSeries Default Java Encoding «

OutType

INPUT

Type of descriptor requested.

0	CCSID
1	OS/400 supported CCSID
2	short text description
3	AIX 3.1
4	AIX 4.1
5	Windows 3.1
6	Windows 95
7	Windows NT
8	OS/2 Version 3
9	OS/2 Version 4
10	MVS description
11	IANA string
12	Developer Kit for Java
» 13	iSeries Default Java Encoding «

InDescriptor

INPUT

The input descriptor to be converted.

InDescSize

INPUT

The size (in bytes) of the descriptor to be converted.

OutDescriptor
OUTPUT

The converted descriptor.

OutDescSize
INPUT

The size (in bytes) of the out descriptor area.

JobCCSID
INPUT

The CCSID that the *InDescriptor* and *OutDescriptor* are encoded in. A value of zero means to use the job's default CCSID.

Return Value

- >0 **QlgCvtTextDescToDesc** function was successful. The value returned in *OutDescriptor* is a pointer to a descriptor of the requested type. The value returned is the length (in bytes) of the returned descriptor.
- <0 **QlgCvtTextDescToDesc** was not successful.

Error Conditions

If **QlgCvtTextDescToDesc** is not successful, the return value indicates one of the following errors:

- 1 **Qlg_InternalError1**
QlgCvtTextDescToDesc function detected that an unknown resource failure occurred.
- 2 **Qlg_InternalError2**
QlgCvtTextDescToDesc function detected that an internal table mismatch occurred.
- 3 **Qlg_BadInType**
QlgCvtTextDescToDesc function detected that InType value was not recognized.
- 4 **Qlg_BadOutType**
QlgCvtTextDescToDesc function detected that OutType value was not recognized.
- 5 **Qlg_InTypeEqualsOutType**
QlgCvtTextDescToDesc function detected that the InType equals the OutType value.
- 6 **Qlg_CCSIDMapError**
QlgCvtTextDescToDesc function detected that the OutDescriptor will equal the InDescriptor in this case.
- 7 **Qlg_BadCCSIDInDesc**
QlgCvtTextDescToDesc function detected that the InDescriptor's CCSID is not a valid CCSID number.
- 8 **Qlg_JobCCSIDNotFound**
QlgCvtTextDescToDesc function detected that the value for JobCCSID is not defined to the system.
- 9 **Qlg_BadJobCCSID**
QlgCvtTextDescToDesc function detected that the value for JobCCSID is not a valid CCSID number.
- 10 **Qlg_InDescConvertFail**
QlgCvtTextDescToDesc function detected that the value InDescriptor cannot be converted to CCSID 37.
- 11 **Qlg_InDescriptorNotFound**
QlgCvtTextDescToDesc function detected that the value for InDescriptor is unknown.

- 12 Qlg_OutTypeDescNotFound
QlgCvtTextDescToDesc function detected that the value for InDescriptor is known, but could not be related to the requested OutDescriptor.
- 13 Qlg_OutDescSizeTooSmall
QlgCvtTextDescToDesc function detected that OutDescSize is too small to hold the requested OutDescriptor.
- 14 Qlg_OutDescSizeExceedsMax
QlgCvtTextDescToDesc function detected that the OutDescriptor exceeds the maximum defined in the header file.

Usage Notes

The following is an example of the kind of information returned by this API for the possible choices. An asterisk (*) indicates that the descriptor is not supported for this CCSID.

Sample Information from the API for CCSID 00037

Descriptor type	String returned
CCSID	00037
OS/400 Supported CCSID	00037
short text description	COM_EUROPE_EBCDIC
AIX 3.1	IBM-037
AIX 4.1	IBM-037
Windows 3.1	CP00037
Windows 95	*
Windows NT	*
OS/2 Version 3	*
OS/2 Version 4	*
MVS description	IBM-037
IANA string	IBM037
Developer Kit for Java	Cp037
➤ iSeries Default Java Encoding	ISO8859_1 ⚡

Table 2. Sample Information from the API for CCSID 00932

Descriptor type	String returned
CCSID	00932
OS/400 Supported CCSID	00932
short text description	JAPAN_MIX_PC-DATA
AIX 3.1	IBM-932
AIX 4.1	IBM-932
Windows 3.1	*
Windows 95	*
Windows NT	*
OS/2 Version 3	CP00932
OS/2 Version 4	CP00932

Descriptor type	String returned
MVS description	*
IANA string	*
Developer Kit for Java	*
➤ iSeries Default Java Encoding	SJIS ⬅

Table 3. Sample Information from the API for CCSID 00437

Descriptor type	String returned
CCSID	00437
OS/400 Supported CCSID	00437
short text description	USA_PC-DATA
AIX 3.1	IBM-437
AIX 4.1	IBM-437
Windows 3.1	CP00437
Windows 95	CP00437
Windows NT	CP00437
OS/2 Version 3	CP00437
OS/2 Version 4	CP00437
MVS description	*
IANA string	IBM437
Developer Kit for Java	Cp437
➤ iSeries Default Java Encoding	ISO8859_1 ⬅

Table 4. Sample Information from the API for CCSID 01252

Descriptor type	String returned
CCSID	01252
OS/400 Supported CCSID	01252
short text description	MS-WIN_LATIN-1
AIX 3.1	*
AIX 4.1	*
Windows 3.1	CP01252
Windows 95	CP01252
Windows NT	CP01252
OS/2 Version 3	CP01252
OS/2 Version 4	CP01252
MVS description	*
IANA string	Windows-1252
Developer Kit for Java	Cp1252
➤ iSeries Default Java Encoding	ISO8859_1 ⬅

API introduced: V4R3

UniQueryCompChar()—Number of Composite Character Sequences API

Syntax

```
#include <UniChar.h>

int UniQueryCompChar(void *reserved,
                    const UniChar *ucs,
                    size_t *composite_count);
```

Service Program: QTQULSFN

Default Public Authority: *USE

Threadsafe: No

The **UniQueryCompChar**—Number of Composite Character Sequences function computes the number of composite character sequences in the code element array pointed to by *ucs*. The function sets *composite_count* to be the number of sequences that consist of a base character followed by zero or more combining characters up to, but not including, the null code element.

Parameters

reserved

INPUT; PTR(SPP)

The reserved parameter must be set to null.

ucs

INPUT; PTR(SPP)

The composite character sequence to be analyzed.

composite_count

OUTPUT; PTR(SPP)

This is set to the number of composite character sequences in the code element array pointed to by *ucs*.

Return Value

zero **UniQueryCompChar** function was successful. The value returned in *composite_count* is the number of sequences found in *ucs*.

non-zero

UniQueryCompChar was not successful and *composite_count* is set to zero.

Error Conditions

If **UniQueryCompChar** is not successful, the return value indicates one of the following errors:

UNI_BADOBJ

UniQueryCompChar function detected that the *reserved* parameter is not null.

UNI_INVALID

UniQueryCompChar function detected that *ucs* is a null pointer.

UNI_NODATA

UniQueryCompChar function detected that *ucs* points to a null code element.

UNI_COMP_FOUND

UniQueryCompChar function detected that *ucs* points to a combining character.

UNI_RANGE

UniQueryCompCharLen function detected that the value to be set in *element_count* was too large to fit.

API introduced: V3R6

Top | “National Language Support APIs,” on page 1 | APIs by category

Retrieve CCSID Data (QLGRTVCD)

Required Parameter Group:

1 Receiver variable

Output Char(*)

2 Length of receiver variable

Input Binary(4)

3 Request type

Input Binary(4)

4 Encoding Scheme

Input Binary(4)

5 Error code

I/O Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve CCSID Data (QLGRTVCD) API retrieves different subsets of CCSIDs based on the selection type. A typical use of this API would be to create a list of all the CCSIDs that are valid for a job.

Authorities and Locks

None.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The buffer that receives the CCSID information. Refer to the “Format of Receiver Variable” on page 17 for details about the format.

Length of receiver variable

INPUT; BINARY(4)

The size, in bytes, of the buffer that receives the CCSID information. The minimum size is 8.

Request type

INPUT; BINARY(4)

The type of information requested.

- 0 Retrieve all supported CCSIDs valid for this job.
- 1 Retrieve all supported CCSIDs valid for any job.
- 2 Retrieve all supported CCSIDs for the provided encoding scheme.
- 3 Retrieve the current default job CCSID value.

Encoding scheme

INPUT; BINARY(4)

The encoding scheme requested, if request type is 2. The parameter must be x'0000' if the request type is not 2. Some examples of supported encoding schemes are:

Single-byte data '1100'X, '2100'X, '3100'X, '4100'X, '4105'X, '4155'X, '5100'X, '5150'X, '6100'X
 Double-byte data '1200'X, '2200'X, '3200'X, '5200'X, '7200'X
 Mixed EBCDIC data '1301'X
 Mixed PC data '2300'X, '3300'X
 ISO-2022 data '5404'X
 EUC data '4403'X

See the Globalization topic for more information on CCSID values and encoding schemes.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

Format of Receiver Variable

The following table describes the information that is returned in the receiver variable. For a detailed description of each field, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	CCSIDs returned
4	4	BINARY(4)	CCSIDs available
		ARRAY (*)of BINARY(4)	Array of CCSIDs

Field Descriptions

Receiver variable. An array of 4-byte CCSID values that match the requested type.

CCSIDs available. The number of possible entries being returned. All available data is returned if enough space is provided.

CCSIDs returned. The actual number of CCSIDs that is being returned. If the data is truncated because the receiver variable is not large enough to hold all the data available, this value is less than the CCSIDs available.

Error Messages

Message ID	Error Message Text
CPF3BF9 E	Encoding scheme value not correct for requested type.
CPF3BFA E	Request type not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF2647 E	Buffer length not valid.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API Introduced: V4R4

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Retrieve CCSID Text (QLGRTVCT) API

Required Parameter Group:	
1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Format name
Input	Char(8)
4	CCSID request type
Input	Binary(4)
5	CCSID request type encoding scheme selection
Input	Char(2)
6	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: Yes	

The Retrieve CCSID Text (QLGRTVCT) API retrieves different subsets of CCSIDs' values and their descriptions, if available. A typical use of this API would be to retrieve all the CCSIDs that are valid for a job.

Authorities and Locks

None.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable to receive the requested information. You can specify the size of the area to be smaller than the format requested, provided you specify the length of the receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name

INPUT; CHAR(8)

The content and format of the information returned. The possible format name is:

» RTVT0100 « Basic CCSID information format. For details about the format, see “Format of the Generated Information.”

CCSID Request Type

INPUT; Binary(4)

The type of CCSID requested.

- 0 Retrieve all supported CCSIDs valid for this job.
- 1 Retrieve all supported CCSIDs valid for any job.
- 2 Retrieve all supported CCSIDs for the provided encoding scheme.
- 3 Retrieve the current default job CCSID value.

CCSID request type encoding scheme selection

INPUT; Char(2)

The encoding scheme requested. If request type is not 2, this parameter must be x'0000'.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Information

Following is the format of the information returned. For detailed descriptions of the fields in the table, see “Field Descriptions” on page 20.

» RTVT0100 Format

Offset		Type	Field «
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
12	C	BINARY(4)	CCSID value for descriptive texts
16	10	BINARY(4)	Offset to start of CCSID identifier array
20	14	BINARY(4)	Number of CCSIDs returned
24	18	BINARY(4)	Size of each CCSID entry
28	1D	CHAR(*)	Reserved

Format of CCSID Array

The following fields are repeated for each CCSID entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	CCSID identifier
4	4	BINARY(4)	Length of descriptive text
2	2	» CHAR(*) «	Descriptive text

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID value for descriptive text. The coded character set identifier (CCSID) in which all the descriptive texts are encoded.

Offset to start of CCSID array. The offset to the start of the CCSID array.

Number of CCSIDs retrieved. The number of CCSIDs in the list.

Size of each CCSID entry. The size in bytes for each CCSID entry.

CCSID identifier The CCSID value.

Length of Descriptive Text. The length (in bytes) of the descriptive text.

Descriptive text. The zero to » 95-byte « long descriptive text associated with the CCSID.

Reserved. An ignored field.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Sever error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V5R1

Top | "National Language Support APIs," on page 1 | APIs by category

Retrieve Country or Region Identifiers (QLGRTVCI) API

Required Parameter Group:	
1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Format name
Input	Char(8)
4	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: Yes	

The Retrieve Country or Region Identifiers (QLGRTVCI) API retrieves a list of country or region identifiers and their descriptions.

Authorities and Locks

None.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable to receive the requested information. You can specify the size of the area to be smaller than the format requested, provided you specify the length of the receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name

INPUT; CHAR(8)

The content and format of the information returned. The possible format names are:

RTVC0100 Basic country or region identifier format

For details about the formats, see “Format of the Generated Information” on page 22.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Information

The following table shows the format of the information returned. For a detailed description of each field, see "Field Descriptions."

RTVC0100 Format

Offset		Type	Field
Dec	Hex		
0	0	Binary(4)	Bytes returned
4	4	Binary(4)	Bytes available
12	C	Binary(4)	CCSID value of descriptive text
16	10	Binary(4)	Offset to start of country or region identifier array
8	8	Binary(4)	Number of country or region identifiers returned
20	14	CHAR(*)	Reserved
Note: Format of country or region identifier array. The following fields are repeated for each country or region identifier entry.			
0	0	CHAR(2)	Country or region identifier
2	2	CHAR(40)	Descriptive text

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID value of descriptive text. The coded character set identifier (CCSID) in which all the descriptive texts are encoded.

Descriptive text. The 40-byte descriptive text associated with the country or region identifier.

Country or region identifier. The two-character country or region identifier.

Number of country or region identifiers retrieved. The number of country or region identifiers in the list.

Offset to start of country or region identifier array. The offset to the start of the country or region identifier array.

Reserved. An ignored field.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Sever error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Retrieve Default CCSID (QLGRTVDC)

Required Parameter Group:	
1	Default CCSID
Output	Binary(4)
2	Language ID
Input	Char(3)
3	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: Yes	

The Retrieve Default CCSID (QLGRTVDC) API retrieves the default CCSID given a language ID.

Authorities and Locks

None.

Required Parameter Group

Default CCSID

OUTPUT; BINARY(4)

The default CCSID associated with the Language ID.

Language ID

INPUT; CHAR(3)

The language ID requested.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
------------	--------------------

CPF24B4 E	Severe error while addressing parameter list.
-----------	---

CPF3BCC E	Language identifier &1 not valid.
-----------	-----------------------------------

CPF3CF1 E	Error code parameter not valid.
-----------	---------------------------------

CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
-----------	--

API introduced: V5R2

Retrieve Language IDs (QLGRTVLI) API

Required Parameter Group:	
1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Format name
Input	Char(8)
4	Error code
I/O	Char(*)
	Default Public Authority: *EXCLUDE
Threadsafe: No	

The Retrieve Language Identifiers (QLGRTVLI) API retrieves a list of language identifiers and their descriptions.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The receiver variable that receives the information requested. You can specify the size of the area to be smaller than the format requested, provided you specify the length of receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Format name

INPUT; CHAR(8)

The content and format of the information returned. The possible format names are:

RTVL0100 Basic language identifier format

See “RTVL0100 Format” for a description of this format.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

RTVL0100 Format

Following is the format of the information returned. For a description of the fields in this format, see “Field Descriptions” on page 25.

Offset		Type	Field
Dec	Hex		
0	0	Binary(4)	Bytes available
4	4	Binary(4)	Bytes returned
8	8	Binary(4)	Number of language identifiers retrieved
12	B	Binary(4)	CCSID value of descriptive text
16	10	Binary(4)	Offset to start of language identifier array
20	14	Char(*)	Reserved
Note: Format of language identifier array. The following fields are repeated for each language identifier entry.			
0	0	Char(3)	Language Identifier
3	3	Char(40)	Descriptive text

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID value of descriptive text. The coded character set identifier (CCSID) that all the descriptive texts are encoded in.

Descriptive text. The 40-character descriptive text associated with the language identifier.

Language identifier. The 3-character language identifier.

Number of language identifiers retrieved. The number of language identifiers in the list.

Offset to start of language identifier array. The offset to the start of the language identifier array.

Reserved. An ignored field.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "National Language Support APIs," on page 1 | APIs by category

Retrieve Language Information (QLGRLNGI) API

Required Parameter Group:	
1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Language selection information format name
Input	Char(8)
4	Language selection information
Input	Char(*)
5	Output format name
Input	Char(8)
6	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Retrieve Language Information (QLGRLNGI) API returns a selected national language version (NLV) based on the specified product, option, and language identifier. This API can be used to determine the correct NLV directory to be used to locate translatable information.

Note: The format LNGR0100 returns CCSID information about an NLV. These values are the CCSIDs that are used when language translation is done for a given NLV, but do not imply whether or not a particular product or portion of a product has actually been translated. If runtime conversions are to be done on translatable data, they should be based on how the specific data is tagged.

Authorities and Locks

Product Availability Lock

*SHRRD

The product availability object resides in the QUSRSYS library.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable to receive the requested information.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable in bytes. The value specified must be at least 8.

Language selection information format name

INPUT; CHAR(8)

The content and format of the detailed input specification. The possible format names are:

- LNGI0100* Specifies basic information about which product and option to return information for and the language ID to use. For more information, see “LNGI0100 Format.”
- LNGI0110* Specifies basic information about which product and option to return information for and the language ID to use. This format will return the same information as format LNGI0100, but message CPF3BEE will not be issued if the primary language is not installed. For more information, see “LNGI0110 Format.”

Language selection information

INPUT; CHAR(*)

The detailed input specification information for this request.

Output format name

INPUT; CHAR(8)

The content and format of the information returned. The possible format names are:

- LNGR0100* Returns basic information about the national language version (NLV) that best matches the specified product, option, and language ID. For more information, see LNGR0100 Format (page 28).
- LNGR0200* Returns basic information about the national language version (NLV) that best matches the specified product, option, and language ID. It also returns information about the coded character set identifiers (CCSIDs) that are used to store the NLV information. For more information, see LNGR0200 Format (page 28).

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

LNGI0100 Format

Information passed in the language selection information parameter can be in the following format. For detailed descriptions of each field, see “Field Descriptions” on page 28.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(7)	Product ID
13	D	CHAR(4)	Product option
17	11	CHAR(10)	Language ID

LNGI0110 Format

Information passed in the language selection information parameter can be in the following format. For detailed descriptions of each field, see “Field Descriptions” on page 28.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(7)	Product ID
13	D	CHAR(4)	Product option
17	11	CHAR(10)	Language ID

Field Descriptions

Product ID. The product ID for which information is being requested. You can use this special value for the product ID:

**OPSYS* The product ID for the operating system for the current release.

Product option. The option number for which information is being requested. Use 0000 for the base option. Valid values are 0000 through 0099, where each character is a digit.

Language ID. The 3-character language ID that should be used to locate the correct NLV. Valid values can be found by prompting the Create User Profile (CRTUSRPRF) command and looking at the LANGID parameter. You can use these special values:

**CURUSR* Extract the language ID from the user profile that is currently in effect for the requesting job.

**SYSVAL* Extract the language ID from the QLANGID system value.

Format of the Returned Information

Information that is returned in the receiver variable parameter can be in the following format. For detailed descriptions of the fields for each format, see "Field Descriptions" on page 29.

If the product load is not known to the system, an error (CPF0CAF) will occur. If the product load is known to the system, but the specified product option is not installed, an error (CPF3BDF) will occur. If the product and option are known and installed, but there is no primary language installed for it, an error (CPF3BEE) will occur.

LNGR0100 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Selected NLV
12	C	CHAR(4)	NLV of primary language
16	10	CHAR(3)	Language ID

LNGR0200 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Selected NLV
12	C	CHAR(4)	NLV of primary language
16	10	BINARY(4)	EBCDIC CCSID of selected NLV
20	14	BINARY(4)	PC ASCII CCSID of selected NLV
24	18	BINARY(4)	ISO ASCII CCSID of selected NLV
28	1C	CHAR(3)	Language ID

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes returned to the user. This is the lesser of the number of bytes available and the length of the receiver variable.

EBCDIC CCSID of selected NLV. The CCSID that is used to store translated, EBCDIC textual data for the selected NLV.

ISO ASCII CCSID of selected NLV. The CCSID that is used to store translated, Internet-ASCII textual data for the selected NLV.

Language ID. The language identifier that is used in the determination of NLV. This will either be the specified value, the value from the current user profile, or the value from the QLANGID system value.

NLV of primary language. The NLV that is associated with the primary language installed for the specified product and option. If language selection information format LNGI0110 is specified, this field may contain blanks.

PC ASCII CCSID of selected NLV. The CCSID that is used to store translated, PC-ASCII textual data for the selected NLV.

Selected NLV. The NLV chosen as a result of the specified or determined language ID, and what NLVs are installed for the specified product and option. This will be the same as the NLV of the primary language if the language ID maps to an NLV that is not installed for the product.

Error Messages

Message ID	Error Message Text
CPF0C4A E	Product record not found.
CPF0C4B E	Product availability object &2/&1 recovery required.
CPF0C4C E	Cannot allocate object &1 in library &2.
CPF0C4D E	Error occurred while processing object &1 in library &2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3BEE E	Primary language not installed.
CPF3BDF E	CCSID Product not installed.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V4R1

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Retrieve Locale Information (QLGRTVLC, QlgRetrieveLocaleInformation) API

Required Parameter Group:

1 Receiver variable

Output Char(*)

2 Length of receiver variable

Input Binary(4)

3 Format

Input Char(8)

4 Locale path name

Input Char(*)

5 Locale category key to return

Input Binary(4)

6 Error code

I/O Char(*)

Service Program: QLGLOCAL

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve Locale Information (OPM, QLGRTVLC; ILE, QlgRetrieveLocaleInformation) API retrieves information from one category of a locale or retrieves all categories of a locale. For further information on locales, see the Globalization topic in the iSeries Information Center.

Authorizations and Locks

Shipped API Authority

*USE

Locale Object Authority

*USE

Locale Library Authority

*USE


Locale Object Lock

*SHRNUP

Required Parameters

Receiver variable

OUTPUT; CHAR(*)

The variable that is to receive the information requested. If requesting the LC_CTYPE or the LC_COLLATE categories, the information returned could potentially be very large if the locale data type indicator (returned within the receiver variable) indicates that the locale data is mixed-byte, double-byte (includes ISO 10646 16-bit character encoding standard (UCS2)), 

UTF8 or UTF32. ❄ In this case, a user space with the auto-extendable attribute set may provide a better means for the receiver variable. For further information on using a user space for a receiver variable, see Receiver variables in API concepts.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format

INPUT; CHAR(8)

The format of the locale information to be returned. The following format name must be used:

LOCI0100 Locale information

Refer to “Format of the Locale Information” on page 33 for more information on the locale format.

Locale path name

INPUT; CHAR(*)

The structure that contains the path name information of the locale used to retrieve the information. For more information on this structure, see Path name format.

The following special values are allowed for the path name field in the path name structure:

<i>*C</i>	Use the C locale. This special value must be specified in the first 2 characters of the path name field of the path name structure. The C locale is equivalent to the POSIX locale.
<i>*ENV</i>	Use the locale defined in the environment of the job. This special value must be specified in the first 4 characters of the path name field of the path name structure.
<i>*POSIX</i>	Use the POSIX locale, which is designed to meet the X/Open POSIX locale specification. This special value must be specified in the first 6 characters of the path name field of the path name structure. The POSIX locale is equivalent to the C locale.

When **ENV* is specified for the locale path name, the API will retrieve the locale information from the environment variable. The API will assume the '/' character as the path name delimiter. If the CCSID for the path name in the environment variable is 65535, the API will assume the job default CCSID.

If **ENV* is specified and the requested category is not *LC_ALL*, the API will use the locale defined in the environment variable as defined by the first condition met below:

- Check the *LC_ALL* environment variable. If defined and not null, use the locale specified. Otherwise, go to the next step.
- Check the specific category environment variable (for example *LC_CTYPE*, *LC_MONETARY*, and so on). If defined and not null, use the locale specified. Otherwise, go to the next step.
- Check the *LANG* environment variable. If defined and not null, use the locale specified. Otherwise, the API returns an error.

If **ENV* is specified and the requested category is *LC_ALL*, the API uses the locale defined in the environment variable as defined by the first condition met below:

- Check the *LC_ALL* environment variable. If defined and not null, use the locale specified for all locale categories. Otherwise, go to the next step.
- Check the specific category environment variable (for example, *LC_CTYPE*, *LC_MONETARY*, and so on) for each category. If defined and not null, use the locale specified for the corresponding locale category. This could result in the locale information for each category being retrieved from a different locale object. Otherwise, go to the next step.

- Check the LANG environment variable. If defined and not null, use the locale specified. Otherwise, the API returns an error.

If none of the locale environment variables are defined for the requested category or if the locale defined is found to be not valid or does not exist, an error is returned.

Locale category key to return

INPUT; BINARY(4)

The key of the locale category to retrieve information on. Refer to “Locale Category Keys” for more information.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

Locale Category Keys

The following table lists the valid locale category keys for the Locale category key to return parameter. For a detailed description of each field, see “Locale Category Key Descriptions.”

Key	Field
-1	LC_ALL - All categories
1	LC_CTYPE - Character classification and conversion
2	LC_COLLATE - Character collation
3	LC_TIME - Date and time formatting
4	LC_NUMERIC - Numeric and non-monetary formatting
5	LC_MESSAGES - Affirmative and negative responses
6	LC_MONETARY - Monetary related numeric formatting
7	LC_TOD - Time zone information

Locale Category Key Descriptions

LC_ALL - All categories. Return all categories in the locale.

LC_COLLATE - Character collation. Provides a collation sequence definition for characters and character strings. A collation sequence defines the relative order between the collating elements in the locale.

LC_CTYPE - Character classification and conversion. Defines character classification, conversion and other character attributes such as printable and punctuation characters.

LC_MESSAGES - Affirmative and negative responses. Defines the format and values for affirmative and negative responses.

LC_MONETARY - Monetary related numeric formatting. Defines the rules and symbols that are used to format monetary numeric information.

LC_NUMERIC - Numeric and non-monetary formatting. Defines the rules and symbols that are used to format non-monetary numeric information.

LC_TIME - Date and time formatting. Defines the interpretation for date and time formatting. This category supports the Gregorian (7-day weeks, 12-month years, leap years, and so forth) style calendar only.

LC_TOD - Time zone information. Defines the time zone information (name of the time zone, daylight savings time start and end, and so forth).

Format of the Locale Information

LOCI0100 Format



The following information is returned in the receiver variable. For a detailed description of each field, see Path name format. For a detailed description of each locale category, refer to the locale category formats.

Upon return from the call to the API, it is highly recommended that the bytes returned and bytes available fields be examined first. If the bytes available field is greater than the bytes returned field, the API had more information to return than what could fit in the receiver variable provided on the call. When this occurs, calling the API again with a receiver variable at least the size of the value in the bytes available field is highly recommended. This is due to the wide use of offsets by this API. If not enough room was available, an offset may contain a value of where the information would have started if the receiver variable were large enough. Attempting to access the information at this offset may cause unpredictable results.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Locale category key
12	C	BINARY(4)	Offset to category information. Refer to “Locale Category Formats” for further details.
16	10	BINARY(4)	Length of category information
20	14	CHAR(*)	Reserved

Locale Category Formats

A format is defined for each locale category. An offset and length field are provided for all fields that are variable in length. The variable length fields may contain character data, binary data or a combination of character and binary data. The type of data that the field contains is described in the field descriptions.

The locale data type is used to determine if the locale data returned contains single-byte data, mixed-byte data (single-byte and double-byte mixed), pure double-byte data,  UTF8 data or UTF32 data.  If the data returned is EBCDIC mixed-byte data, the double-byte data is encased within a shift out (X'0E') and shift in (X'0F'), except when the single-byte and double-byte data is returned separately (for example, LC_CTYPE).

If there is no data available for the field and the field type is an offset to the information, the offset is set to a value of zero.

LC_CTYPE - Character Classification and Conversion Format

The format for which this information is returned is determined by the locale data type. For example, if the locale data type is 1, the information returned for this category would be in the Single-Byte Character Classification Structure. Refer to for information on the possible data types for the locale data type field. For a detailed description of each field, see Path name format.

The explanations for the various arrays that are returned by the LC_CTYPE category all assume base zero.

Single-Byte Character Classification and Conversion Structure

The following information is returned for the single-byte character classification and conversion locale category.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		CHAR(256)	Single-byte uppercase conversion mapping. Refer to the "Single-Byte Conversion Mapping Structures" on page 35 for further information on this field.
		CHAR(256)	Single-byte lowercase conversion mapping. Refer to the "Single-Byte Conversion Mapping Structures" on page 35 for further information on this field.
		ARRAY(256) of UNSIGNED BINARY(2)	Single-byte character classification. Refer to the "Single-Byte Character Classification Structure" on page 39 for further information on this field.
		CHAR(*)	Reserved

Double-Byte Character Classification Structure

The following information is returned for the double-byte character classification and conversion locale category.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		BINARY(4)	Offset to double-byte uppercase conversion mapping. Refer to the "Double-Byte Conversion Mapping Structures" on page 36 for further information on this field.
		BINARY(4)	Offset to double-byte lowercase conversion mapping. Refer to the "Double-Byte Conversion Mapping Structures" on page 36 for further information on this field.
		BINARY(4)	Offset to double-byte character classification. Refer to the "Double-Byte Character Classification Structure" on page 41 for further information on this field.
		CHAR(*)	Reserved

Mixed-Byte Character Classification and Conversion Structure

The following information is returned for the mixed-byte character classification and conversion category. The shift out and shift in control characters are not returned with this data.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type

Offset		Type	Field
Dec	Hex		
		CHAR(256)	Single-byte uppercase conversion mapping. Refer to the “Single-Byte Conversion Mapping Structures” for further information on this field.
		CHAR(256)	Single-byte lowercase conversion mapping. Refer to the “Single-Byte Conversion Mapping Structures” for further information on this field.
		ARRAY(256) of UNSIGNED BINARY(2)	Single-byte character classification. Refer to the “Single-Byte Character Classification Structure” on page 39 for further information on this field.
		BINARY(4)	Offset to double-byte uppercase conversion mapping. Refer to the “Double-Byte Conversion Mapping Structures” on page 36 for further information on this field.
		BINARY(4)	Offset to double-byte lowercase conversion mapping. Refer to the “Double-Byte Conversion Mapping Structures” on page 36 for further information on this field.
		BINARY(4)	Offset to double-byte character classification. Refer to the “Double-Byte Character Classification Structure” on page 41 for further information on this field.
		CHAR(*)	Reserved



Four-Byte Character Classification Structure

The following information is returned for the four-byte character classification and conversion locale category.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		BINARY(4)	Offset to four-byte uppercase conversion mapping. Refer to the “Four-Byte Conversion Mapping Structures” on page 37 for further information on this field.
		BINARY(4)	Offset to four-byte lowercase conversion mapping. Refer to the “Four-Byte Conversion Mapping Structures” on page 37 for further information on this field.
		BINARY(4)	Offset to four-byte character classification. Refer to the “Four-Byte Character Classification Structure” on page 42 for further information on this field.
		CHAR(*)	Reserved



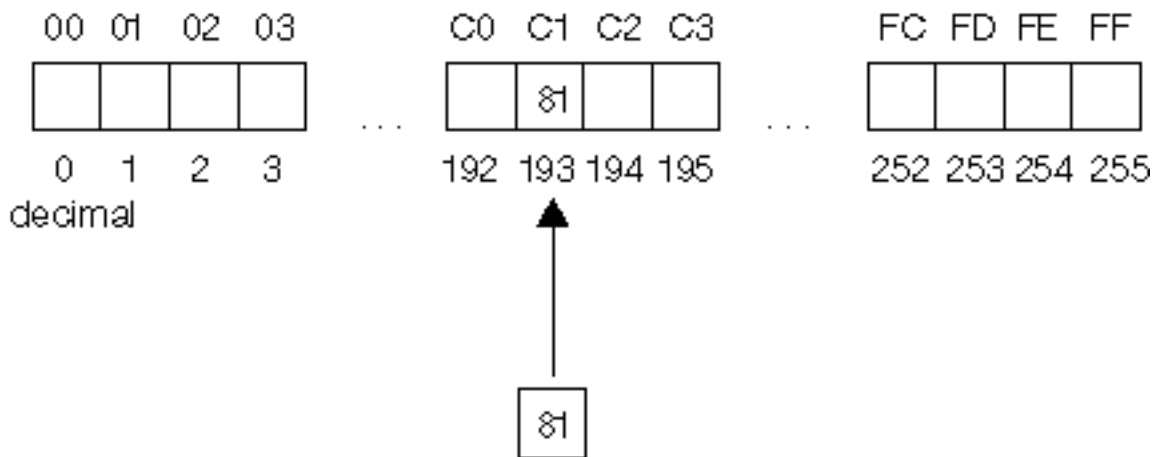
Single-Byte Conversion Mapping Structures

The conversion information for the single-byte characters is stored within an Array(256) of Char(1).

As illustrated in the example below, the conversion is performed by using the code point for the character that is to be converted as the subscript into the conversion mapping array. For example, if you have an uppercase character that has a code point (hexadecimal value) of 'C1', using the decimal value of this code point which is 193, you would go to the 193rd subscript of the lowercase conversion array and the value that is stored in that subscript is the code point for the lower case character. In this example, the lower case code point for uppercase character C1 is 81.

Conversion from lower to upper would be performed the same way except the code point for the lower case character would be used as the subscript into the uppercase conversion array.

hexadecimal



Double-Byte Conversion Mapping Structures

The conversion information for the double-byte characters is stored within an Array(256) of Binary(4) in which the Binary(4) elements of this array contain offsets to a second Array(256) of Character(2). The Character(2) elements of the second array contain the upper or lowercase character conversion mappings.

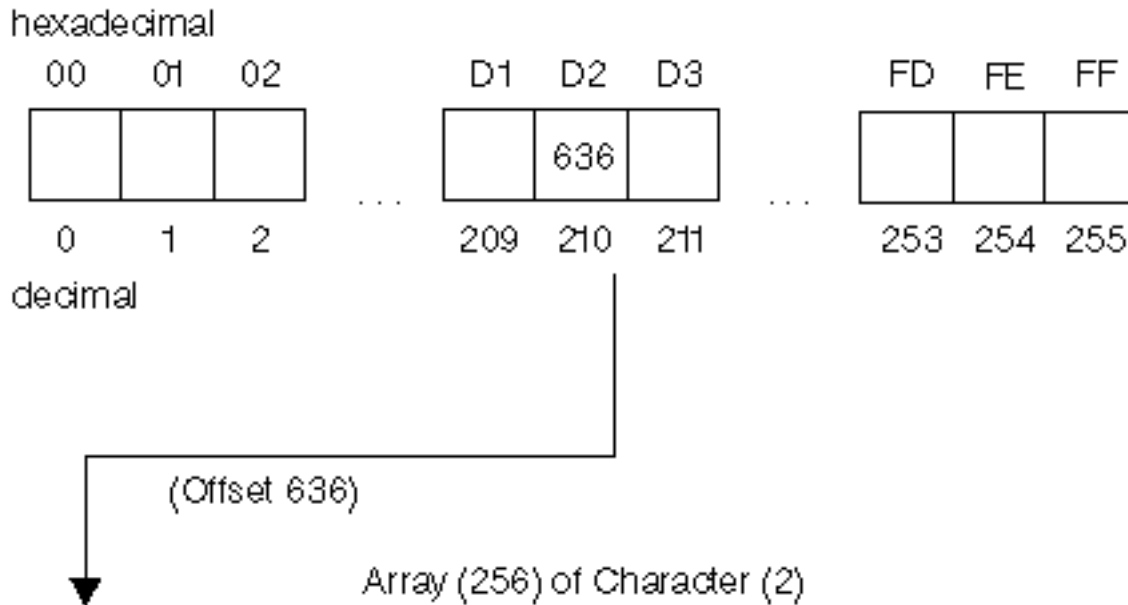
The double-byte code point for the character is used as the subscripts to access the arrays. The left-most byte of the double-byte character is used to access the first Array(256) of Binary(4). Within the subscripts of the first array are offsets to the second Array(256) of Character(2). The right-most byte of the double-byte character is used as the subscript to access this second array. Within this subscript is the double-byte upper or lowercase conversion character.

If the offset in the Binary(4) element of the first array is zero, then there is no case conversion available for that range of characters. For example, if you have an uppercase double-byte code point '40CE' and the offset in the first array for the lowercase conversion mapping was set to zero, this would indicate that all uppercase double-byte characters for this locale in the range where the left-most byte is '40' have no lowercase conversion available.

As illustrated in the example below, if you had a double-byte uppercase character 'D2AE', the decimal value of 'D2' is used as the subscript for the first array of the lowercase conversion mapping. The offset of 636 at this subscript is used to gain access to the second array and the decimal value of 'AE' is used as the subscript for the second array to access the double-byte lowercase character. In this example, the lowercase character for uppercase character 'D2AE' is 'B3CD'.

Conversion from lower to upper would be performed the same way except the code point for the lowercase character would be used as the subscripts into the uppercase conversion arrays.

Array (256) of Binary(4)



Four-Byte Conversion Mapping Structures

The conversion information for the four-byte characters is stored within an `Array(256) of Binary(4)`. The `Binary(4)` elements of this array contain offsets to a second `Array(256) of Binary(4)`. The `Binary(4)` elements of this second array contain offsets to a third `Array(256) of Binary(4)`. The `Binary(4)` elements of this third array contain offsets to a fourth `Array(256) of Binary(4)`. The `Binary(4)` elements of the fourth array contain the upper or lowercase character conversion mappings.

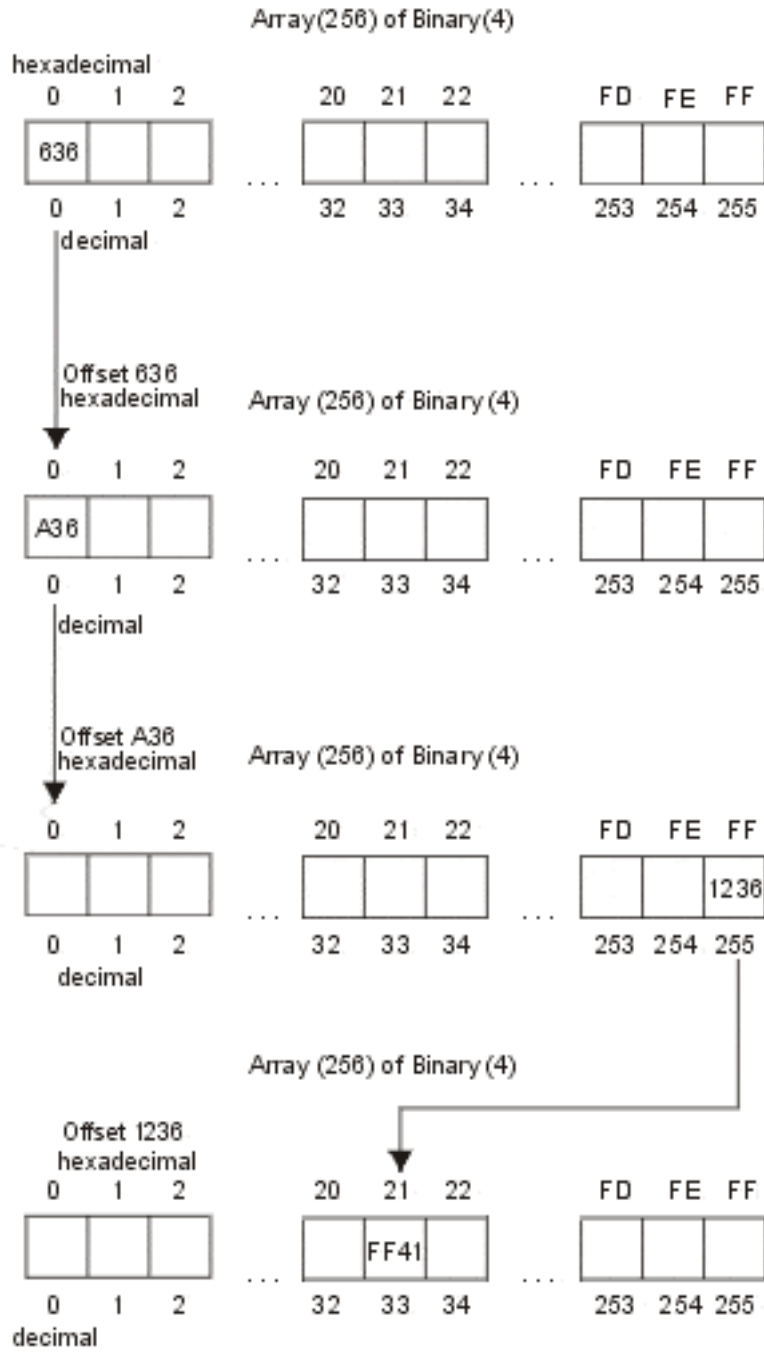
The four-byte code point for the character is used as the subscripts to access the arrays. The left-most byte of the four-byte character is used to access the first `Array(256) of Binary(4)`. Within the subscripts of the first array are offsets to the second `Array(256) of Binary(4)`. The second left-most byte of the four-byte

character is used as the subscript to access this second array. Within the subscripts of the second array are offsets to the third Array(256) of Binary(4). The third left-most byte of the four-byte character is used as the subscript to access this third array. Within the subscripts of the third array are offsets to the fourth Array(256) of Binary(4). The right-most byte of the four-byte character is used as the subscript to access this fourth array. The Binary(4) elements of the fourth array contain the upper or lowercase character conversion mappings.

If the offset in the Binary(4) element of the first array is zero, then there is no case conversion available for that range of characters. For example, if you have an uppercase four-byte code point '000040CE' and the offset in the first array for the lowercase conversion mapping was set to zero, this would indicate that all uppercase four-byte characters for this locale in the range where the left-most byte is '00' have no lowercase conversion available. The same is also true when the offset in the second or third array is zero.

As illustrated in the example below, if you had a four-byte uppercase character '0000FF21', the value of '00' is used as the subscript for the first array of the lowercase conversion mapping. The offset of 636 at this subscript is used to gain access to the second array. The value of '00' is used as the subscript for the second array to access the third array. The offset of A36 at this subscript is used to gain access to the third array and the value of 'FF' is used as the subscript for the third array to access the fourth array. The offset of 1236 at this subscript is used to gain access to the fourth array and the value of '21' is used as the subscript for the fourth array to access the four-byte lowercase character. In this example, the lowercase character for uppercase character '0000FF21' is '0000FF41'.

Conversion from lower to upper would be performed the same way except the code point for the lowercase character would be used as the subscripts into the uppercase conversion arrays.

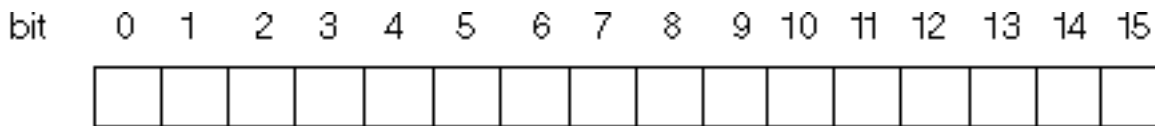


RBAFX663-0

Single-Byte Character Classification Structure

The character classification information for the single-byte character is stored within 2 bytes (16 bits) with one element of the Array(256) for each single-byte character. The decimal value for the hexadecimal code point for the single-byte character is used as the subscript for the array.

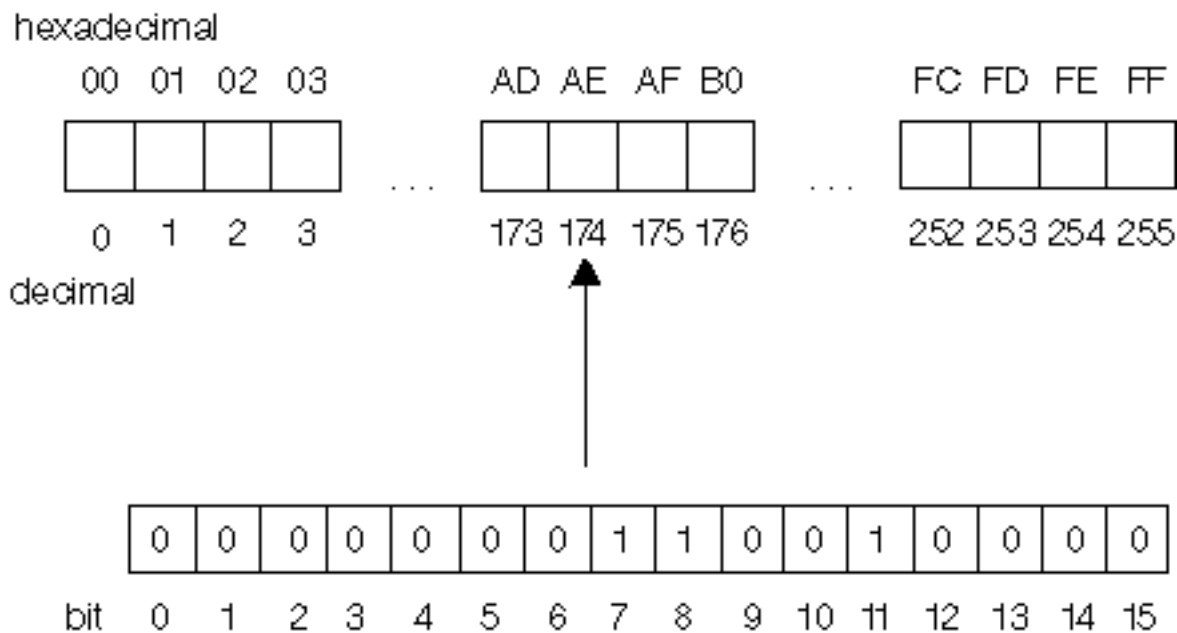
Each bit represents a character classification. If the value of the bit is set to 1, the character is of that classification. If set to 0, the character is not of that classification. The bit representations are as follows:



Bit	Character Classification
0-4	Undefined
5	Punctuation - characters classified as punctuation characters
6	Graph - characters classified as printable characters, not including the space character
7	Alphabetic - characters classified as letters
8	Upper - characters classified as uppercase letters
9	Lower - characters classified as lowercase letters
10	Control - characters to be classified as control characters
11	Print - characters classified as printable characters, including the space character
12	Blank - characters classified as blank characters
13	Space - characters classified as white-space characters
14	Digit - characters classified as numeric digits
15	Hexadecimal digit - characters classified as hexadecimal digits

As illustrated in the example below, the conversion is performed by using the code point for the single-byte character as the subscript into the array. For example, if you have a single-byte character that has a code point of 'AE', use the decimal value of this code point as the subscript into the array. The Binary(2) element of the array contains the 16 character classification bits. This example shows that this character is of the print, upper, and alphabetic character classifications.

Array (256) of Binary (2)



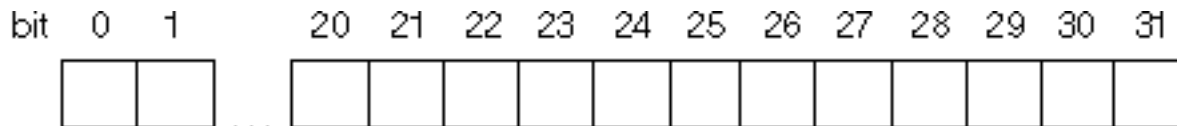
Double-Byte Character Classification Structure

The character classification information for the double-byte characters are stored within an Array(256) of Binary(4) in which the Binary(4) elements of this array contain offsets to a second Array(256) of Binary(4). The Binary(4) elements of the second array contain the 4 bytes (32 bits) that hold the bit representation for the double-byte character classification.

If the offset in the first array is set to zero, this double-byte character range is not defined. For example, if you have a double-byte code point 'CDFE' and the offset for the first array is set to zero, this would indicate that all double-byte characters for this locale in the range where the left-most byte was 'CD' have no character classification available.

The double-byte code point for the character is used as the subscripts to access the array. The left-most byte of the double-byte character is used to access the first Array(256) of Binary(4). Within the subscripts of the first array are offsets to the second Array(256) of Binary(4). The right-most byte of the double-byte character is used as the subscript to access this second array. Within this subscript are the 32 bits that contain the character classification information.

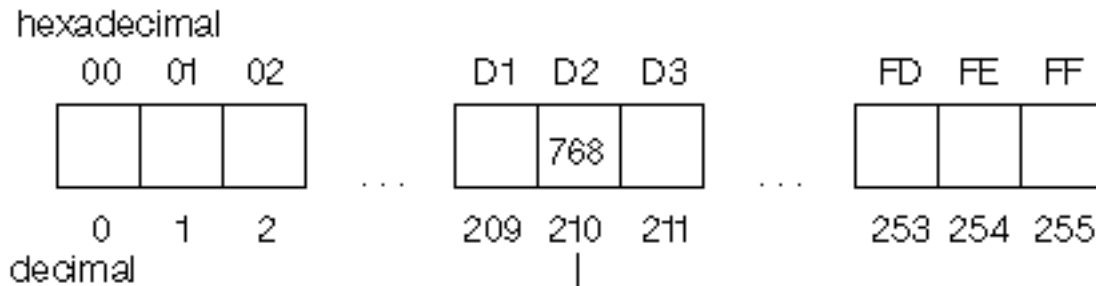
Each bit represents a character classification. If the value of the bit is set to 1, the character is of that classification. If set to 0, the character is not of that classification. The bit representations are as follows:



Bit	Character Classification
0-20	Undefined
21	Punctuation - characters classified as punctuation characters
22	Graph - characters classified as printable characters, not including the space character
23	Alphabetic - characters classified as letters
24	Upper - characters classified as uppercase letters
25	Lower - characters classified as lowercase letters
26	Control - characters to be classified as control characters
27	Print - characters classified as printable characters, including the space character
28	Blank - characters classified as blank characters
29	Space - characters classified as white-space characters
30	Digit - characters classified as numeric digits
31	Hexadecimal digit - characters classified as hexadecimal digits

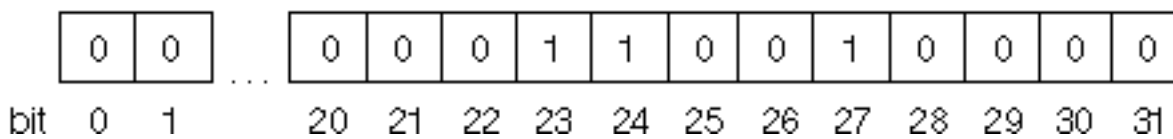
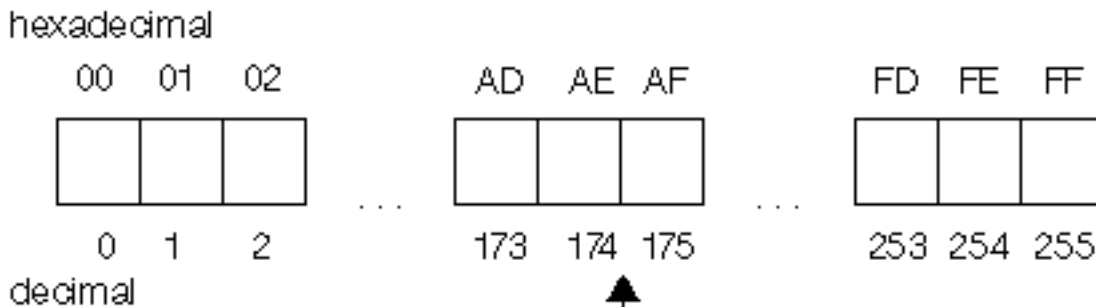
As illustrated in the example below, if you had a double-byte character 'D2AE', the decimal value of 'D2' is used as the subscript for the first array. The offset of 768 at this subscript is used to gain access to the second array. The decimal value of 'AE' is then used as the subscript for the second array. At this subscript are the 32 bits that contain the character classification information for the double-byte character. This example shows that this character is of the print, upper, and alphabetic character classifications.

Array (256) of Binary(4)



(Offset 768)

Array (256) of Binary (4)



Four-Byte Character Classification Structure

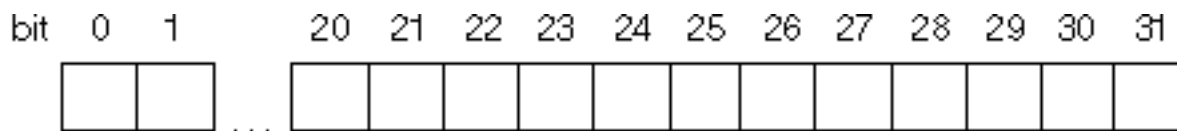
The character classification information for the four-byte characters is stored within an Array(256) of Binary(4). The Binary(4) elements of this array contain offsets to a second Array(256) of Binary(4). The Binary(4) elements of this second array contain offsets to a third Array(256) of Binary(4). The Binary(4) elements of this third array contain offsets to a fourth Array(256) of Binary(4). The Binary(4) elements of the fourth array contain the 4 bytes (32 bits) that hold the bit representation for the four-byte character classification.

If the offset in the Binary(4) element of the first array is zero, then this four-byte character range is not defined. For example, if you have an four-byte code point '000040CE' and the offset in the first array was

set to zero, this would indicate that all four-byte characters for this locale in the range where the left-most byte is '00' have no character classification available. The same is also true when the offset in the second or third array is zero.

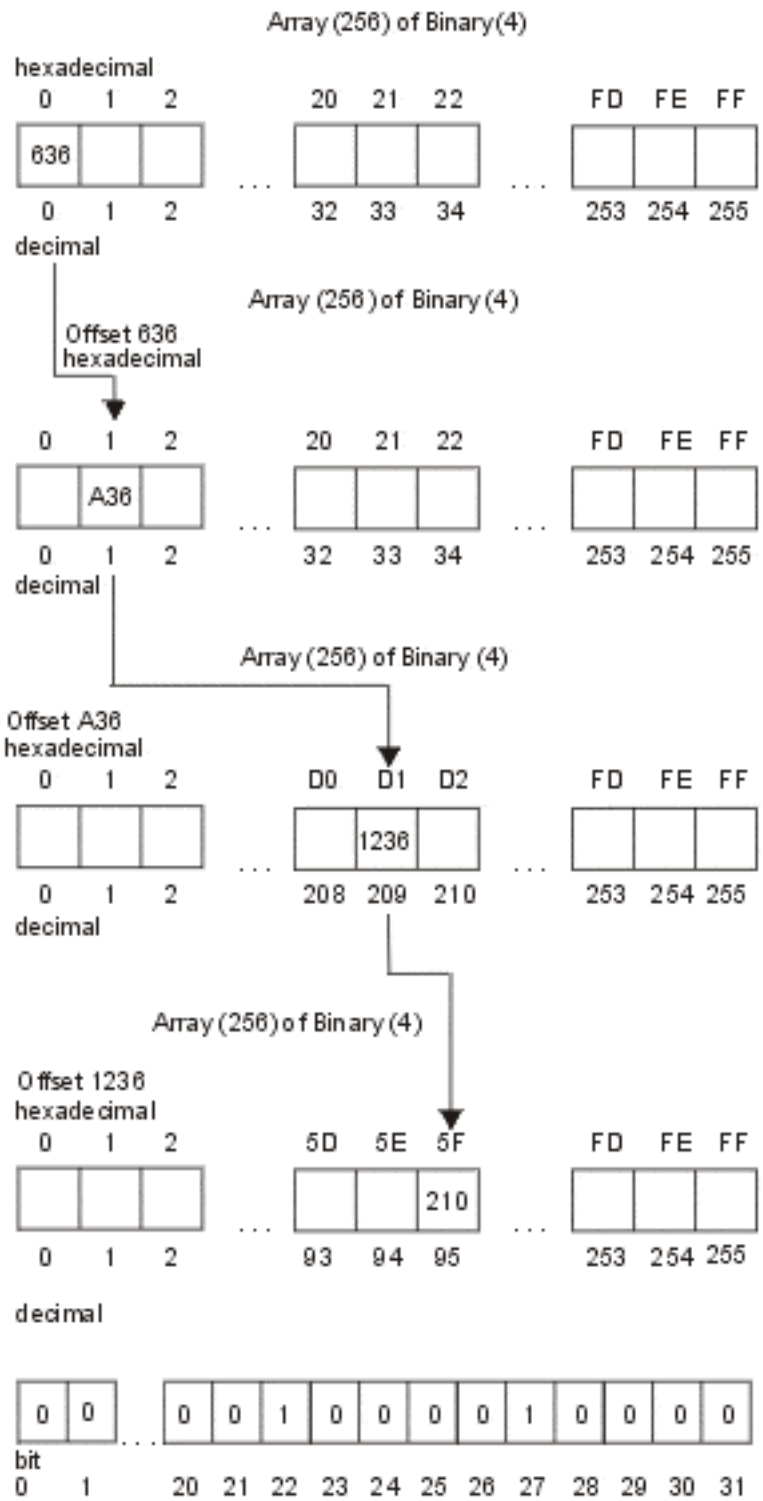
The four-byte code point for the character is used as the subscripts to access the arrays. The left-most byte of the four-byte character is used to access the first Array(256) of Binary(4). Within the subscripts of the first array are offsets to the second Array(256) of Binary(4). The second left-most byte of the four-byte character is used as the subscript to access this second array. Within the subscripts of the second array are offsets to the third Array(256) of Binary(4). The third left-most byte of the four-byte character is used as the subscript to access this third array. Within the subscripts of the third array are offsets to the fourth Array(256) of Binary(4). The right-most byte of the four-byte character is used as the subscript to access this fourth array. Within this subscript are the 32 bits that contain the character classification information.

Each bit represents a character classification. If the value of the bit is set to 1, the character is of that classification. If set to 0, the character is not of that classification. The bit representations are as follows:



Bit	Character Classification
0-20	Undefined
21	Punctuation - characters classified as punctuation characters
22	Graph - characters classified as printable characters, not including the space character
23	Alphabetic - characters classified as letters
24	Upper - characters classified as uppercase letters
25	Lower - characters classified as lowercase letters
26	Control - characters to be classified as control characters
27	Print - characters classified as printable characters, including the space character
28	Blank - characters classified as blank characters
29	Space - characters classified as white-space characters
30	Digit - characters classified as numeric digits
31	Hexadecimal digit - characters classified as hexadecimal digits

As illustrated in the example below, if you had a four-byte character '0001D15F', the value of '00' is used as the subscript for the first array of the character classification information. The offset of 636 at this subscript is used to gain access to the second array. The value of '01' is used as the subscript for the second array to access the third array. The offset of A36 at this subscript is used to gain access to the third array and the value of 'D1' is used as the subscript for the third array to access the fourth array. The offset of 1236 at this subscript is used to gain access to the fourth array and the value of '5F' is used as the subscript to access the fourth array. Within this subscript are the 32 bits that contain the character classification information.



RBAFX664-0



LC_COLLATE - Character Collation Format

The format for which this information is returned is determined by the locale data type. For example, if the locale data type is 1, the information returned for this category would be in the Single-byte Character Collation Structure. For a detailed description of each field, see Path Name Format.

The explanations for the various arrays that are returned by the LC_COLLATE category all assume base zero.

Single-Byte Character Collation Structure

The following information is returned for the single-byte character collation structure.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data.
		BINARY(4)	Locale data type.
		BINARY(4)	Number of weights and sort rules.
		BINARY(4)	Offset to sort rules. Refer to the "Character Collation Sort Rules" on page 47 for further information on this field.
		BINARY(4)	Offset to single-byte character collating table. Refer to the "Single-byte Character Collating Table" on page 48 for further information on this field.
		BINARY(4)	Offset to multi-character collating elements. Refer to the "Multi-Character Collating Elements" on page 52 for further information on this field.
		BINARY(4)	Offset to single-byte one-to-many character mappings. Refer to the "One-to-Many Character Mappings" on page 52 for further information on this field.
		CHAR(*)	Reserved

Double-Byte Character Collation Structure

The following information is returned for the double-byte character collation structure.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data.
		BINARY(4)	Locale data type.
		BINARY(4)	Number of weights and sort rules.
		BINARY(4)	Offset to sort rules. Refer to the "Character Collation Sort Rules" on page 47 for further information on this field.
		BINARY(4)	Offset to double-byte character collating table. Refer to the "Double-byte Character Collating Table" on page 48 for further information on this field.
		BINARY(4)	Offset to multi-character collating elements. Refer to the "Multi-Character Collating Elements" on page 52 for further information on this field.
		BINARY(4)	Offset to double-byte one-to-many character mappings. Refer to the "One-to-Many Character Mappings" on page 52 for further information on this field.
		CHAR(*)	Reserved

Mixed-Byte Character Collation Structure

The following information is returned for the mixed-byte character collation structure. If the data returned is EBCDIC mixed-byte, the shift out and shift in control characters are only returned for the multi-character collating elements.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data.
		BINARY(4)	Locale data type.
		BINARY(4)	Number of weights and sort rules.
		BINARY(4)	Offset to sort rules. Refer to the "Character Collation Sort Rules" on page 47 for further information on this field.
		BINARY(4)	Offset to single-byte character collating table. Refer to the "Single-byte Character Collating Table" on page 48 for further information on this field.
		BINARY(4)	Offset to double-byte character collating table. Refer to the "Double-byte Character Collating Table" on page 48 for further information on this field.
		BINARY(4)	Offset to multi-character collating elements. Refer to the "Multi-Character Collating Elements" on page 52 for further information on this field.
		BINARY(4)	Offset to single-byte one-to-many character mappings. Refer to the "One-to-Many Character Mappings" on page 52 for further information on this field.
		BINARY(4)	Offset to double-byte one-to-many character mappings. Refer to the "One-to-Many Character Mappings" on page 52 for further information on this field.
		CHAR(*)	Reserved



Four-Byte Character Collation Structure

The following information is returned for the four-byte character collation structure.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data.
		BINARY(4)	Locale data type.
		BINARY(4)	Number of weights and sort rules.
		BINARY(4)	Offset to sort rules. Refer to the "Character Collation Sort Rules" on page 47 for further information on this field.
		BINARY(4)	Offset to four-byte character collating table. Refer to the "Four-byte Character Collating Table" on page 50 for further information on this field.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Offset to multi-character collating elements. Refer to the “Multi-Character Collating Elements” on page 52 for further information on this field.
		BINARY(4)	Offset to four-byte one-to-many character mappings. Refer to the “One-to-Many Character Mappings” on page 52 for further information on this field.
		CHAR(*)	Reserved






Character Collation Sort Rules

The sort rules are returned as an Array(*) of Binary(4) where the number of Binary(4) elements (or sort rules) is determined by the Number of weights and sort rules field. Each Binary(4) element is a sort rule. The sort rules are applied when comparing strings and they apply to all the collating element tables returned. The first sort rule is applied when comparing strings using the first weight; the second sort rule is applied when comparing strings using the second weight, and so on. The possible sort rules are:

- 1 Forward - specifies that comparison operations for the weight level proceed from the start of the string towards the end of the string.
- 2 Backward - specifies that comparison operations for the weight level proceed from the end of the string towards the beginning of the string.
- 8 Position - specifies that comparison operations for the weight level will consider the relative position of elements in the strings not subject to ignore. Refer to “Orders and Weights” for further information on the ignore value.
- 9 Forward/Position - specifies that comparison operations for the weight level proceed from the start of the string towards the end of the string and that comparison operations for the weight level will consider the relative position of elements in the strings not subject to ignore. Refer to “Orders and Weights” for further information on the ignore value.
- 10 Backward/Position - specifies that comparison operations for the weight level proceed from the end of the string towards the beginning of the string and that comparison operations for the weight level will consider the relative position of elements in the strings not subject to ignore. Refer to “Orders and Weights” for further information on the ignore value.

Orders and Weights

The order and weights are returned as an Array(*) of Binary(4). The first element in the array specifies the order for the character or multi-character. The remaining elements are the weights for the character or multi-character. The number of remaining elements is determined by the Number of weights and sort rules field. In other words, the total number of elements in the array is 1 + Number of weights and sort rules. Refer to “Single-byte Character Collating Table” on page 48  or “Double-byte Character Collating Table” on page 48  or “Four-byte Character Collating Table” on page 50  for further information on the how the orders and weights are returned.

Each collating element is assigned a collation value (order) that defines its order in the character collation sequence in the locale. This order is used by regular expressions and pattern matching, and if no collation weight is explicitly defined (Number of weights and sort rules field equals zero), it also serves as the collation weight for sorting.

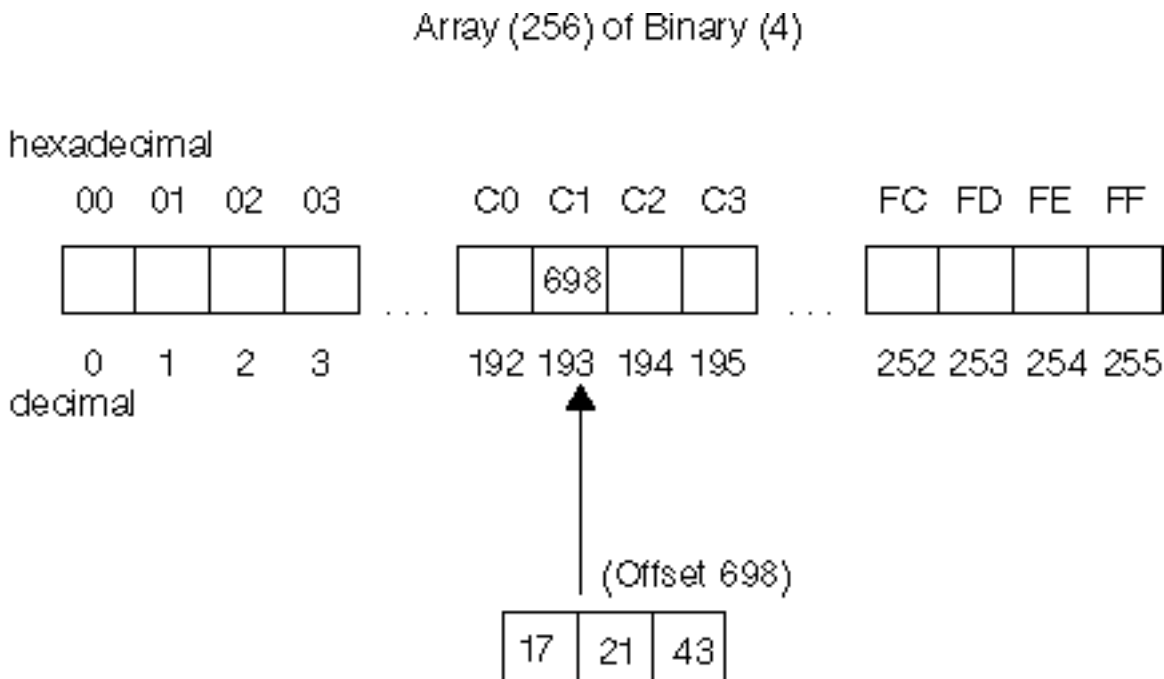
The collation weights are used when comparing two strings to determine their relative order. The following special values are possible for the weight:

- 1 Indicates that when strings are compared using weights at the level where the -1 is specified, the collating element is ignored as if the string did not contain the collating element.
- 2 Indicates that this collating element is mapped into a string of collating elements. When this value is encountered, the replacement string for the one-to-many mapping is found in the appropriate one-to-many character mapping structure. Refer to “One-to-Many Character Mappings” on page 52 for further information on one-to-many mappings.

Single-byte Character Collating Table

The single-byte character collating information is stored within an Array(256) of Binary(4). The Binary(4) elements contain offsets to the order and weights for that collating element. Refer to “Orders and Weights” on page 47 for further information on the order and weights.

The decimal value for the single-byte code point for the character is used as the subscript to access the collating element array. As illustrated in the example below, if you had a character with a code point of 'C1', you would use the decimal value of this code point as the subscript into the array. The offset of 698 at this subscript is used to gain access to the order and weights for that character. Assuming that the number of weights and sort rules field is set to 2, the example shows that this character has an order of 17 and weights of 21 and 43.



If the offset to the order and weights is set to a value of zero, this indicates that there is no collating value available for that collating element.

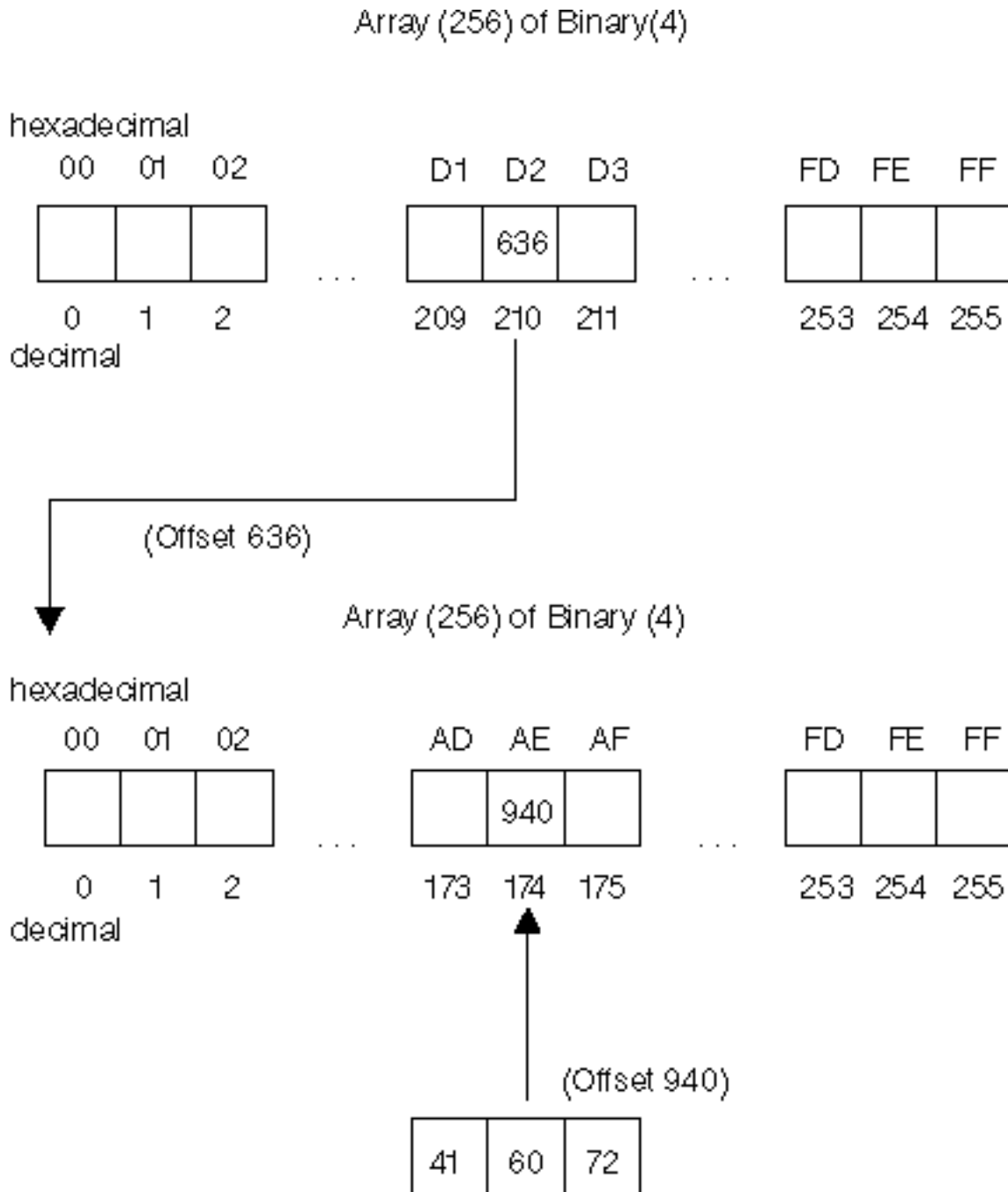
Double-byte Character Collating Table

The double-byte character collating information is stored within an Array(256) of Binary(4). The Binary(4) elements of this first array contain offsets to a second Array(256) of Binary(4). This second array contains offsets to the order and weights for that collating element. Refer to “Orders and Weights” on page 47 for further information on the order and weights.

The double-byte code point for the character is used as the subscripts to access the collating element arrays. The left-most byte of the double-byte character is used to access the first Array(256) of Binary(4). Within the subscripts of the first array are offsets to the second Array(256) of Binary(4). The right-most

byte of the double-byte character is used as the subscript to access this second array. Within the subscripts of the second array are offsets to the order and weights for that double-byte character.

As illustrated in the example below, if you had a double-byte character 'D2AE', the decimal value of 'D2' is used as the subscript for the first array. The offset of 636 at this subscript is used to gain access to the second array and the decimal value of 'AE' is used as the subscript for the second array. The offset of 940 is used to gain access to the order and weights. Assuming that the number of weights and sort rules field is set to 2, the example shows that this double-byte character has an order of 41 and weights of 60 and 72.



If the offset in the first array is set to zero, this indicates that there are no collating values available for this double-byte character range of collating elements. For example, if you have a double-byte code point 'CDFE' and the offset for the first array is set to zero, this would indicate that all double-byte characters

for this locale in the range where the left-most byte was 'CD' have no collating values available. Or, if the offset in the second array is set to zero, this indicates that there are no collating values available for this double-byte collating element. For example, if you have a double-byte code point 'CDFE' and the offset for the first array is not zero, but the offset for the second array is zero, then this indicates that there are no collating values available for this double-byte character.

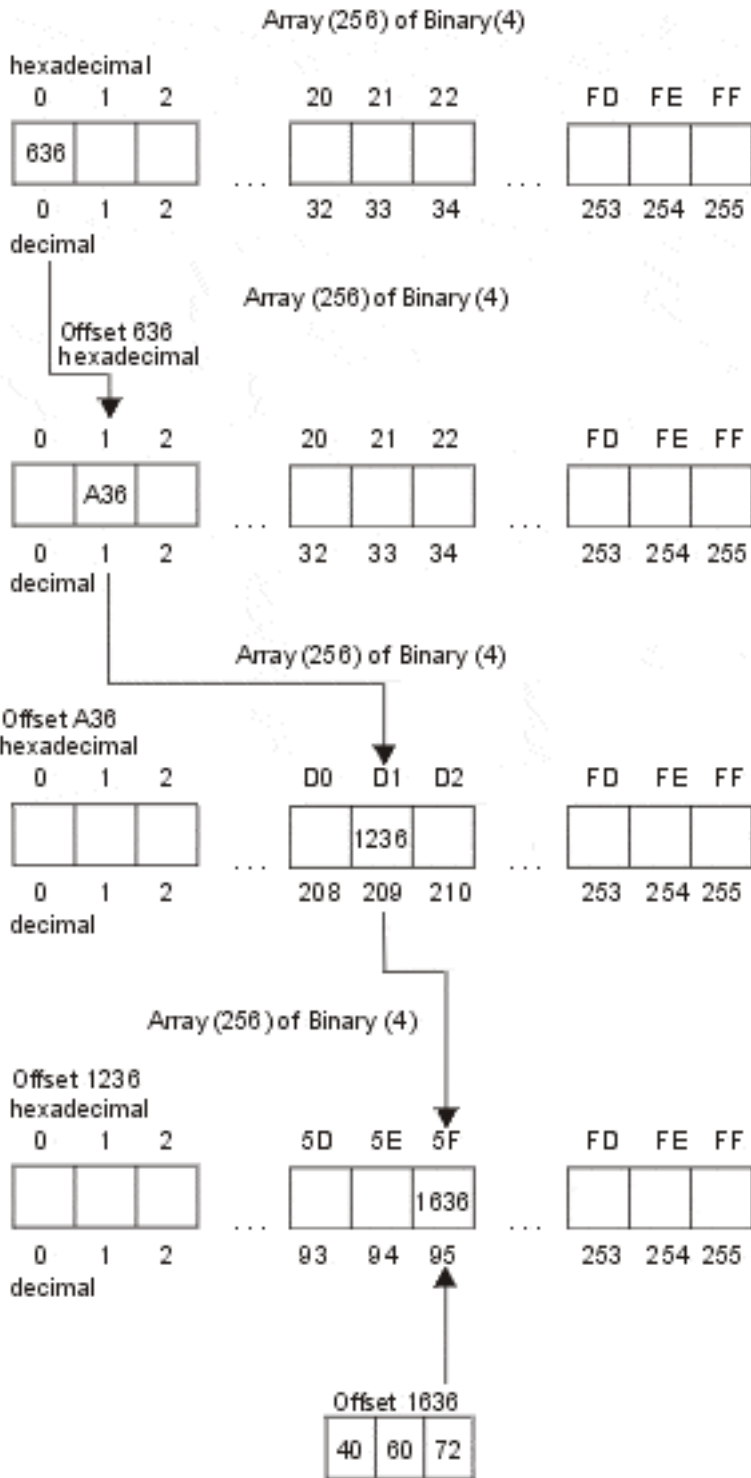


Four-byte Character Collating Table

The four-byte character collating information is stored within an Array(256) of Binary(4). The Binary(4) elements of this first array contain offsets to a second Array(256) of Binary(4). The Binary(4) elements of this second array contain offsets to a third Array(256) of Binary(4). The Binary(4) elements of this third array contain offsets to a fourth Array(256) of Binary(4). This fourth array contains offsets to the order and weights for that collating element. Refer to "Orders and Weights" on page 47 for further information on the order and weights.

The four-byte code point for the character is used as the subscripts to access the collating element arrays. The left-most byte of the four-byte character is used to access the first Array(256) of Binary(4). Within the subscripts of the first array are offsets to the second Array(256) of Binary(4). The second left-most byte of the four-byte character is used as the subscript to access this second array. Within the subscripts of the second array are offsets to the third Array(256) of Binary(4). The third left-most byte of the four-byte character is used as the subscript to access this third array. Within the subscripts of the third array are offsets to the fourth Array(256) of Binary(4). The rightmost byte of the four-byte character is used as the subscript to access this fourth array. Within the subscripts of the fourth array are offsets to the order and weights for that four-byte character.

As illustrated in the example below, if you had a four-byte character '0001D15F', the value of '00' is used as the subscript for the first array. The offset of 636 at this subscript is used to gain access to the second array and the value of '01' is used as the subscript for the second array. The offset of A36 at this subscript is used to gain access to the third array and the value of 'D1' is used as the subscript for the third array. The offset of 1236 at this subscript is used to gain access to the fourth array and the value of '5F' is used as the subscript for the fourth array. The offset of 1636 is used to gain access to the order and weights. Assuming that the number of weights and sort rules field is set to 2, the example shows that this four-byte character has an order of 40 and weights of 60 and 72.



RBAFX665-0

If the offset in the first array is set to zero, this indicates that there are no collating values available for this four-byte character range of collating elements. For example, if you have a four-byte code point '0000CDFE' and the offset for the first array is set to zero, this would indicate that all four-byte characters for this locale in the range where the left-most byte was '00' have no collating values available. Or, if the offset in the second array is set to zero, this indicates that there are no collating values available for this four-byte collating element. For example, if you have a four-byte code point '0000CDFE' and the offset for

the first array is not zero, but the offset for the second array is zero, then this indicates that there are no collating values available for this four-byte character. This is also true for the third and fourth arrays.



Multi-Character Collating Elements

The multi-character collating elements are returned as a character(*) field. The offset to the first multi-character element returned is the offset provided to the start of this field in the main structure. If the offset to the first multi-character element is zero, then there are no multi-character elements available. All elements are of the following format: (For a detailed description of each field, see Path name format.)

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Offset to multi-character collating elements.
		BINARY(4)	Length of multi-character collating element.
		BINARY(4)	Offset to order and weights for the multi-character element. Refer to the "Orders and Weights" on page 47 for further information on this field.>
		BINARY(4)	Offset to next multi-character collating element structure.

One-to-Many Character Mappings

The one-to-many character mappings are returned as a character(*) field. The offset to the first one-to-many mapping element returned is the offset provided to the start of this field in the main structure. If the offset to the first one-to-many mapping element is zero, then there are no one-to-many mapping elements available. All elements are of the following format: (For a detailed description of each field, see Path name format.)

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Offset to collating element. The length of this field is determined by the type of data. For example, if the type of data is single byte, then the length of the collating element is 1.
		BINARY(4)	Length of replacement string.
		BINARY(4)	Number of replacement strings.
		BINARY(4)	Offset to replacement string.
		BINARY(4)	Offset to next one-to-many character mapping.

LC_TIME - Date and Time Formatting Format

The following information is returned for the date and time formatting category. For a detailed description of each field, see Path Name Format.

The date and time formats have conversion specifiers that are used to build the formats. Refer to "Date and Time Conversion Specifiers" on page 53 for the list of conversion specifiers and their meaning.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Offset to date and time representation
		BINARY(4)	Length of date and time representation
		BINARY(4)	Offset to date representation
		BINARY(4)	Length of date representation
		BINARY(4)	Offset to time representation
		BINARY(4)	Length of time representation
		BINARY(4)	Offset to ante meridiem and post meridiem representation
		BINARY(4)	Length of ante meridiem and post meridiem representation
		BINARY(4)	Offset to time representation in a 12 hour clock format
		BINARY(4)	Length of time representation in a 12 hour clock format
		BINARY(4)	Offset to abbreviated weekday names
		BINARY(4)	Length of abbreviated weekday names
		BINARY(4)	Offset to full weekday names
		BINARY(4)	Length of full weekday names
		BINARY(4)	Offset to abbreviated month names
		BINARY(4)	Length of abbreviated month names
		BINARY(4)	Offset to full month names
		BINARY(4)	Length of full month names
		BINARY(4)	Offset to era
		BINARY(4)	Length of era
		BINARY(4)	Offset to era date format
		BINARY(4)	Length of era date format
		BINARY(4)	Offset to era time format
		BINARY(4)	Length of era time format
		BINARY(4)	Offset to era date and time format
		BINARY(4)	Length of era date and time format
		BINARY(4)	Offset to alternate digits
		BINARY(4)	Length of alternate digits
		CHAR(*)	Date and time data

Date and Time Conversion Specifiers

The conversion specifiers are used to build the formats for the date and time information. An example for a time format is %H:%M:%S. This time format specifies the hours as a decimal number followed by the minutes as a decimal number and the seconds as a decimal number with the colon (:) as the time separator.

Specifier	Conversion Specifier Description
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name

Specifier	Conversion Specifier Description
%B	Full month name
%c	Appropriate date and time representation
%C	Long format date and time representation
%d	Day of the month as a decimal number (01,31)
%D	Date in the format %m/%d/%y
%e	Day of the month as a decimal number (1,31) in a 2-digit field with leading space character fill
%E	Combined alternative era year and name, respectively in %o %N format
%h	Abbreviated month name (same as %b)
%H	Hour (24-hour clock) as a decimal number (00,23)
%I	Hour (12-hour clock) as a decimal number (01,12)
%j	Day of the year as a decimal number (001,366)
%m	Month as a decimal number (01,12)
%M	Minute as a decimal number (00,59)
%n	Newline character
%N	Alternative era name
%o	Alternative era year
%O	Alternative digits
%p	Ante meridiem (am) or post meridiem (pm)
%r	Time in AM/PM notation according to U.K./U.S. conventions in the format %I:%M:%S %p
%R	Time in 24 hour notation in the %H:%M format
%S	Second as a decimal number (00,61) - the 61 allows for the leap second and double leap second
%t	Tab character
%T	Time in format %H:%M:%S
%U	Week number of the year (Sunday as the first day of the week) as a decimal number (00,53)
%w	Weekday (Sunday as the first day of the week) as a decimal number (0,6)
%W	Week number of the year (Monday as the first day of the week) as a decimal number (00,53)
%x	Appropriate date representation
%X	Appropriate time representation
%y	Year without a century as a decimal number (00,99)
%Y	Year with a century as a decimal number
%Z	Timezone name

LC_NUMERIC - Numeric and Non-Monetary Formatting Format

The following information is returned for the numeric and non-monetary formatting category. For a detailed description of each field, see Path name format.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		BINARY(4)	Offset to decimal delimiter

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Length of decimal delimiter
		BINARY(4)	Offset to thousands separator
		BINARY(4)	Length of thousands separator
		BINARY(4)	Offset to digit grouping
		BINARY(4)	Number of digit groupings
		CHAR(*)	Numeric formatting data

LC_MESSAGES - Affirmative and Negative Responses Format

The following information is returned for the affirmative and negative responses category. For a detailed description of each field, see Path name format.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		BINARY(4)	Offset to affirmative response expression
		BINARY(4)	Length of affirmative response expression
		BINARY(4)	Offset to negative response expression
		BINARY(4)	Length of negative response expression
		BINARY(4)	Offset to affirmative response string
		BINARY(4)	Length of affirmative response string
		BINARY(4)	Offset to negative response string
		BINARY(4)	Length of negative response string
		CHAR(*)	Message response data

LC_MONETARY - Monetary Related Numeric Formatting Format

The following information is returned for the monetary-related numeric formatting category. For a detailed description of each field, see Path name format.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		BINARY(4)	Number of fractional digits for international currency symbol
		BINARY(4)	Number of fractional digits for currency symbol
		BINARY(4)	Currency symbol positioning for monetary non-negative values
		BINARY(4)	Currency symbol positioning for monetary negative values
		BINARY(4)	Currency symbol separator for monetary non-negative values
		BINARY(4)	Currency symbol separator for monetary negative values

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Positive sign positioning for monetary non-negative values
		BINARY(4)	Negative sign positioning for monetary negative values
		BINARY(4)	Offset to international currency symbol
		BINARY(4)	Length of international currency symbol
		BINARY(4)	Offset to local currency symbol
		BINARY(4)	Length of local currency symbol
		BINARY(4)	Offset to monetary decimal delimiter
		BINARY(4)	Length of monetary decimal delimiter
		BINARY(4)	Offset to monetary thousands separator
		BINARY(4)	Length of monetary thousands separator
		BINARY(4)	Offset to monetary digit grouping
		BINARY(4)	Number of monetary digit groupings
		BINARY(4)	Offset to monetary positive sign
		BINARY(4)	Length of monetary positive sign
		BINARY(4)	Offset to monetary negative sign
		BINARY(4)	Length of monetary negative sign
		CHAR(*)	Monetary formatting data

LC_TOD - Time Zone Information Format

The following information is returned for the time zone information category. For a detailed description of each field, see “Field Descriptions” on page 57.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	CCSID of locale data
		BINARY(4)	Locale data type
		Array(4) of BINARY(4)	Start for daylight savings time
		Array(4) of BINARY(4)	End for daylight savings time
		BINARY(4)	Greenwich Mean Time difference
		BINARY(4)	Daylight savings time shift
		BINARY(4)	Offset to time zone name
		BINARY(4)	Length of time zone name
		BINARY(4)	Offset to daylight savings time name
		BINARY(4)	Length of daylight savings time name
		CHAR(*)	Time zone data

LC_ALL - All Locale Categories

All information for all categories is returned in the following format. Refer to the specific locale category formats for details on each locale category. For a detailed description of each field, see “Field Descriptions.”

If the locale does not have any information defined for a locale category, the offset to the category field is set to zero.

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Offset to LC_CTYPE category
		BINARY(4)	Offset to LC_COLLATE category
		BINARY(4)	Offset to LC_TIME category
		BINARY(4)	Offset to LC_NUMERIC category
		BINARY(4)	Offset to LC_MESSAGES category
		BINARY(4)	Offset to LC_MONETARY category
		BINARY(4)	Offset to LC_TOD category
		CHAR(*)	Category information

Field Descriptions

The following section describes the fields returned in further detail.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Category information. The data for all the locale categories. Use the corresponding offsets and lengths to access the data that is included in this field. Refer to each locale category for further information.

CCSID of locale data. The coded character set identifier of the locale information that is returned.

Currency symbol positioning for monetary negative values. Indicates where the local or international currency symbol is positioned for a monetary quantity with a negative value. The possible values are:

- 0 The currency symbol succeeds the value.
- 1 The currency symbol precedes the value.

Currency symbol positioning for monetary non-negative values. Indicates where the local or international currency symbol is positioned for a monetary quantity with a non-negative value. The possible values are:

- 0 The currency symbol succeeds the value.
- 1 The currency symbol precedes the value.

Currency symbol separator for monetary negative values. Indicates whether a space separates the local or international currency symbol from the value for a monetary quantity with a negative value. The possible values are:

- 0 No space separates the currency symbol from the value.
- 1 A space separates the currency symbol from the value.
- 2 A space separates the currency symbol and the negative sign string, if adjacent.

Currency symbol separator for monetary non-negative values. Indicates whether a space separates the local or international currency symbol from the value for a monetary quantity with a non-negative value. The possible values are:

- 0 No space separates the currency symbol from the value.
- 1 A space separates the currency symbol from the value.
- 2 A space separates the currency symbol and the positive sign string, if adjacent.

Date and time data. The data for the date and time formatting. This includes the date and time representations, weekday names, month names, era information and alternate digits. This information is returned in the LC_TIME category. Use the corresponding offsets and lengths to get to the data that is included in this field.

Daylight savings time shift. The number of seconds that the locale's time is shifted when Daylight Savings Time takes effect.

End for daylight savings time. The instant when Daylight Savings Time ceases to be in effect. The format for this field is 'month week day time' with each field a Binary(4) value. For example, '0004 0000 0023 0000' sets the end time as April 23 at midnight. The blank spaces between each Binary(4) value is only used here for ease of reading. The actual value returned will not have blanks placed between the Binary(4) values.

Greenwich Mean Time difference. The number of minutes that the locale's time zone is different from Greenwich Mean Time.

Length of abbreviated month names. The length of the abbreviated month names.

Length of abbreviated weekday names. The length of the abbreviated weekday names.

Length of affirmative response expression. The length of the affirmative response expression.

Length of affirmative response string. The length of the affirmative response string.

Length of alternate digits. The length of the alternate digits.

Length of ante meridiem and post meridiem representation. The length of the ante meridiem and post meridiem representation.

Length of category information. The length of the category information. This field is the length of only the first level of the category. For example, if the LC_CTYPE category were specified and the data returned was double-byte, this field would contain a value of 20 (5 Binary(4) fields in the first level of the category). Whereas if the data returned was single-byte, this field would contain a value of 1032 (2 Binary(4), 2 char(256) and 1 Array(256) of unsigned Binary(2) fields in the first level of the category).

Length of date representation. The length of the date representation.

Length of date and time representation. The length of the date and time representation.

Length of daylight savings time name. The length of the daylight savings time name.

Length of decimal delimiter. The length of the decimal delimiter.

Length of era. The length of the era.

Length of era date format. The length of the era date format.

Length of era date and time format. The length of the era date and time format.

Length of era time format. The length of the era time format.

Length of full weekday names. The length of the full weekday names.

Length of full month names. The length of the full month names.

Length of international currency symbol. The length of the international currency symbol.

Length of local currency symbol. The length of the local currency symbol.

Length of monetary decimal delimiter. The length of the monetary decimal delimiter.

Length of monetary negative sign. The length of the monetary negative sign.

Length of monetary positive sign. The length of the monetary positive sign.

Length of monetary thousands separator. The length of the monetary thousands separator.

Length of multi-character collating element. The length of the multi-character collating element in bytes.

Length of negative response expression. The length of the negative response expression.

Length of negative response string. The length of the negative response string.

Length of replacement string. The length of the replacement string for the one-to-many character mapping.

Length of thousands separator. The length of the thousands separator.

Length of time representation. The length of the time representation.

Length of time representation of a 12-hour clock format. The length of the time representation of a 12-hour clock format.

Length of time zone name. The length of the time zone name.

Locale category key. The locale category key specified for the Locale category key to return parameter on the call to the API.

Locale data type. The indicator for the type of data returned from the locale. The possible values are:

- 1 The data returned is single-byte.
- 2 The data returned is double-byte (includes ISO 10646 16-bit character encoding standard (UCS2)).
- 3 The data returned is mixed single-byte and double-byte.
- » 4 The data returned is UTF8 data. «
- » 5 The data returned is UTF32 data. «

Message response data. The data for the affirmative and negative responses. This includes the affirmative and negative response expressions and strings information. This information is returned in the LC_MESSAGES category. Use the corresponding offsets and lengths to access the data that is included in this field.

Monetary formatting data. The data for monetary formatting. This includes the fractional digit, currency symbol positioning, currency symbol separator, positive and negative sign positioning, currency symbol, decimal delimiter, thousands separator, positive and negative sign and digit grouping information. This information is returned in the LC_MONETARY category. Use the corresponding offsets and lengths to access the data that is included in this field.

Negative sign positioning for monetary negative values. An integer representing a value indicating the positioning of the monetary negative sign for a negative formatted monetary quantity. The possible values are:

- | | |
|---|--|
| 0 | Parentheses enclose the quantity and the local or international currency symbol. |
| 1 | The negative sign string precedes the quantity and local or international currency symbol. |
| 2 | The negative sign string succeeds the quantity and local or international currency symbol. |
| 3 | The negative sign string precedes the local or international currency symbol. |
| 4 | The negative sign string succeeds the local or international currency symbol. |

Number of digit groupings. The number of digit groupings.

Number of fractional digits for currency symbol. An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using the local currency symbol.

Number of fractional digits for international currency symbol. An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using the international currency symbol.

Number of monetary digit groupings. The number of monetary digit groupings.

Number of replacement strings. The number of replacement strings.

Number of weights and sort rules. The number of weights and sort rules for character collating. There is one corresponding sort rule for each weight. Refer to “Character Collation Sort Rules” on page 47 for further information on the sort rules and “Orders and Weights” on page 47 for further information on the weights.

Numeric formatting data. The data for the numeric, non-monetary formatting. This includes the decimal delimiter, thousands separator and digit grouping information. This information is returned in the LC_NUMERIC category. Use the corresponding offsets and lengths to access the data that is included in this field.

Offset to abbreviated month names. Offset to the abbreviated month names. The abbreviated month names consist of twelve quoted strings separated by semicolons. The first string is the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on. The conversion specifier for this field is %b. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to abbreviated weekday names. Offset to the abbreviated weekday names. The abbreviated weekday names consist of seven quoted strings separated by semicolons. The first string is the abbreviated name of the day corresponding to Sunday, the second is the abbreviated name of the day

corresponding to Monday, and so on. The conversion specifier for this field is %a. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to affirmative response expression. Offset to the quoted string of an extended regular expression that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

Offset to affirmative response string. Offset to the quoted string that describes an acceptable affirmative response.

Offset to alternative digits. Offset to the alternative digits. The alternative digits is a group of quoted strings separated by semicolons. The first string represents the alternate string for zero, the second string represents the alternate string for one, and so on. A maximum of 100 alternate strings can be returned. The conversion specifier is %O. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to ante meridiem and post meridiem representation. Offset to the ante meridiem (am) and post meridiem (pm) representation. The ante meridiem (before noon) and post meridiem (after noon) representation consists of two quoted strings separated by a semicolon. The first string represents the ante meridiem designation and the second string represents the post meridiem designation. The conversion specifier for this field is %p. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to category information. Offset to the start of the category information returned.

Offset to collating element. Offset to the collating element that has a one-to-many mapping. The length of this field is determined by the type of data that is being returned. For example, if the type of data is single-byte, then the length of this field would be 1 byte.

Offset to date representation. Offset to the date representation. The date representation consists of a quoted string and can contain any combination of characters and field descriptors that represent the date. The conversion specifier for this field is %x. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to date and time representation. Offset to the date and time representation. The date and time representation consists of a quoted string and can contain any combination of characters and field descriptors that represent the date and time. The conversion specifier for this field is %c. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to daylight savings time name. Offset to the quoted string of characters that represent the time zone when Daylight Savings Time is in effect.

Offset to decimal delimiter. Offset to the quoted string containing a symbol that is used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities.

Offset to digit grouping. Offset to the digit grouping. The digit grouping defines the size of each group of digits in formatted non-monetary quantities. This field consists of a sequence of integers. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter. The following integers define the preceding groups. When the last integer is not -1, then the size of the previous group (if any) is used repeatedly for the remainder of the digits. When the last integer is -1, then no further grouping is performed. The number of integers in the sequence is determined by the Number of digit groupings field.

Offset to double-byte character classification. Offset to the character classification for a double-byte character. This information indicates what classifications a character has, for example if the character is a control character. Refer to the “Double-Byte Character Classification Structure” on page 41 for further information on this field.

Offset to double-byte character collating table. Offset to the double-byte character collating table. Refer to the “Double-byte Character Collating Table” on page 48 for further information on this field.

Offset to double-byte lowercase conversion mapping. Offset to the lowercase conversion mapping for a double-byte character. This mapping is used to find the lowercase character for the corresponding uppercase character. Refer to the “Double-Byte Conversion Mapping Structures” on page 36 for further information on this field.

Offset to double-byte one-to-many character mappings. Offset to the double-byte one-to-many character mappings. Refer to the “One-to-Many Character Mappings” on page 52 for further information on this field.

Offset to double-byte uppercase conversion mapping. Offset to the uppercase conversion mapping for a double-byte character. This mapping is used to find the uppercase character for the corresponding lowercase character. Refer to the “Double-Byte Conversion Mapping Structures” on page 36 for further information on this field.

Offset to era. Offset to the era. The era defines how the years are counted and displayed for each era, corresponding to the %E conversion specifier. The era consists of a quoted string in the format: “direction:offset:start_date:end_date:era_name:”. If more than one era is defined, each quoted string for each era is separated by a semicolon. Each element of the era format is defined as:

<i>direction</i>	Specifies a - (minus sign) or a + (plus sign) character. The plus character indicates that years count in the positive direction when moving from the start date to the end date. The minus character indicates that years count in the negative direction when moving from the start date to the end date.
<i>offset</i>	Specifies a number representing the first year of the era.
<i>start_date</i>	Specifies the starting date of the era in the yyyy/mm/dd format, where yyyy, mm and dd are the year, month and day, respectively. Years prior to the year AD 1 are represented as negative numbers.
<i>end_date</i>	Specifies the ending date of the era in the same form used for the start_date variable or one of the two special values -* or +*. A -* value indicates that the ending date of the era extends backward to the beginning of time. A +* value indicates that the ending date of the era extends forward to the end of time.
<i>era_name</i>	Specifies a string representing the name of the era. The conversion specifier is %EC.
<i>era_format</i>	Specifies a string for formatting the year in the era. The conversion specifier is %EY.

Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to era date format. Offset to the era date format. The era date format is a quoted string that defines the string used to represent the date in the alternate-era format. The conversion specifier is %Ex. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to era date and time format. Offset to the era date and time format. The era date and time format is a quoted string that defines the string used to represent the date and time in the alternate-era format. The conversion specifier is %Ec. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to era time format. Offset to the era time format. The era time format is a quoted string that defines the string used to represent the time in the alternate-era format. The conversion specifier is %EX. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.



Offset to four-byte character classification. Offset to the character classification for a four-byte character. This information indicates what classifications a character has, for example if the character is a control character. Refer to the “Four-Byte Character Classification Structure” on page 42 for further information on this field.



Offset to four-byte character collating table. Offset to the four-byte character collating table. Refer to the “Four-byte Character Collating Table” on page 50 for further information on this field.



Offset to four-byte lowercase conversion mapping. Offset to the lowercase conversion mapping for a four-byte character. This mapping is used to find the lowercase character for the corresponding uppercase character. Refer to the “Four-Byte Conversion Mapping Structures” on page 37 for further information on this field.



Offset to four-byte one-to-many character mappings. Offset to the four-byte one-to-many character mappings. Refer to the “One-to-Many Character Mappings” on page 52 for further information on this field.



Offset to four-byte uppercase conversion mapping. Offset to the uppercase conversion mapping for a four-byte character. This mapping is used to find the uppercase character for the corresponding lowercase character. Refer to the “Four-Byte Conversion Mapping Structures” on page 37 for further information on this field.



Offset to full month names. Offset to the full month names. The full month names consists of twelve quoted strings separated by semicolons. The first string is the full name of the first month of the year (January), the second the full name of the second month, and so on. The conversion specifier for this field is %B. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to full weekday names. Offset to the full weekday names. The full weekday names consists of seven quoted strings separated by semicolons. The first string is the full name of the day corresponding to Sunday, the second is the full name of the day corresponding to Monday, and so on. The conversion specifier for this field is %A. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to international currency symbol. Offset to the international currency symbol. The international currency symbol consists of a four character quoted string that denotes the symbol used for international monetary quantities. The first three characters contain the alphabetic international currency symbol. The fourth character is the character used to separate the international currency symbol from the monetary quantity.

Offset to LC_COLLATE category. Offset to the start of the character collation information.

Offset to LC_CTYPE category. Offset to the start of the character classification and conversion information.

Offset to LC_MESSAGES category. Offset to the start of the affirmative and negative responses information.

Offset to LC_MONETARY category. Offset to the start of the monetary related numeric formatting information.

Offset to LC_NUMERIC category. Offset to the start of the numeric and non-monetary formatting information.

Offset to LC_TIME category. Offset to the start of the date and time formatting information.

Offset to LC_TOD category. Offset to the start of the time zone information.

Offset to local currency symbol. Offset to the quoted string that denotes the symbol used for local monetary quantities.

Offset to monetary decimal delimiter. Offset to the quoted string containing a symbol that is used as the decimal delimiter (radix character) in monetary formatted quantities.

Offset to monetary digit grouping. Offset to the monetary digit grouping. The monetary digit grouping defines the size of each group of digits in formatted monetary quantities. This field consists of a sequence of integers. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter. The following integers define the preceding groups. When the last integer is not -1 then the size of the previous group (if any) is used repeatedly for the remainder of the digits. When the last integer is -1 no further grouping is performed. The number of integers in the sequence is determined by the Number of monetary digits groupings field.

Offset to monetary negative sign. Offset to the quoted string used to indicate a negative-valued formatted monetary quantity.

Offset to monetary positive sign. Offset to the quoted string used to indicate a non-negative-valued formatted monetary quantity.

Offset to monetary thousands separator. Offset to the quoted string containing the symbol that is used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.

Offset to multi-character collating element. Offset to the multi-character collating element. The multi-character collating element is a sequence of two or more characters to be collated as an entity.

Offset to multi-character collating elements. Offset to the multi-character collating elements. Refer to the "Multi-Character Collating Elements" on page 52 for further information on this field.

Offset to negative response expression. Offset to the quoted string of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

Offset to negative response string. Offset to the quoted string that describes an acceptable negative response.

Offset to next multi-character collating element. Offset to the next multi-character collating element. A value of zero in this field denotes the end of the list of multi-character collating elements.

Offset to next one-to-many character mapping. Offset to the next one-to-many character mapping. A value of zero in this field denotes the end of the list of one-to-many character mappings.

Offset to order and weights for the multi-character element. Offset to the order and weights for the multi-character collating element. Refer to the “Orders and Weights” on page 47 for further information on this field.

Offset to replacement string. Offset to the replacement string for a one-to-many mapping of a collating element. The replacement string is comprised of one or more quoted strings separated by semicolons. The number of replacement strings is determined by the number of replacement strings field.

Offset to single-byte character collating table. Offset to the single-byte character collating table. Refer to the “Single-byte Character Collating Table” on page 48 for further information on this field.

Offset to single-byte one-to-many character mappings. Offset to the single-byte one-to-many character mappings. Refer to the “One-to-Many Character Mappings” on page 52 for further information on this field.

Offset to sort rules. Offset to the sort rules for character collation. These rules are applied when comparing single or multi-character collating elements. Refer to the “Character Collation Sort Rules” on page 47 for further information on this field.

Offset to thousands separator. Offset to a quoted string containing a symbol that is used as the decimal separator for groups of digits to the left of the decimal delimiter in numeric, non-monetary formatted quantities.

Offset to time representation. Offset to the time representation. The time representation consists of a quoted string and can contain any combination of characters and field descriptors that represent the time. The conversion specifier for this field is %X. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to time representation of a 12-hour clock format. Offset to the time representation of a 12-hour clock format. The time representation of a 12-hour clock format consists of a quoted string and can contain any combination of characters and field descriptors. The conversion specifier for this field is %r. Refer to “Date and Time Conversion Specifiers” on page 53 for the list of conversion specifiers.

Offset to time zone name. Offset to the quoted string of characters that represent the time zone name.

Positive sign positioning for monetary non-negative values. An integer set to a value indicating the positioning of the monetary positive sign for a monetary quantity with a non-negative value. The possible values are:

- | | |
|---|--|
| 0 | Parentheses enclose the quantity and the local or international currency symbol. |
| 1 | The positive sign string precedes the quantity and local or international currency symbol. |
| 2 | The positive sign string succeeds the quantity and local or international currency symbol. |
| 3 | The positive sign string precedes the local or international currency symbol. |
| 4 | The positive sign string succeeds the local or international currency symbol. |

Reserved. An ignored field.

Single-byte character classification. The character classification for a single-byte character. This information indicates what classifications a character has, for example if the character is a control character. Refer to the “Single-Byte Character Classification Structure” on page 39 for further information on this field.

Single-byte lowercase conversion mapping. The lowercase conversion mapping for a single-byte character. This mapping is used to find the lowercase character for the corresponding uppercase character. Refer to the “Single-Byte Conversion Mapping Structures” on page 35 for further information on this field.

Single-byte uppercase conversion mapping. The uppercase conversion mapping for a single-byte character. This mapping is used to find the uppercase character for the corresponding lowercase character. Refer to the “Single-Byte Conversion Mapping Structures” on page 35 for further information on this field.

Start for daylight savings time. The instant when Daylight Savings Time comes into effect. The format for this field is ‘month week day time’ with each field a Binary(4) value. For example, ‘0010 0003 0006 0000’ sets the start time as the 3rd week of October at midnight on Saturday. The blank spaces between each Binary(4) value is only used here for ease of reading. The actual value returned will not have blanks placed between the Binary(4) values.

Time zone data. The data for the time zone information. This includes daylight savings time, Greenwich Mean Time, and time zone information. This information is returned in the LC_TOD category. Use the corresponding offsets and lengths to access the data that is included in this field.

Error Messages

Message ID	Error Message Text
CPFA0AB E	Object name not a directory.
CPFA0A2 E	Information passed to this operation was not valid.
CPFA0A3 E	Path name resolution causes looping.
CPFA0A7 E	Path name too long.
CPFA0A9 E	Object not found.
CPFA0C1 E	CCSID &1 not valid.
CPFA09C E	Not authorized to object.
CPFA09E E	Object in use.
CPFA09F E	Object damaged.
CPFA092 E	Path name not converted.
CPF24B4 E	Severe error while addressing parameter list.
CPF3BED E	Requested function cannot be performed at this time.
CPF3BEF E	C or POSIX locale specified in the environment variable.
CPF3BF1 E	Locale &1 does not exist.
CPF3BF2 E	Locale &1 is not a valid locale.
CPF3BF3 E	Locale not defined in environment variable.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C1E E	Required parameter &1 omitted.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R6

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Retrieve Sort Sequence Table (QLGRTVSS) API

Required Parameter Group:	
1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Qualified table name
Input	Char(20)
4	Language identifier requested
Input	Char(10)
5	CCSID of returned table
Input	Binary(4)
6	Format name
Input	Char(8)
7	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: Yes	

The Retrieve Sort Sequence Table (QLGRTVSS) API returns a sort sequence table based on the required input parameters. It will also return some associated job information.

Authorities and Locks

Sort Sequence Table Authority
*USE

Sort Sequence Table Library Authority
*USE

Sort Sequence Table Lock
*SHRNUP

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The receiver variable that receives the information requested. You can specify the size of the area to be smaller than the format requested, provided you specify the length of receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Qualified table name

INPUT; CHAR(20)

The sort sequence table to retrieve. The first 10 characters contain the table name, and the second 10 characters contain the name of the library in which the table resides.

The following special values are supported for the table name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

<i>*JOB</i>	The sort sequence of the job.
<i>*JOBRUN</i>	The API treats this value the same as <i>*JOB</i> .
<i>*LANGIDUNQ</i>	The unique-weight sort sequence table that is associated with the language identifier requested parameter.
<i>*LANGIDSHR</i>	The shared-weight sort sequence table that is associated with the language identifier requested parameter.
<i>*HEX</i>	The sort sequence according to the hexadecimal value of the characters.

Note: Both **JOB* and **JOBRUN* are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following special values are supported for the library name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

<i>*LIBL</i>	The user's library list.
<i>*CURLIB</i>	The current library.

If a special value is used for the table name, the second 10 characters used for the library name must be blank.

Language identifier requested

INPUT; CHAR(10)

The language identifier of the sort sequence table to be used. All values must be in uppercase and must be padded with blanks to the right of the value. This value will be ignored if the value of the table name parameter is a table object or **HEX*. Valid values are:

<i>*JOB</i>	Use the language identifier of the job.
<i>*JOBRUN</i>	The API treats this value the same as <i>*JOB</i> .
<i>Specific language identifier</i>	A valid 3-character language identifier. For example, Danish would be "DAN". See the globalization topic for a complete list of valid language identifiers.

Note: Both **JOB* and **JOBRUN* are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

CCSID of returned table

INPUT; BINARY(4)

The coded character set identifier (CCSID) in which the sort sequence table should be returned. The valid values for this parameter are:

0	The CCSID of the job will be used to determine the CCSID of the sort sequence table to be returned.
1-65533	A valid CCSID in this range.
65535	The CCSID of the sort sequence table that is found should be used. No CCSID conversion should be done on the found table.

Format name

INPUT; CHAR(8)

The content and format of the information returned for each table. The format name is:

RSST0100 Sort sequence format for single-byte sort sequence table.

See “RSST0100 Format” for a description of this format.

RSST0200 Sort sequence format for a single-byte or UCS-2 sort sequence table.

See “RSST0200 Format” for a description of this format.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

RSST0100 Format

Following is the format of the information returned. For a description of the fields in this format, see “Field Descriptions” on page 70.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	BINARY(4)	Job CCSID value
12	C	BINARY(4)	CCSID of returned table
16	10	CHAR(1)	Substitution values encountered
17	11	CHAR(1)	Weighting of returned sort sequence table
18	12	CHAR(10)	Returned table name
28	1C	CHAR(10)	Returned table library name
38	26	CHAR(10)	Job sort sequence table name
48	30	CHAR(10)	Job sort sequence table library name
58	3A	CHAR(3)	Job language identifier
61	3D	CHAR(2)	Job country or region identifier
63	3F	CHAR(256)	Returned sort sequence table

RSST0200 Format

Following is the format of the information returned. For a description of the fields in this format, see “Field Descriptions” on page 70.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	BINARY(4)	Job CCSID value
12	C	BINARY(4)	CCSID of returned table
16	10	CHAR(1)	Substitution values encountered

Offset		Type	Field
Dec	Hex		
17	11	CHAR(1)	Weighting of returned sort sequence table
18	12	CHAR(10)	Returned table name
28	1C	CHAR(10)	Returned table library name
38	26	CHAR(10)	Job sort sequence table name
48	30	CHAR(10)	Job sort sequence table library name
58	3A	CHAR(3)	Job language identifier
61	3D	CHAR(2)	Job country or region identifier
63	3F	CHAR(1)	Type of sort sequence table returned
64	40	BIN(4)	Offset to returned sort sequence table
68	44	BINARY(4)	Size of returned sort sequence table
		CHAR(*)	Returned sort sequence table

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID of returned table. The coded character set identifier (CCSID) in which the returned sort sequence table is encoded.

Notes:

1. No conversion is performed if the CCSID tag of the stored table is 65535 or if the job CCSID value is used and it is 65535.
2. All tables created before Version 2 Release 3 will always be assumed to be tagged with a CCSID value of 65535.
3. If *HEX is specified for the sort sequence to get, this value is set to 65535.

Job CCSID value. The CCSID value of this job.

Job country or region identifier. The country or region identifier of this job.

Job language identifier. The language identifier of this job.

Job sort sequence table library name. The sort sequence table library name of this job.

Job sort sequence table name. The sort sequence table name of this job. It can have one of the special values:

- *LANGIDUNQ The unique-weight sort sequence table that is associated with the language identifier requested parameter.
- *LANGIDSHR The shared-weight sort sequence table that is associated with the language identifier requested parameter.
- *HEX The sort sequence according to the hexadecimal value of the characters.

Offset to returned sort sequence table. The offset in bytes to the first element in the returned sort sequence table. A value of 0 indicates no table was returned.

Returned table name. The name of the sort sequence table that was returned.

The value of the returned table name is "*N " if the table returned is the result of a requested sort sequence of *HEX or the result of converting an existing table from its stored CCSID representation to the CCSID requested. (A new table object will not be created in this case.)

Returned table library name. The name of the sort sequence table library that was returned.

The value of the returned table library name is "*N " if the table returned is the result of a requested sort sequence of *HEX or the result of converting an existing table from its stored CCSID representation to the CCSID requested.

If *LIBL or *CURLIB was specified for the qualified table name, the name of the library in which the sort sequence resides will be returned.

Returned sort sequence table. The variable length sort sequence table that is described by the input values. The returned UCS-2 sort sequence table can be used as control map type E-1 for the XLATEMB instruction by using the information that starts at offset 94 from the beginning of the table and ends at the end of the table.

Size of returned sort sequence table. The number of bytes in the sort sequence table that is returned.

Substitution values encountered. Whether substitution values were involved during the conversion from the source CCSID of the table to the requested CCSID. A substitution occurs when a character in the source CCSID does not exist in the requested CCSID. The valid values are:

0	No substitutions were involved.
1	Substitutions were involved.

Type of sort sequence table returned. The type of sort sequence table returned. The valid values are:

0	Single byte sort sequence table returned.
1	UCS-2 sort sequence table returned.

Weighting of returned sort sequence table. The weighting of sort sequence table returned. The valid values are:

1	A shared-weight table.
2	A unique-weight table.

Note: Any table created prior to Version 2 Release 3 is always returned as a shared-weight table.

Layout of a Returned UCS-2 Sort Sequence Table

The size of a returned sort sequence table includes 128 bytes of the header, 4 bytes containing the number of cells, and the number of cells * 4 bytes.

A cell has the following layout:

2 bytes for the code point

2 bytes for the weight

The remainder of the line is ignored and can be used for comments.

Header			CHAR (128)
Number of cells			BIN (4)
Code point	Weight	comment	
Code point	Weight	comment	
.	.		
.	.		
Code point	Weight	comment	
.	.		
.	.		

The following is an example of a UCS-2 sort sequence table.

Header			CHAR (128)
Number of cells			BIN (4)
00610041		lowercase a	
00C10042		uppercase A	
00620043		lowercase b	
00C20044		uppercase B	
.			
.			
00310095		number one	
.			
.			

Error Messages

Message ID	Error Message Text
CPF2115 E	Object &1 in &2 type *&3 damaged.
CPF2169 E	Job's sort sequence information not available.
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3BC6 E	Sort sequence &1 not valid.
CPF3BC7 E	CCSID &1 outside of valid range.
CPF3BC8 E	Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E	Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3BCC E	Language identifier &1 not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.

Message ID	Error Message Text
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Scan String for Mixed Data (QLGSCNMX) API

Required Parameter Group:	
1	Double-byte indicator
Output	Char(1)
2	Input data buffer
Input	Char(*)
3	Length of input data buffer
Input	Binary(4)
4	Error code
I/O	Char(*)
	Default Public Authority: *EXCLUDE
Threadsafe: No	

The Scan String for Mixed Data (QLGSCNMX) API tests a mixed EBCDIC input string to determine if the data contains any double-byte characters. If so, the double-byte indicator will be set to 1.

The QLGSCNMX API determines that a mixed EBCDIC input string contains double-byte characters when the shift-out (X'0E') control character is present.

Required Parameter Group

Double-byte indicator

OUTPUT; CHAR(1)

Information about the input data. The valid values are:

0	Input data does not contain double-byte data.
1	Input data contains double-byte data.

Input data buffer

INPUT; CHAR(*)

The data to be scanned.

Length of input data buffer

INPUT; BINARY(4)

The length of the input data buffer. The valid range is 1 to 32767 bytes.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID Error Message Text

CPF24B4 E Severe error while addressing parameter list.

CPF2647 E Buffer length not valid.

CPF3C90 E Literal value cannot be changed.

CPF3CF1 E Error code parameter not valid.

CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Sort (QLGSORT) API

Required Parameter Group:	
1	Request control block
Input	Char(*)
2	Input data buffer
Input	Char(*)
3	Output data buffer
Output	Char(*)
4	Length of output data buffer
Input	Binary(4)
5	Length of returned data
Output	Binary(4)
6	Error code
I/O	Char(*)
Optional Parameter Group:	
7	Returned records feedback
Output	Char(*)
8	Length of returned records feedback
Input	Binary(4)
	Default Public Authority: *USE
Threadsafe: No	

The Sort (QLGSORT) API provides a generalized sort function that can be directly called by any application program. This API can be used to sort data in files or data in an input buffer with a single call. This API can also be used to initialize a sort function where a series of calls using the Sort Input/Output (QLGSRTIO) API is done to repeatedly add data to be sorted as a set of records and return sorted data as a set of records.

The following types of sort operations can be done with the API:

- Sort up to 32 input files and produce 1 to 32 output files, each containing the full list of sorted records.
- Sort up to 32 input files and return all of the records in the output data buffer parameter.
- Sort up to 32 input files, and return the output a set at a time through the QLGSRTIO API.
- Sort a set of records in the input data buffer parameter and produce 1 to 32 output files, each containing the full list of sorted records.
- Sort a set of records in the input data buffer parameter and return all of the records in the output data buffer parameter.
- Sort a set of records in the input data buffer parameter and return the output a set at a time through the QLGSRTIO API.
- Initialize a sort where the records are provided a set at a time through the QLGSRTIO API and produce 1 to 32 output files, each containing the full list of sorted records.

- Initialize a sort where the records are provided a set at a time through the QLGSRATIO API and are also returned a set at a time through that API.

The sort allows the application to indicate whether character data should be sorted as a binary (hexadecimal) sort or sorted using a national language sort sequence for obtaining sort results consistent with a specific locale. When specifying a character field as binary (hexadecimal), the national language sort sequence is not used. The hexadecimal sort is helpful if DBCS-graphic data is being sorted. This sort can also be used to define character fields that contain hexadecimal or bit data. The caller can indicate if character data being sorted might contain double-byte characters. This type of field is considered a DBCS-open field, where DBCS data is surrounded by control characters. If a field is defined as such, the API will apply a national language sort sequence only to the single-byte data in the field. Performance may be slower when this type of character field is defined because of additional checking that occurs for DBCS data. If a field is defined as a single-byte field but DBCS data exists in the field, the national language sort sequence is applied to the DBCS data as if it were single-byte data, which may result in an unexpected sequence.

The sort allows a character field to contain 2-byte characters in UCS-2. If a field is defined as such, the API will apply a UCS-2 national language sort sequence if specified in the national language sort information.

The sort allows a character field to be variable length. All occurrences of the variable length character field have the same maximum length. The actual data length is defined by a 2-byte binary value that immediately precedes the character field. When using a variable length field, all bytes between the actual data length and the maximum data length are sorted as if they were blanks. Variable length fields are allowed only when sorting buffers of data or when using files for both input and output.

The sort allows a record to be defined for variable length record access. When using variable length record access, a 2-byte binary value indicates the actual data length of the record. When sorting records of different length, the shorter record is logically padded to the length of the longer record with blanks. This occurs only if the sort specification refers to a key size greater than the actual record length of the shorter record. This support is mutually exclusive of variable-length-field sort support. It is allowed only when sorting buffers of data and when using externally described database files. When using database files for both input and output, the file must be defined as having all fixed length fields except for the last field. The last field must be variable length.

The sort also allows fields to be defined as null capable. Fields that are set to null sort higher than any non-null value. Null support is allowed only when sorting buffers of data and when using externally described database files for both input and output.

The QLGSRATIO API requires that no earlier sort be active for this job at the time the sort is called. This is especially important when the QLGSRATIO API is called to initialize a sort and the QLGSRATIO API is repeatedly called to add data to the sort or return data from the sort. When retrieving data from the sort using the QLGSRATIO API, the normal operation is to continue calling QLGSRATIO until all of the data is returned. Whenever a get request is made and there is no more data to be returned, the sort is ended. If a get request is made and data still remains to be returned, the sort is still active until another get request is made and all of the data is returned. Another way to end the sort is to specify Cancel (request type 4) on the QLGSRATIO API call. The cancel request causes the sort to end immediately, regardless of whether there is still data to be returned. Whenever a sort is active and QLGSRATIO is called again by the same job, an error is returned.

When using the QLGSRATIO API to put data to the sort and the output is to be put in output files, the end put operation on the call to the QLGSRATIO API causes the data to be sorted and the output to be generated. The sort is then ended without the need to call QLGSRATIO with a cancel request.

When QLG SORT is called once to perform sorting of data in an input data buffer or in files, with the output going to an output data buffer or files, the sort is ended within that one call. All internal spaces have been deleted. Subsequent calls to the QLG SORT API are therefore valid.

If input files are specified and an error occurs with any of them, an error message is returned and no sorting occurs. It is up to the caller to determine if the QLG SORT API must be called again to perform the sort operation. There is no capability to continue with the previous sort.

If output files are specified, the file member of a physical output file is cleared before the sorted data is put to the file. If output files are specified and an open or put error occurs with any of the files, the sort continues to put data to the files that are not in error. For some I/O errors, an inquiry message is sent to the system operator message queue (QSYSOPR). Processing stops until a reply is received for the message. If all of the output files are in error, a message is returned indicating that the data was sorted but there were no valid output files in which to put the data.

Authorities and Locks

Input File Authority

*USE

Input File Library Authority

*USE

Output File Authority

Both *OBJOPR and *ADD or both *OBJALT and *ADD

Output File Library Authority

*USE

Sort Sequence Table Authority

*USE

Sort Sequence Table Library Authority

*USE

Input File Lock

*SHRNUP

Output File Lock

*EXCL

Sort Sequence Table Lock

*SHRNUP

Required Parameter Group

Request control block

INPUT; CHAR(*)

Information defining the sort, such as whether a list of files or an input data buffer will be sorted. It also defines the keys to be used for the sort. Refer to "Format of Request Control Block" on page 79 for details.

Input data buffer

INPUT; CHAR(*)

The input data to be sorted. The calling program is responsible for adding all of the data to be sorted to this parameter before calling the QLG SORT API. The input data buffer parameter must contain the data to be sorted when the type of request field is 4, 5, or 6. For information on how to format this buffer, see "Buffer Layout Examples" on page 88.

This parameter is ignored when the type of request field is 1, 2, 3, 7, or 8 because either the file data will be sorted or the data will be provided through calls to the QLGSRATIO API.

Output data buffer

OUTPUT; CHAR(*)

The sorted output data to be returned to the calling program. The output data buffer parameter will contain the sorted output data when the type of request field in the request control block is 2 or 5.

This parameter is ignored when the type of request field is 1, 3, 4, 6, 7, or 8 because data is either provided in output files or through calls to the QLGSRATIO API. For information on how this buffer is formatted, see “Buffer Layout Examples” on page 88.

The QLGSRATIO API returns only the data that the buffer can hold, up to the length of data that is available to be returned. The size of the output data buffer must be greater than or equal to the record length field of the request control block. This is to ensure that at least one record can be provided as output. The output data buffer parameter can be the same as the input data buffer parameter.

Length of output data buffer

INPUT; BINARY(4)

The maximum length of the output data to be returned to the calling program in the output data buffer parameter. If this length is larger than the actual size of the output data buffer parameter, the results may not be predictable. The length of output data buffer must be at least as large as the record length value specified in the request control block when the type of request field in the request control block is 2 or 5. If it is not, an error is returned.

The length of output data must be set to 0 when the type of request field is 1, 3, 4, 6, 7, or 8 because the data is returned either as output files or through calls to the QLGSRATIO API. The maximum length allowed is 16MB less 512 bytes.

Length of returned data

OUTPUT; BINARY(4)

The actual length of the output data added to the output data buffer parameter. When no output data is returned, this parameter is set to 0.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Optional Parameter Group

Returned records feedback

OUTPUT; CHAR(*)

The number of records added to each output file to be returned to the calling program. The returned records feedback parameter contains the number of records added to each output file.

This parameter is ignored when the type of request field is 2, 3, 5, 6, 7, or 8. This is because data is returned in buffers or through calls to the QLGSRATIO API.

The number of records added to each output file corresponds to the list of files specified in the request control block. To receive results, you must also set the options field in the request control block to 4, 5, 6, or 7.

See “Format of Returned Records Feedback Information” on page 80 for details.

Length of returned records feedback

INPUT; BINARY(4)

The length of the returned records feedback parameter to be returned to the calling program. If this length is larger than the actual size of the returned records feedback parameter, the results may not be predictable. The minimum length is 8. See the “Format of Returned Records Feedback Information” on page 80 for details.

This parameter is ignored when the type of request field is 2, 3, 5, 6, 7, or 8. This is because data is returned in buffers or through calls to the QLGSRATIO API.

Format of Request Control Block

For a description of the fields in this format, see “Field Descriptions” on page 80.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of request control block
4	4	BINARY(4)	Type of request
8	8	BINARY(4)	Reserved
12	C	BINARY(4)	Options
16	10	BINARY(4)	Record length
20	14	BINARY(4)	Record count
24	18	BINARY(4)	Offset to key list
28	1C	BINARY(4)	Number of keys
32	20	BINARY(4)	Offset to national language sort information
36	24	BINARY(4)	Offset to input file list
40	28	BINARY(4)	Number of input files
44	2C	BINARY(4)	Offset to output file list
48	30	BINARY(4)	Number of output files
52	34	BINARY(4)	Length of key entry
56	38	BINARY(4)	Length of national language sort sequence information
60	4C	BINARY(4)	Length of input file entry
64	50	BINARY(4)	Length of output file entry
68	54	BINARY(4)	Offset to null byte map
72	58	BINARY(4)	Offset to variable length record access information
76	6C	BINARY(4)	Reserved
<p>Note: Format of entries in the key list. The following fields are repeated for each key entry. The decimal and hexadecimal offsets depend on the number of key entries. The first key entry is found by using the offset to key list field. At least one key must be specified. Sorting will be done on the keys in the order they are specified.</p>			
		BINARY(4)	Key starting position
		BINARY(4)	Key string size
		BINARY(4)	Key data type
		BINARY(4)	Sort order
		BINARY(4)	Ordinal position of key field
<p>Note: Format of national language sort information. The following fields are not repeated. The decimal and hexadecimal offsets depend on the number of key entries. The first field is found by using the offset to national language sort information field. If you want a hexadecimal sort instead of a national language sort, set the offset to national language sort information field to 0. In this case, you do not need to specify the sort sequence fields.</p>			
		CHAR(20)	Qualified sort table name

Offset		Type	Field
Dec	Hex		
		BINARY(4)	Sort sequence CCSID
		CHAR(10)	Sort sequence language ID
		CHAR(256)	Sort sequence table
<p>Note: Format of entries in the input file list. The following fields are repeated for each input file entry. The decimal and hexadecimal offsets depend on the number of input file entries and the number of key entries. The first input file entry is found by using the offset to input file list field. If the input data buffer parameter contains data to be sorted, or data will be provided through the QLSRTIO API, the offset to input file list field must be set to 0.</p>			
		CHAR(20)	Qualified input file name
		CHAR(10)	Input member name
		BINARY(4)	Variable length record access
		BINARY(4)	Null-capable fields
<p>Note: Format of entries in the output file list. The following fields are repeated for each output file entry. The decimal and hexadecimal offsets depend on the number of input file list entries and key list entries. The first output file entry is found by using the offset to output file list field. If the sorted records are to be returned in the output data buffer, or data will be returned through calls to the QLSRTIO API, the offset to output file list field must be set to 0.</p>			
		CHAR(20)	Qualified output file name
		CHAR(10)	Output member name
		BINARY(4)	Variable length record access
		BINARY(4)	Null-capable fields

Format of Returned Records Feedback Information

For a description of the fields in this format, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	BINARY(4)	Offset to start of returned records array
12	C	BINARY(4)	Number of output files
		ARRAY of BINARY(4)	Returned records

Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Input member name. The name of the member in the input file. A value of *FIRST indicates that the first member of the file will be used. A value of *LAST indicates that the last member of the file will be used. If the specified member does not exist, an error is returned and no sorting is done.

Key data type. The following are the values that may be specified:

0	Signed binary.
1	Signed binary floating point.
2	Signed zoned decimal.
3	Signed packed decimal.
4	Character with national language sort sequence applied, if specified. DBCS data will be treated as single-byte data.
5	Mixed character with national language sort sequence applied, if specified, only to single-byte data.
6	Character with no national language sort sequence applied, if specified.
7	Unsigned packed decimal. All numbers will have the sign forced positive ('F'X).
8	Unsigned zoned decimal. All numbers will have the sign forced positive ('F'X).
9	Unsigned binary.
10	Zoned decimal with a sign over the leading digit. The sign occupies the left-most 4-bits of the value.
11	Zoned decimal with a separate trailing sign. Valid values are +, -, and blank.
12	Zoned decimal with a separate leading sign. Valid values are +, -, and blank.
13	Date in the form of MM/DD/YY, where: YY Year MM Month DD Day
14	Date in the form of DD/MM/YY
15	Date in the form of DD.MM.YYYY
16	Date in the form of MM/DD/YYYY
17	All other date/time formats
18	Time in the form of HH:MM xM, where: HH Hour MM Minute x A or P
19	Variable length character with national language sort sequence applied, if specified. DBCS data will be treated as single-byte data.
20	Variable length mixed character with national language sort sequence applied, if specified, only to single-byte data.
21	Variable length character with no national language sort sequence applied, if specified.
22	Variable length UCS-2 character with national language sort sequence applied, if specified.
23	UCS-2 character with national language sort sequence applied, if specified.

The use of any other value will cause an error to be returned.

Key starting position. The starting position of the key field in the record. The starting position must be greater than 0 and cannot exceed the record length specified in the record length field, or an error is returned.

The same key starting position is used for all records to be sorted. Padding of blanks occurs on records that are shorter than the record length field. If a key starting position is within the area that is padded, these records are normally sorted into the first positions.

For key data types 19, 20, 21, and 22, the starting position should be byte 0 of the 2-byte length that precedes the character data.

Key string size. The number of bytes to be used for sorting this field. If the key field data type is 19, 20, 21, or 22, this must be the maximum length of the field in single bytes not including the 2-byte binary length in front of the data. The key string size must be greater than 0, and the string size plus the key

starting position cannot exceed the record length field; otherwise, an error is returned. Also, the sum of the size of all of the keys cannot exceed 2000 bytes. If the sum exceeds this maximum, an error is returned.

Note: Data that is encoded in UCS-2 requires 2 bytes for each character.

Length of input file entry. The total length of the input file entry. If specified, the minimum length must be 30 bytes. If the input file includes any of the following, the length of the entry must be 38 bytes:

- Variable length record access
- Null-capable fields

Length of key entry. The total length of the key entry. If a null key is allowed, the key length must be 20 bytes. If specified, the minimum length must be 16 bytes.

Length of national language sort sequence information. The total length of the national language sort sequence information. If specified, the minimum length must be 290 bytes.

Length of output file entry. The total length of the output file entry. If specified, the minimum length must be 30 bytes. If the output file includes either of the following, the length of the entry must be 38 bytes:

- Variable length record access
- Null-capable fields

Length of request control block. The total length of the request control block. The minimum size is 72 bytes, which allows for one key in the key list, no input or output files, and no national language sort sequence information. An error is returned if the length specified is less than the minimum.

Null-capable fields. Whether null-capable fields are supported in a file. The possible values follow:

0	The file does not contain null-capable fields.
1	The file contains null-capable fields. This value is allowed only when sorting externally described database files.

Number of input files. The number of input files specified in the input file list. If the number of input files is 0 and the request type field is 1, 2, or 3 (file sort), an error is returned. Up to 32 input files can be specified.

Number of keys. The number of keys specified in the request control block. The sum of the lengths of all of the key fields cannot exceed 2000 bytes. Therefore the number of keys is limited to a maximum of 2000, which allows each key to be 1 byte long. At least one key field must be defined.

Number of output files. The number of output files specified in the output file list. If the number of output files is 0 and the request type field is 1, 4, or 7 (file output), an error is returned. Up to 32 output files can be specified.

Offset to input file list. The offset to the start of the input file list structure. If data in the input data buffer parameter is being sorted, or input data will be provided by the QLGSRIO API, the input file list structure is not required and this offset must be set to 0.

Offset to key list. The offset to the start of the key list structure. At least one key must be defined in the key list structure, or an error is returned.

Offset to national language sort information. The offset to the start of the national language sort information structure. If you want a hexadecimal sort instead of a national language sort, set this offset to 0. In this case, you do not need to specify the sort sequence fields.

Offset to null byte map. The offset to the start of the null byte map. The null byte map contains a 1-byte value for each field in the record. The order of the bytes corresponds to the order of the fields. If the field contains null data, the value should be set to 1; otherwise, the value should be 0.

This offset must be set if there are null-capable fields in the input buffer. If there are no null-capable fields in the input buffer, the value must be set to 0. Also, the value must be set to 0 if the type of request is 1, 2, 3, 4, or 7. See the “Buffer Layout Examples” on page 88 for information on how to format this buffer.

Offset to output file list. The offset to the start of the output file list structure. If data is being returned in the output data buffer parameter or if the QLGSRATIO API will be used to return the sorted data, the output file list structure is not required and this offset must be set to 0.

Offset to start of returned records array. The offset to the returned records array.

Offset to variable length record access information. The offset to the start of the variable length record access information. This offset must be set if variable length record access is being used for buffer processing.

The offset should be set to the maximum length of the actual user-defined data for a record in a buffer or file plus 1 byte. For example, if the largest user data in the buffer is 80 bytes, this field would be 81. At offset 81 within each record to be sorted, a 2-byte binary value is set to indicate the actual length of the user data.

When used in conjunction with null-capable fields, this offset value must be less than the value specified for the offset to null byte map field. This offset must be 0 unless the type of request is 5, 6, or 8 and the buffer contains variable length records.

Options. Additional options to be handled by the sort. The following values are defined:

- | | |
|---|--|
| 0 | No options used. |
| 1 | Log record count messages. One message is returned in the job log for each output file indicating the number of records written to the output file. When output files are not used, one message is returned indicating the total number of records sorted. |
| 2 | Throw away duplicate keyed records when writing to files requiring unique keys. The normal processing is to maintain duplicates in the order in which they are received. This option allows for succeeding data to be written to the file or buffer by bypassing records with duplicate keys. If this option is not selected and a duplicate key is encountered when writing data to a database file defined to have unique keys, a message is returned. No further data is entered into the file. |
| 3 | Log record count messages and throw away duplicate keyed records. This option is a combination of options 1 and 2. |
| 4 | Store record count for each file in the returned records feedback. When output files are not used, a message is returned to the job log indicating the total number of records sorted. |
| 5 | Store record count for each file in the returned records feedback and log record count messages. When output files are not used, a message is returned to the job log indicating the total number of records sorted. This option is a combination of options 1 and 4. |
| 6 | Store record count for each file in the returned records feedback and throw away duplicate keyed records. This option is a combination of options 2 and 4. |
| 7 | Store record count for each file in the returned records feedback, log record count messages, and throw away duplicate keyed records. This option is a combination of options 1, 2, and 4. |

Ordinal position of key field. The ordinal position of the key field relative to all fields within the record. This value must be set if sorting on a null-capable field.

Output member name. The name of the member in the output file. A value of *FIRST indicates that the first member of the file should be used. A value of *LAST indicates that the last member of the file should be used.

Qualified input file name. The name of a file whose data is to be sorted, and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name.

The library name can have the following special values:

*LIBL	The current library list.
*CURLIB	The job's current library.

All values must be padded with blanks to the right of the value.

If a file is specified that does not exist or is unavailable (for example, locked), an error is returned for that file and no further files are read. No sorted data is returned for any of the files already read without errors.

The QLGSORT API supports database (both logical and physical), diskette, and tape files. Any other file type is not recognized and an error is returned. Variable length record access and null-capable fields are supported for externally described database files only.

The data in each of the files is sorted as the common record length specified in the record length field. However, if the file contains variable length fields or uses variable length record access, the data is sorted using the actual length of the data. The data is truncated or padded with blanks as necessary when the record length of the input file is different than the record length value.

The total length of all the records in all the input files cannot exceed 16MB for the data to be returned in the output data buffer parameter. If your files have more data than this, you should consider having the output data sent to output files, or use the QLGSRTIO API to repeatedly retrieve sets of sorted data.

Qualified output file name. The name of a file that is to receive sorted data, and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name.

The library name can have the following special values:

*LIBL	The current library list.
*CURLIB	The job's current library.

All values must be padded with blanks to the right of the value.

If a file is specified that does not exist or is unavailable (for example, locked), a diagnostic error is returned indicating the error and the file in error. If an error occurs when the file is opened or put, a message is returned indicating the error and the file in error. For some I/O errors, an inquiry message is sent to the system operator message queue (QSYSOPR). Processing stops until a reply is received for the message. The QLGSORT API continues to put data to the other files that do not have errors. If none of the output files could be opened, an error message is returned indicating that the sort was successful but there were no files in which the sorted data could be returned.

The QLG SORT API supports database (both logical and physical), diskette, and tape files. Any other file type is not recognized and an error is returned, but the sort continues to provide output to any other output file types that are supported. Variable length record access and null-capable fields are supported for externally described database files only. The same file can be used both as an input and as an output file.

If the sorted output is returned in the output data buffer parameter, or returned using the QLG SRTIO API, the output file list is not used. If the number of output files field is greater than 0, there must be exactly that number of output files specified, or an error is returned.

Each output file contains the full sorted set of records unless an error occurred for a file. Record truncation or padding of blanks occurs when writing to the output file and when the file record length is different than the sorted record length. One reason to have more than one output file is to provide a backup or shadow of your output data as it is being sorted.

Qualified sort table name. The name of a sort table to be used for sorting the character data, and the library in which it is located. The first 10 characters contain the sort table name, and the second 10 characters contain the library name.

When a qualified sort table name is specified, the sort sequence language ID and sort sequence table fields are ignored.

The following special values are supported for the table name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

<i>*JOB</i>	The sort sequence of the job.
<i>*JOB RUN</i>	The API treats this value the same as <i>*JOB</i> .
<i>*LANGIDUNQ</i>	The unique-weight sort sequence table that is associated with the sort sequence language ID field.
<i>*LANGIDSHR</i>	The shared-weight sort sequence table that is associated with the sort sequence language ID field.
<i>*HEX</i>	The hexadecimal value of the characters.
<i>*TABLE</i>	The sort sequence table provided in the sort sequence table field.

Note: Both **JOB* and **JOB RUN* are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following special values are supported for the library name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

<i>*LIBL</i>	The user's library list.
<i>*CURLIB</i>	The current library.

If a special value is used for the table name, the second 10 characters used for the library name must be blank.

Record count. The number of records that exist in the input data buffer parameter when input data is to be sorted. Only the number of records specified in the record count field are sorted, even if more than this number exist in the input data buffer parameter. The record count multiplied by the record length must not be greater than 16MB.

The record count field must be greater than 0 when the type of request field is 4, 5, or 6 (input data to be sorted). If it is not, an error is returned.

The record count field must be set to 0 when the type of request field is 1, 2, 3, 7, or 8 because input files are being sorted or the data will be provided by the QLG SRTIO API. If it is not, an error is returned.

Record length. The record length that the sort uses for processing. When a data buffer is being sorted, this is the maximum length of a record in the buffer. The value includes the following:

- The bytes used to store the actual user data
- The 2-byte binary length for data type 19, 20, 21, or 22
- The 2-byte binary length of the user data when using variable length record access
- The null byte map if using null field support

For example, assume a user record contains five fields having a total length of 80 bytes, supports nulls, and is processed using variable length record access. The record length field would be 87. That is, 80 for the logical record, 2 for the length of the record, and 5 for the null byte map.

When a list of files is being sorted, this record length defines the length used for sorting all the files. If the data is to be returned in the output data buffer, the record count multiplied by the record length cannot be greater than 16MB.

If an input file has a record length that is longer than the specified record length, it is truncated. If an input file has a record length that is shorter, it is padded with blanks.

When data is being provided through the QLGSRATIO API, this record length defines the length used for sorting all the records. Each set of records added to the sort through the QLGSRATIO API defines a record length for that set. Records that are longer than the record length defined here are truncated. Records that are shorter are padded with blanks.

The following special value is defined:

- | | |
|---|--|
| 0 | QLGSORT assumes the maximum record length to be that of the first input file processed. If 0 is specified when an input data buffer is being sorted, or if data is being provided through calls to the QLGSRATIO API, an error occurs. |
|---|--|

Reserved. A reserved field. This field must be set to 0.

Returned records. The number of records returned in an output file.

Sort order. The order to be used for sorting.

The following special values are defined:

- | | |
|---|--------------------------------|
| 1 | Sort in an ascending sequence. |
| 2 | Sort in a descending sequence. |

Sort sequence CCSID. The CCSID to be used, along with the sort sequence language ID, for retrieving the national language sort sequence table for sorting the character data.

The valid values for this parameter are:

- | | |
|---------|---|
| 0 | The CCSID of the job is the CCSID of the sort sequence table to be used. |
| 1-65533 | A valid CCSID in this range must be specified or an error is returned. |
| 65535 | The CCSID of the sort sequence table that is found should be used. No CCSID conversion should be done on the found table. |

The following table summarizes when the CCSID value is required and when it will be ignored based on the value in the qualified sort table name field in the request control block.

Qualified Sort Table Name	CCSID
*JOB	Required
*JOBRUN	Required
*LANGIDUNQ	Required
*LANGIDSHR	Required
*HEX	Ignored
*TABLE	Ignored
Table name and library	Ignored

Sort sequence language ID. The language ID to be used to obtain a national language sort sequence table for sorting the character data. All values must be padded with blanks to the right of the value and must be in uppercase.

Possible values for this parameter are:

<i>*JOB</i>	The language identifier of the job.
<i>*JOBRUN</i>	The API treats this value the same as *JOB.
<i>Specific language identifier</i>	A valid 3-character language identifier. For example, Danish would be "DAN". See the globalization topic for a complete list of valid language identifiers.

Note: Both *JOB and *JOBRUN are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following table summarizes when the sort sequence language ID is required and when it is ignored based on the value in the qualified sort table name field in the request control block.

Qualified Sort Table Name	Language ID
*JOB	Required
*JOBRUN	Required
*LANGIDUNQ	Required
*LANGIDSHR	Required
*HEX	Ignored
*TABLE	Ignored
Table name and library	Ignored

Sort sequence table. The sort table to be used for sorting the character data. This field is only used when the qualified sort table name field is *TABLE. Otherwise, it is ignored. If UCS-2 data is to be sorted, use the QLGRTVSS API to retrieve a UCS-2 sort sequence table.

Type of request. The type of sort operation being requested. The following values are defined:

- 1 Input files are sorted, and the output goes to an output file.
- 2 Input files are sorted, and the output goes to the output data buffer parameter.
- 3 Input files are sorted, and the output is held for the QLGSRATIO API to retrieve one set at a time.
- 4 Data in an input data buffer is sorted, and the output goes to an output file.
- 5 Data in an input data buffer is sorted, and the output goes to the output data buffer parameter.
- 6 Data in an input data buffer is sorted, and the output is held for the QLGSRATIO API to retrieve one set at a time.
- 7 Input data is provided by the QLGSRATIO API, and the output goes to an output file.

A sort is initialized so that data can be provided through the QLSRTIO API one set at a time and retrieved one set at a time through the QLSRTIO API.

The following table summarizes the input source and output target for each type of request.

Type of Request	Source	Target
1	Input files	Output files
2	Input files	Output data buffer
3	Input files	QLSRTIO API calls
4	Input data buffer	Output files
5	Input data buffer	Output data buffer
6	Input data buffer	QLSRTIO API calls
7	QLSRTIO API calls	Output files
8	QLSRTIO API calls	QLSRTIO API calls

If the input files contain variable length fields, null fields, or use variable length record access, the only type of request allowed is 1.

If the input buffer contains variable length fields, null fields, or uses variable length record access, the only type of requests allowed are 5, 6, or 8.

Variable length record access. Whether variable length record access should be used for input or output files.

The possible values follow:

- 0 Do not use variable length record access.
- 1 Use variable length record access. This value is allowed only when using externally described database files. All fields but the last one in the file are defined as fixed length, and the last field is defined as variable length.

Buffer Layout Examples

The following examples show the layout of the buffer for fixed and variable length fields, null-capable fields, and variable length record access. The null byte map indicates whether the field is null (1) or contains data (0).

Buffer with Fixed Length Fields

If the sort includes none of the following, the record length field is set to the total number of bytes for all the fields:

- Variable length fields
- Null-capable fields
- Variable length record access

Following is an example of a buffer containing 4 fields that are fixed length. The record length and the fields used for the sort are indicated.

```
Field 1      Char 8
Field 2      Char 10
Field 3      Char 3
```

Field 4 Char 10

Record length = 31
Sort on fields 1 and 3

Field 1	Field 2	Field 3	Field 4
JJJJJJJ	A1B2C3D4E5	XXX	1111111111
FFF	A5B4C3D2E1	YY	222222
KKKKK	A1B2C3D4E5	ZZZ	33333333

Buffer with Variable Length Character Fields

This is an example of a buffer that contains variable length fields. The length of the each variable length field must be contained in a 2-byte binary value that precedes the data.

Field 1 Char 8
Field 2 Char 10
Field 3 Char 3
Field 4 is variable length character

Record length = 33 (31 bytes for the maximum length of the data plus 2 bytes for the field length)
Sort on fields 1 and 3

Field 1	Field 2	Field 3	Field 4	
Data	Data	Data	Length	Data
JJJJJJJ	A1B2C3D4E5	XXX	10	1111111111
FFF	A5B4C3D2E1	YY	6	222222
KKKKK	A1B2C3D4E5	ZZZ	8	33333333

Buffer with Variable Length Character Fields and Null-Capable Fields

If the buffer contains variable length fields and null-capable fields, the null byte map must follow the record. The offset to the null byte map is set in the offset to null byte map field. The null byte map contains one byte for each field. The bytes correspond to the field order. For example, for null byte map 0010, fields 1, 2, and 4 contain data and field 3 is null.

The position of each key must be set in the ordinal position of key field.

Field 1 Char 8
Field 2 Char 10
Field 3 is null capable Char 3
Field 4 is variable length character

Record length = 37 (31 bytes for the maximum length of the data plus 2 bytes for the field length of field 4 plus 4 bytes for the null byte map)
Sort on fields 1 and 3

Field 1	Field 2	Field 3	Field 4		Null Byte Map
Data	Data	Data	Length	Data	
JJJJJJJ	A1B2C3D4E5	XXX	10	1111111111	0000
FFF	A5B4C3D2E1	*NULL	6	222222	0010
KKKKK	A1B2C3D4E5	ZZZ	8	33333333	0000

Buffer with Variable Length Record Access and Null-Capable Fields

If the buffer contains variable length records, the actual length of the record must be added to the end of the data. It must precede the null byte map.

Field 1 Char 8
 Field 2 Char 10
 Field 3 is null capable Char 3
 Field 4 is variable length character

Record length = 37 (31 bytes for the maximum length
 of the data plus 2 bytes for
 the user record length plus
 4 bytes for the null byte map)

Sort on fields 1 and 3

Field 1	Field 2	Field 3	Field 4	Actual Record Length	Null Byte Map
Data	Data	Data	Data		
JJJJJJJ	A1B2C3D4E5	XXX	111111111	31	0000
FFF	A5B4C3D2E1	*NULL	222222	27	0010
KKKKK	A1B2C3D4E5	ZZZ	33333333	29	0000

Error Messages

Message ID	Error Message Text
CPF2115 E	Object &1 in &2 type *&3 damaged.
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3BC6 E	Sort sequence &1 not valid.
CPF3BC7 E	CCSID &1 outside of valid range.
CPF3BC8 E	Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E	Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3BCC E	Language identifier &1 not valid.
CPF3BD3 E	Sort already active.
CPF3BD5 E	Sort request control block field &1 is not valid.
CPF3BD6 E	Key field &1 is not valid for key number &2.
CPF3BD7 E	Key size exceeds maximum.
CPF3BD8 E	Input file entry &3 field &4 is not valid.
CPF3BD9 E	Output file entry &3 field &4 is not valid.
CPF3BDA E	No output files could be opened.
CPF3BDB E	Sort internal storage limit exceeded.
CPF3BDC E	Duplicate keys encountered for file &1 in library &2.
CPF3BE2 E	Returned record feedback length not valid.
CPF3BE4 E	Buffer information is not valid. Reason &1.
CPF3BE6 E	Number of parameters, &1, entered for this API was not valid.
CPF3BE7 E	File &1 in library &2 or file entry &4 has error &3.
CPF3BF0 E	Sort sequence table specified or supplied is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5715 E	File &1 in library &2 not found.
CPF9504 E	An invalid search order was specified.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.

Message ID	Error Message Text
CPF9845 E	Error occurred while opening file &1.
CPF9846 E	Error while processing file &1 in library &2.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "National Language Support APIs," on page 1 | APIs by category

Sort Input/Output (QLGSRTIO) API

Required Parameter Group:	
1	Request control block
Input	Char(16)
2	Input data buffer
Input	Char(*)
3	Output data buffer
Output	Char(*)
4	Length of output data buffer
Input	Binary(4)
5	Output data information
Output	Char(16)
6	Error code
I/O	Char(*)
Optional Parameter Group:	
7	Returned records feedback
Output	Char(*)
8	Length of returned records feedback
Input	Binary(4)
	Default Public Authority: *USE
Threadsafe: No	

The Sort Input/Output (QLGSRTIO) API is used to provide a set of records at a time to be sorted and to return a set of records at a time that have already been sorted. This API can only be used after the QLGSORT API has been called to initialize the sort function. This initialization defines such things as the length of data to be sorted and the fields to be sorted. An error is returned if QLGSRTIO is called without QLGSORT having been called for initializing.

After the initialization is complete, the QLGSRTIO API can be called repeatedly to add a set of records to be sorted. When all of the records have been provided, the QLGSRTIO API must be called with an end put request, which causes the added records to be sorted. Once the records are sorted, the QLGSRTIO API can be called repeatedly to return sets of sorted records until no more records can be returned or the application has determined that it no longer wants sorted records.

The QLGSRATIO API can also be used to provide sets of records at a time to the sort and have the output from the sort go to output files. The type of request in the request control block on the QLGSRATIO API can be set to specify that the output from the sort is output files.

The QLGSRATIO API provides a function to cancel the sort at any time. When the sort is canceled, all work areas initialized for the sort are deactivated, so the QLGSRATIO API can be called again to perform another sort. If the QLGSRATIO API is being used to return records from a sort a set at a time, the QLGSRATIO API automatically ends the sort when a get request is made and all of the sorted records have been returned. If all of the records are not retrieved, a cancel is required to complete this sort if another call to the QLGSRATIO API will be made within this job. If a cancellation is not requested, the sort is still considered active and an error is returned on any future calls to the QLGSRATIO API within this job. When the job ends, the sort automatically ends. If the QLGSRATIO API is only being used to add records to the sort a set at a time, with the output going to output files, the end put request causes the data to be sorted and put to the output files, and the sort ends. No additional cancel request is needed.

Authorities and Locks

The locks needed for this API are set in the QLGSRATIO API and remain in effect until a cancel request is made by the QLGSRATIO API or until an end put request is made to return the sorted data to output files.

Required Parameter Group

Request control block

INPUT; CHAR(16)

Whether data is being put to the sort or retrieved from the sort. This parameter also defines information about the input and output data. See "Format of Request Control Block" on page 93 for details.

Input data buffer

INPUT; CHAR(*)

The data to be added to the sort. This parameter is only used on a put request. It is ignored for end put, get, and cancel requests. See the "Buffer Layout Examples" on page 88 for information on how to format this buffer. If the call to QLGSRATIO specified that input files or an input data buffer were to be sorted and if QLGSRATIO is then called specifying a put request and input data, an error is returned.

Output data buffer

OUTPUT; CHAR(*)

The sorted data being returned to the application. This parameter is only used on a get request. It is ignored for put, end put, and cancel requests. If the call to the QLGSRATIO API specified that files were to be returned and the QLGSRATIO API is then called specifying a get request, an error is returned. This parameter can be the same as the input data buffer parameter. The format of the data in the buffer is the same as for the input data buffer.

Length of output data buffer

INPUT; BINARY(4)

The maximum amount of data that can be returned from the sort on a get request. This value must be greater than 0 or an error is returned. QLGSRATIO returns only the number of records specified in the request control block parameter, but only up to the size of the output data buffer. The length of the output data buffer must be at least as large as the record length value specified in the request control block on the QLGSRATIO API call. That length specified the record length to be used by the sort. If this output data length is not large enough to hold at least one record, an error is returned. The maximum length allowed is 16MB less 512 bytes.

Output data information

OUTPUT; CHAR(16)

Information to be returned to the calling program as a result of a put or get request. See “Format of Output Data Information” for details on the format.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Optional Parameter Group

Returned records feedback

OUTPUT; CHAR(*)

The number of records added to each output file to be returned to the calling program. The returned records feedback parameter contains the number of records added to each output file.

The number of records added to each output file corresponds to the list of files specified in the request control block.

This parameter is used only for sorted data returned in an output file for the following:

- Request type 7 to the Sort (QLGSORT) API
- Request type 2 (End Put) to the QLGSRTIO API

To receive results, set the options field in the request control block for the QLGSORT API to 4, 5, 6, or 7.

Length of returned records feedback

INPUT; BINARY(4)

The length of the returned records feedback parameter to be returned to the calling program. If this length is larger than the actual size of the returned records feedback parameter, the results may not be predictable. The minimum length is 8.

Format of Request Control Block

For a description of the fields in this format, see “Field Descriptions” on page 94.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Type of request
4	4	BINARY(4)	Reserved
8	8	BINARY(4)	Record length
12	C	BINARY(4)	Record count

Format of Output Data Information

For a description of the fields in this format, see “Field Descriptions” on page 94.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of records processed
4	4	BINARY(4)	Number of records available
8	8	CHAR(8)	Reserved

Format of Returned Records Feedback Information

For a description of the fields in this format, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	BINARY(4)	Offset to start of returned records array
12	C	BINARY(4)	Number of output files returned
16	10	CHAR(*)	Reserved
		ARRAY of BINARY(4)	Returned records

Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Number of output files returned. The number of output files specified in the request control block of the QLGSORT API.

Number of records available. For a put request, the total number of records that have been put (are available) for sorting. For the first put request, this will equal the number of records processed. After the second put request, it will contain the total of the first two put requests. An application might find this helpful if it only wants to sort a maximum number of records without counting the number from each put.

For a get request, this field defines the total number of records that have not been returned to the application. An application can use this information to determine an appropriate time to end if it does not intend to process all of the sorted records.

Number of records processed. For a put request, the number of records actually put to the sort function from the input data buffer parameter. This value is always the same as the record count specified in the request control block.

For a get request, this field defines the number of records actually returned in the output data buffer parameter. This value never exceeds the record count specified in the QLGSRTIO request control block parameter. If more records are requested than are available, the records processed count is set to the actual number returned. The number of records processed multiplied by the record length must not be greater than 16MB.

Offset to start of returned records array. Offset to the returned records array.

Record count. For a put request, the number of records that are being provided to the sort. The record count must be greater than 0 or an error is returned. Only the number of records specified by this field are processed even if more records are provided in the input data buffer parameter.

For a get request, this field defines the maximum number of records that can be returned to the application. The number of records processed field in the output data information parameter indicates the actual number of records returned.

Using record counts greater than 1 can improve performance.

Record length. The size of each record given to the sort or the size of each record returned from the sort. For a put request, each record retrieved from the input data buffer must have the length specified by the record length field.

This value includes the following:

- The bytes used to store the length of any variable length fields
- Variable length record access information
- Any null byte map

For example, assume a logical record equals 80, and the record is variable length with three of the five fields null capable. The record length would be 87. That is, 80 for the logical record, 2 for the length of the record, and 5 for the null byte map.

The offsets to the variable length record access information and the null byte map are specified in the QLGSORT API request control block.

When data is provided through this API, the record length specified in the QLGSORT API defines the length used for sorting all the records. The record length specified here defines the record length for the current set of data to be sorted. If the input data record length is shorter than the sort record length, each of the records is padded with blanks. If the input data record length is longer than the sort record length, each input record is truncated.

On a get request, each record is put to the output data buffer parameter with the specified record length. If the sort record length is longer, each record is truncated. If the sort record length is shorter, each record is padded with blanks. The number of records processed multiplied by the record length cannot be greater than 16MB.

Reserved. A reserved field. This field must be set to 0 in the request control block. In the output information, the reserved character field is set to blanks.

Returned records. The number of records returned in an output file.

Type of request. The type of operation being requested. The following values are defined:

- | | |
|----------------------|--|
| 1 (<i>Put</i>) | Input data is provided to the sort function where it is stored by the sort program in internal buffers until an end put request is made and the data is sorted. One or more calls to the API with the put request can be done until all of the data to be sorted has been provided to the sort program.

A put request is only valid after a call to the QLGSORT API is done to initialize a sort for input from QLGSRTIO and before an end put request is made. If a put request is made at any other time, an error is returned. |
| 2 (<i>End put</i>) | All of the input data has been added to the sort, and the data should now be sorted. This operation must be requested before a get request is made, or an error is returned.

An end put request causes the input to the sort to be complete. No additional put requests can be made to the QLGSRTIO API without first doing an initialization through the QLGSORT API; otherwise, an error is returned. |

3 (*Get*) Sorted data is returned to the caller. One or more calls to the API with the get request can be done until all of the data has been retrieved or no further data is needed. If no further get requests are made but data is still available to be retrieved, a cancel request must be made to end the sort.

A get request is valid for each of the following conditions:

- After an end put request was done (to cause the data to be sorted)
- After a previous get request
- If the sort was done on input files or on an input data buffer using sort request type 3 or 6 of the QLGSORT API

4 (*Cancel*) If a get request is made at any other time, an error is returned. The sort should be canceled immediately. All internal buffers are cleared and deleted. This operation can be requested at any time. If requested after a put request, no sorting is performed. If requested after a get request, no further sort records are returned. The sort is automatically ended when a get request is made and no data is available to be returned. If all of the data was not retrieved, a cancel request should always be made. If this is not done and another sort request is made using the QLGSORT API, an error occurs because a sort is already active.

Error Messages

Message ID	Error Message Text
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3BD0 E	QLGSRTIO request control block field &1 is not valid.
CPF3BD1 E	Output data length parameter is not valid.
CPF3BD2 E	Type of request &1 is not valid at this time.
CPF3BD7 E	Key size exceeds maximum.
CPF3BDA E	No output files could be opened.
CPF3BDB E	Sort internal storage limit exceeded.
CPF3BDC E	Duplicate keys encountered for file &1 in library &2.
CPF3BF0 E	Sort sequence table specified or supplied is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3BE2 E	Returned record feedback length not valid.
CPF3BE4 E	Buffer information is not valid. Reason &1.
CPF3BE7 E	File &1 in library &2 or file entry &4 has error &3.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5715 E	File &1 in library &2 not found.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9845 E	Error occurred while opening file &1.
CPF9846 E	Error while processing file &1 in library &2.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

QlgTransformUCSData()—Transform UCS Data API

Syntax

```
#include <qlgusr.h>
int QlgTransformUCSData(int xformtype,
                        char  **inbuf,
                        size_t *inbytesleft,
                        char  **outbuf,
                        size_t *outbytesleft,
                        size_t *outspacereg);
```

Service Program: QLGUSR

Default Public Authority: *USE

Threadsafe: Yes

The **QlgTransformUCSData()** function transforms, through a formula as compared to a mapping, data from one form of Unicode to another. A transformation type identification is used to specify the type of transformation.

Parameters

xformtype

(Input) The type of transformation requested for execution (that is, UCS2-UTF8, UTF8-UCS2). The *xformtype* parameter must be one of the following values:

- 1 Transform from UCS-2 to UTF-8.
- 2 Transform from UTF-8 to UCS-2.

Other transformation types are one of the following allowable values from the table of transformation types. The first three decimal digits represent the from encoding and the last three decimal digits represent the to encoding. For instance, 030042 is a transform from UTF-32 LE(Little Endian) (030) to UTF-16 BE(Big Endian) without a BOM(Byte Order Mark) (042).

Allowable decimal values:

010021, 010022, 010031, 010032, 010041, 010042, 010051, 010052, 010061, 010062, 020021, 020022, 020031, 020032, 020041, 020042, 020051, 020052, 020061, 020062, 030021, 030022, 030031, 030032, 030041, 030042, 030051, 030052, 030061, 030062, 040021, 040022, 040031, 040032, 040041, 040042, 040051, 040052, 040061, 040062, 050021, 050022, 050031, 050032, 050041, 050042, 050051, 050052, 050061, 050062, 060021, 060022, 060031, 060032, 060041, 060042, 060051, 060052, 060061, 060062

Table of transformation types:

Transformation	From:	To: with BOM	To: without BOM
Autodetect	010		
UTF-32 BE	020	021	022
UTF-32 LE	030	031	032
UTF-16 BE	040	041	042
UTF-16 LE	050	051	052
UTF-8	060	061	062

inbuf (Input) A pointer to a variable (pointer) that points to the first character in the input buffer. The variable (pointer) is updated to point to the byte following the last byte successfully used in the transformation. The maximum size of the input buffer is 16773104.

inbytesleft

(Input) A pointer to a variable containing the number of bytes to the end of the input buffer to be transformed. This variable is decremented to reflect the number of bytes still not transformed in the input buffer. The maximum number of bytes that can be transformed is 16773104.

outbuf

(Input) A pointer to a variable (pointer) that points to the first character in the output buffer. This variable (pointer) is updated to point to the byte following the last byte of transformed output data. The maximum size of the output buffer is 16773104.

outbytesleft

(Input) A pointer to a variable containing the number of bytes to the end of the output buffer. This variable is decremented to reflect the number of bytes still available in the output buffer. The maximum number of bytes that can be transformed is 16773104.

outpacereq

(Input) A pointer to a variable containing the size of the output buffer required to transform the remaining portion of the input buffer. This value will not be accurate if a sequence error (EILSEQ) occurs, either independently or following an E2BIG error.

Authorities and Locks

None

Return Value

0

QlgTransformUCSDData() was successful. The entire 'inbuf' was transformed.

[E2BIG]

There was insufficient space in the output buffer. As much of the input buffer as can be contained in the output buffer is transformed. The buffer pointers and byte counters reflect the point at which a complete character could not be transformed into the output buffer.

[EBADFUNC]

An invalid transform type request was given on the first parameter, *xformtype*.

[EFAULT]

The address used for an argument is not correct. In attempting to use an argument in a call, the system detected an address that is not valid. While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EILSEQ]

Data provided on an Unicode input buffer was found with an incorrect encoding as determined by the sequence or ordering of the bytes, or a requested *xformtype* from type did not match the data of the *inbuf*. Use *inbytesleft* to determine where the error occurred in the sequence.

[EINVAL]

An invalid *inbytesleft* data length was given. More specifically, an uneven number of bytes were provided when transforming from UCS-2.

[ENOMEM]

There was not enough memory to perform the transformation.

[ENOTSUP]

Operation not supported. Automatic detection could not find a BOM. A transformation will not be performed.

Usage Notes:

1. When requesting a BOM tagged output buffer the following BOM values will be generated.

BOMS	Hex values
UTF-32 BE (020)	00 00 FE FF
UTF-32 LE (030)	FF FE 00 00
UTF-16 BE (040)	FE FF
UTF-16 LE (050)	FF FE
UTF-8 (060)	EF BB BF

A sample conversion:

transformation type: 030021

UTF-32 LE (030) -> UTF-32 BE marked (021)

x'AB 5F 00 00 7C 8E 00 00' -> x'00 00 FE FF 00 00 5F AB 00 00 8E 7C'

2. Auto detection is available when there is a BOM in the beginning of the *inbuf* that represents a Unicode type.

Error Messages

None.

API introduced: V4R5

[Top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

Truncate Character Data (QLGTRDTA) API

Required Parameter Group:	
1	Output data buffer
Output	Char(*)
2	Length of output data returned
Output	Binary(4)
3	Remaining data buffer
Output	Char(*)
4	Remaining data length
Output	Binary(4)
5	Input data buffer
Input	Char(*)
6	Length of buffers
Input	Binary(4)
7	Truncate length
Input	Binary(4)
8	CCSID of input data
Input	Binary(4)
9	Error code
I/O	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Truncate Character Data (QLGTRDTA) API truncates a CCSID-tagged string of character data to a requested length. This API is used to truncate a string of data properly. The string is truncated with respect to the encoding of the data using the encoding scheme of the CCSID. A typical use of this API would be to properly truncate mixed-byte strings for use in a display screen.

This API works in the following manner:

1. Given the truncate length requested for the result, truncate the input data in a proper manner to no longer than this value. If the length of the input data buffer is less than the requested truncate length, the returned length is set to the input length and the data in the input data buffer is placed in the resultant output data buffer.

Proper manner means that no multibyte characters are split and no control information is lost. In the situation where characters would be split, the truncation point is re-adjusted to an acceptable position to allow truncation. If control characters are required to be appended to the output, adjustment to the output may be done. For example, if the requested string is in the ISO-2022 encoding scheme, the truncate length requested is reduced by three. This allows for appending the ASCII single-byte escape sequence ('1B2842'X) to the output data returned.

2. Place this result into the requested output data buffer and set its length in the length of output data returned parameter. Pad the rest of this parameter with the appropriate blank characters. When the length of input is less than the truncate length, all data is placed in the output data buffer because the condition would call for no truncation.
3. Place the remaining input data (properly formed) into the remaining data buffer and set its length in the remaining data length parameter. Properly formed data refers to providing a valid string for the type of data being used. For example, mixed EBCDIC data must have matched control characters (X'0E' and X'0F') around any double-byte character data.

Required Parameter Group

Output data buffer

OUTPUT; CHAR(*)

The buffer that receives the properly formed truncated string. This parameter must be the same size as the value specified for the length of buffers parameter.

Length of output data returned

OUTPUT; BINARY(4)

The number of bytes of data actually returned in the resultant output buffer.

Remaining data buffer

OUTPUT; CHAR(*)

The buffer that receives the data remaining after the truncation has been done. This buffer must be the same size as the input data buffer.

Remaining data length

OUTPUT; BINARY(4)

The length of the data remaining after the truncation has been done. This data is the length of the remaining data buffer.

Input data buffer

INPUT; CHAR(*)

The buffer that holds the input data.

Length of buffers

INPUT; BINARY(4)

The length of each of the following data buffers:

- Input data buffer
- Output data buffer
- Remaining data buffer

The maximum size of each data buffer is 32767 bytes.

Truncate length

INPUT; BINARY(4)

The maximum length of data to be returned in the output data buffer. Valid values are 1 through 32767.

CCSID of input data

INPUT; BINARY(4)

The CCSID of the data to be truncated. Valid values are 0 through 65533 and 65535. If the CCSID value 0 is provided, the job's default CCSID is used.

If the CCSID tag of the data is not known, a value of 65535 should be used. If the CCSID value is 65535, the data is assumed to be a mixed-byte EBCDIC string that is properly formed.

The supported encoding schemes are single-byte data, mixed-byte EBCDIC data, double-byte data, PC mixed data, ISO-2022 data, and extended UNIX coded-character set (EUC) data. The following specific encoding schemes are supported:

<i>Single-byte data</i>	'1100'X, '2100'X, '3100'X, '4100'X, '4105'X, '4155'X, '5100'X, '5150'X, '6100'X
<i>Double-byte data</i>	'1200'X, '2200'X, '3200'X, '5200'X, '7200'X
<i>Mixed EBCDIC data</i>	'1301'X
<i>Mixed PC data</i>	'2300'X, '3300'X
<i>ISO-2022 data</i>	'5404'X
<i>EUC data</i>	'4403'X

See the Globalization topic in the iSeries Information Center for more information on CCSID values and encoding schemes.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID Error Message Text

CPF24B4 E	Severe error while addressing parameter list.
CPF2647 E	Buffer length not valid.
CPF3BC7 E	CCSID &1 outside of valid range.
CPF3BCA E	CCSID &1 not supported.
CPF3BCB E	Encoding scheme &1 of CCSID &2 not supported.
CPF3BCF E	Truncate length not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

[Top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

QtqValidateCCSID()—Validate CCSID API

Syntax

```
int QtqValidateCCSID (CCSID)
    int      CCSID;
```

Service Program: QTQUSF

Default Public Authority: *USE

Threadsafe: No

The **QtqValidateCCSID()** function determines if the specified CCSID is supported by the iSeries server. If the CCSID is supported, the encoding scheme for the CCSID is returned.

Parameters

CCSID

INPUT

An integer that represents the CCSID to be validated. Valid CCSID values are in the range 1 through 65535.

Return Value

If successful, **QtqValidateCCSID()** returns a positive integer that represents the encoding scheme of the specified CCSID. For more information on encoding schemes, see the Globalization topic in the iSeries Information Center.

If unsuccessful, **QtqValidateCCSID()** returns one of the following values:

- 0** Special purpose.
The input CCSID is a special-purpose CCSID as defined by CDRA.
- 1** CCSID not found.
The input CCSID is not recognized by the system.
- 2** Out of range.
The input CCSID is out of the supported range of 1 to 65535.

API introduced: V3R7

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Validate Language ID (QLGVlid) API

b	Required Parameter Group:
1	Language ID to validate
Input	Char(3)
2	Error code
I/O	Char(*)
	Default Public Authority: *USE
	Threadsafe: Yes

The Validate Language ID (QLGVlid) API validates a language identifier to ensure it is supported. If the language identifier is valid, no error is returned.

Required Parameter Group

Language ID to validate

INPUT; CHAR(3)

The language identifier to validate. The identifier must be in uppercase.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF3BCC E	Language identifier &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

Data Conversion APIs

The data conversion APIs are:

- “[iconv\(\)](#)—Code Conversion API” on page 105 ([iconv\(\)](#)) converts a buffer of characters from one coded character set identifier (CCSID) into another CCSID.
- “[iconv_open\(\)](#)—Code Conversion Allocation API” on page 109 ([iconv_open\(\)](#)) performs the necessary initializations to convert character encodings and returns a conversion descriptor of type `iconv_t`.
- “[QtqIconvOpen\(\)](#)—Code Conversion Allocation API” on page 113 ([QtqIconvOpen\(\)](#)) performs the necessary initializations to convert character encodings and returns a conversion descriptor.
- “[iconv_close\(\)](#)—Code Conversion Deallocation API” on page 116 ([iconv_close\(\)](#)) closes the conversion descriptor `cd` that was initialized by the [iconv_open\(\)](#) or [QtqIconvOpen\(\)](#) function.
- “[Convert Data \(QDCXLATE\) API](#)” on page 118 ([QDCXLATE](#)) converts data through the use of a table object.

The following code conversion APIs are designed to meet the X/Open industry standard functions (formerly Spec 1170).

- [iconv\(\)](#)
- [iconv_open\(\)](#)
- [iconv_close\(\)](#)

Note: The [QtqIconvOpen\(\)](#) function does not meet the X/Open industry standard.

The three-step conversion provided by the code conversion APIs allows applications to:

- Open a conversion descriptor with a specified CCSID pair ([iconv_open\(\)](#) or [QtqIconvOpen\(\)](#) function)
- Do multiple conversions ([iconv\(\)](#) function)
- Close the conversion descriptor when done ([iconv_close\(\)](#) function)

This reduces the overhead for applications that need to do multiple conversions using the same CCSID pairs. These functions are carried out on the iSeries^(TM) system as entry points in a bindable Integrated Language Environment (ILE) service program.

Note: The CDRA API, “[Convert a Graphic Character String \(CDRCVRT, QTQCVRT\) API](#)” on page 123 ([CDRCVRT](#)), converts a graphic character data string from one CCSID to another CCSID.

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

iconv()—Code Conversion API

Syntax

```
#include <iconv.h>

size_t iconv (cd, inbuf, inbytesleft,
              outbuf, outbytesleft)

    iconv_t    cd;
    char       **inbuf;
    size_t     *inbytesleft;
    char       **outbuf;
    size_t     *outbytesleft;
```

Service Program: QTQICONV

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 108.

The **iconv()** function converts a buffer of characters specified by the *inbuf* parameter from one coded character set identifier (CCSID) into another CCSID and stores the converted characters into a buffer specified by the *outbuf* parameter. (The *inbuf* parameter points to a variable that points to the first character in the input buffer. The *outbuf* parameter points to a variable that points to the first available byte in the output buffer.) The CCSIDs used are those in the conversion descriptor, *cd*, which was returned from the call to either the **iconv_open()** or the **QtqIconvOpen()** function.

On input, the *inbytesleft* parameter indicates the number of bytes in *inbuf* to be converted. Similarly, the *outbytesleft* parameter indicates the number of bytes available in *outbuf*. These values are decremented when the conversion is done, such that on return they indicate the state of their associated buffers. For encodings dependent on shift state, **iconv()** changes the shift state of the conversion descriptor to match the shift state at the end of the input buffer. For subsequent calls to **iconv()**, conversion begins using the current shift state of the conversion descriptor. The only state-dependent encodings in which **iconv()** supports the updating of the conversion descriptor shift state is mixed-byte EBCDIC.

For encodings dependent on shift state, the conversion descriptor can be returned to its initial shift state by calling **iconv()** with *inbuf* equal to a null pointer, or with *inbuf* pointing to a null pointer. The conversion descriptor can also be set to always return to its initial shift state by specifying the appropriate shift state alternative on the **iconv_open()** and **QtqIconvOpen()** APIs. When **iconv()** is called with the conversion descriptor set in this way, **iconv()** begins conversion from the initial shift state.

If the input buffer ends with an incomplete character or shift sequence, conversion stops after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow.

During conversion, **iconv()** may encounter valid characters in the input buffer that do not exist in the target CCSID. This is known as a character mismatch. In this case, **iconv()** performs the conversion based on the conversion alternative specified on the **iconv_open()** function.

Parameters

cd INPUT

The conversion descriptor returned by the `iconv_open()` or `QtqIconvOpen()` function that represents the following:

- CCSIDs to convert from and to
- The conversion alternative to use for character mismatches
- The substitution alternative
- The shift-state alternative
- The input length option
- The error option for mixed data

inbuf I/O

A pointer to a variable (pointer) that points to the first character in the input buffer. The variable (pointer) is updated to point to the byte following the last byte successfully used in the conversion. The maximum size of the input buffer is 16 773 104 bytes.

inbytesleft

I/O

A pointer to a variable containing the number of bytes to the end of the input buffer to be converted. This variable is decremented to reflect the number of bytes still not converted in the input buffer. The maximum number of bytes that can be converted is 16 773 104.

outbuf

OUTPUT

A pointer to a variable (pointer) that points to the first available byte in the output buffer. This variable (pointer) is updated to point to the byte following the last byte of converted output data. The maximum size of the output buffer is 16 773 104.

outbytesleft

I/O

A pointer to a variable containing the number of the available bytes to the end of the output buffer. This variable is decremented to reflect the number of bytes still available in the output buffer. The maximum number of bytes available is 16 773 104.

Restrictions

If an error occurs during one of the following requested conversions, the *inbuf*, *inbytesleft*, *outbuf*, and *outbytesleft* parameters may not be updated properly. (ES means encoding scheme.)

- EBCDIC mixed-byte (ES X'1301') to or from ASCII mixed-byte (ES X'2300')
- EBCDIC mixed-byte (ES X'1301') to or from ASCII double-byte (ES X'2200')
- EBCDIC double-byte (ES X'1200') to or from ASCII mixed-byte (ES X'2300')
- EBCDIC double-byte (ES X'1200') to or from ASCII double-byte (ES X'2200')

If the input length option field (on the call to `iconv_open()` or `QtqIconvOpen()`) is set for a NULL-terminated input buffer and the conversion completes successfully, the *inbuf* parameter is not updated for conversions involving the following encoding schemes:

- Encoding scheme X'2300'
- Encoding scheme X'4100'
- Encoding scheme X'4403'
- Encoding scheme X'5100'
- Encoding scheme X'5200'
- Encoding scheme X'5404'
- Encoding scheme X'5405'
- Encoding scheme X'5700'

If the input length option field (on the call to `iconv_open()` or `QtqIconvOpen()`) is set for a NULL-terminated input buffer and an error occurs during the conversion, the *inbytesleft* parameter is not updated for conversions involving the following encoding schemes:

- Encoding scheme X'2300'
- Encoding scheme X'4100'
- Encoding scheme X'4403'
- Encoding scheme X'5100'
- Encoding scheme X'5200'
- Encoding scheme X'5404'
- Encoding scheme X'5405'
- Encoding scheme X'5700'

Return Value

If the entire input buffer is successfully converted, `iconv()` may return the number of nonidentical conversions performed based on the substitution alternative. See “`iconv_open()`—Code Conversion Allocation API” on page 109 and “`QtqIconvOpen()`—Code Conversion Allocation API” on page 113. Otherwise, zero will be returned. If an error occurs, `iconv()` returns -1 in the return value, and sets `errno` to indicate the error.

Error Conditions

The following errors can be returned in `errno`:

[E2BIG]

Insufficient space.

Conversion stopped due to lack of space in the output buffer or there was not enough space to store the NULL character in the output buffer.

[EBADDATA]

Shift state not valid in input data

The beginning shift state of the input data buffer does not correspond to the shift state of the conversion descriptor. A shift-state sequence was encountered that tried to change the shift state of the input buffer to the current shift state of the conversion descriptor. For example, an EBCDIC shift-in control character may have been encountered while the conversion descriptor indicated single-byte state. This error is only supported for EBCDIC mixed-byte (X'1301') encoding schemes.

[EBADF]

Descriptor not valid.

The conversion descriptor (`cd`) parameter is not valid.

[ECONVERT]

The mixed input data contained DBCS characters.

Input conversion stopped due to the occurrence of DBCS characters in the input data when converting from a mixed-byte encoding scheme. The shift state for EBCDIC mixed data remains in the initial single-byte shift state. This error can only be returned when the mixed error option has been set accordingly for the `QtqIconvOpen()` or `iconv_open()` function.

This error is supported only for the following conversions:

- EBCDIC mixed-byte (encoding scheme X'1301') to any single-byte encoding scheme
- ASCII mixed-byte (encoding scheme X'2300') to any single-byte encoding scheme
- EBCDIC mixed-byte (encoding scheme X'1301') to EBCDIC mixed-byte (encoding scheme X'1301')

[EFAULT]

Bad address

The system detected an address that was not valid when attempting to use an argument from the parameter list. An escape message may also be signaled as a result.

[EINVAL]

Parameter not valid.

The conversion stopped because of an incomplete character or shift state sequence at the end of the input buffer.

[ENOBUFS]

Number of bytes for the input or output buffer not valid, or the input length cannot be determined.

The specified number of bytes for *inbytesleft* or *outbytesleft* is not valid. If the input length option field (on the call to `iconv_open()` or `QtqIconvOpen()`) specifies that `iconv()` determines the length of the input buffer and if `iconv()` cannot find a NULL character in the input buffer, this error could be returned.

[ENOMEM]

Not enough space

Insufficient storage space was available to perform the conversion.

[EUNKNOWN]

Undetected error

An undetected error occurred. Contact your service organization. An escape message may also be signaled as a result.

The following escape messages can be signaled:

Message ID	Escape Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Usage Notes

This API is threadsafe if threads that share a conversion descriptor do not attempt to preserve the shift state.

Related Information

- “`iconv_open()`—Code Conversion Allocation API” on page 109—Code Conversion Allocation API
- “`QtqIconvOpen()`—Code Conversion Allocation API” on page 113—Code Conversion Allocation API
- “`iconv_close()`—Code Conversion Deallocation API” on page 116—Code Conversion Deallocation API

API introduced: V3R1

[Top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

iconv_open()—Code Conversion Allocation API

Syntax

```
#include <iconv.h>

iconv_t iconv_open (tocode, fromcode)

        char      *tocode;
        char      *fromcode;
```

Service Program: QTQICONV

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 112.

The **iconv_open()** function performs the necessary initializations to convert character encodings from the source CCSID identified by the *fromcode* parameter to the CCSID identified by the *tocode* parameter. It then returns a conversion descriptor of data type `iconv_t`. For EBCDIC mixed-byte encodings, the conversion descriptor is set to the initial single-byte shift state.

Note: This API performs the same function as the **QtqIconvOpen()** API except that the input types of *fromcode* and *tocode* are character strings.

The conversion descriptor remains valid in a job until that job closes it with **iconv_close()** or the job ends. Keeping unneeded conversion descriptors active results in storage being allocated for these. To reduce this storage, you should call the **iconv_close()** API if you no longer need this conversion descriptor. The number of active descriptors is limited to a maximum of 104 000 active descriptors per process. This number may be reduced if the job has many active threads.

Parameters

tocode INPUT

A pointer to a string containing the CCSID to convert to. Refer to Format of *fromcode* and *tocode* for the layout of this string.

fromcode
INPUT

A pointer to the string containing the CCSID to convert from, the type of conversion to perform, the substitution alternative, the shift-state alternative, the input length option, and the mixed error option. Refer to Format of *fromcode* and *tocode* for the layout of this string.

Format of fromcode and tocode

The *fromcode* string consists of the following.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(8)	The word IBMCCSID
8	8	CHAR(5)	Character representation of CCSID number

Offset		Type	Field
Dec	Hex		
13	D	CHAR(3)	Conversion alternative
16	10	CHAR(1)	Substitution alternative
17	11	CHAR(1)	Shift-state alternative
18	12	CHAR(1)	Input length option
19	13	CHAR(1)	Error option for mixed data
20	14	CHAR(12)	Reserved

The *to code* string consists of the following.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(8)	The word IBMCCSID
8	8	CHAR(5)	Character representation of CCSID number
13	D	CHAR(19)	Reserved

Field Descriptions

Character representation of CCSID number. A character representation of the CCSID number to convert from or to. Valid CCSID values are in the range 00001 through 65533. The following special value is supported on the *fromcode* and *to code* parameters:

00000

The conversion descriptor is created with the CCSID of the current job such that calls to **iconv()** with the conversion descriptor use the CCSID of the job at the time the conversion descriptor was opened by **iconv_open()**. If the CCSID of the current job is 65535 (indicating no conversion), the default CCSID from the DFTCCSID job attribute is used.

Conversion alternative. The conversion alternative that is selected to convert graphic character data. This value is only used on the *fromcode* parameter. The following values can be used:

000

The IBM-defined default conversion method and the associated conversion tables. Most of the default tables follow the round-trip conversion criterion. For the default tables that do not follow the round-trip conversion criterion, see the Globalization topic in the iSeries Information Center.

057

The enforced subset match (substitution) criterion. For the CCSID conversion pairs that support this criterion, refer to the Globalization topic in the iSeries Information Center.

102

The best-fit conversion criterion for character mismatch.

Error option for mixed data. Whether **iconv()** returns an error when it converts a character string from a mixed-byte encoding scheme and the input buffer contains double-byte character set (DBCS) characters.

- 0 An error is not returned when converting from a mixed-byte encoding scheme and the input buffer contains DBCS characters. Instead, **iconv()** converts the DBCS characters to the CDRA-standardized control character substitute of the target CCSID.
- 1 An ECONVERT error is returned in **errno** when **iconv()** encounters a DBCS character in the input buffer when converting from a mixed-byte encoding scheme.

Input length option. Whether **iconv()** determines the number of bytes to the end of the input buffer being converted. This value is only used on the *fromcode* parameter. The possible values follow:

- 0 The conversion descriptor is created such that the **iconv()** function does not determine the number of bytes to the end of the input buffer being converted; therefore, a valid value for the *inbytesleft* parameter must be specified.
- 1 The conversion descriptor is created such that the **iconv()** function determines the number of bytes to the end of the input buffer being converted; therefore, the input buffer must end with a NULL character. The resulting converted output buffer is ended with a NULL character by **iconv()**. The *inbytesleft* parameter must point to a value of zero.

Reserved. A reserved field that must be set to hexadecimal zeros.

Shift-state alternative. Whether the shift state should be restored to its initial shift state before conversion is performed by **iconv()**. This value is only used on the *fromcode* parameter. The following values can be used:

- 0 The conversion descriptor is not returned to its initial shift state.

Note: The conversion descriptor can still be returned to its initial shift state by making a separate call to **iconv()**. *inbuf* must be equal to a null pointer or pointing to a null pointer.
- 1 The conversion descriptor is always returned to its initial shift state.

Substitution alternative. Whether the number of substitution characters encountered in the converted output data is returned by **iconv()**. This value is only used on the *fromcode* parameter. The following values can be used:

- 0 The number of substitution characters encountered is not returned.
- 1 The number of substitution characters encountered is returned. This number may be equal to or greater than the actual number of nonidentical conversions (substitutions) performed. The substitution character for a particular encoding scheme is defined in the Character Data Representation Architecture (CDRA).

Note: This value for the substitution alternative is valid only when using the enforced subset conversion criterion (057) for the conversion alternative field. Performance is decreased when using this alternative.

The word **IBMCCSID**. The 8-byte character field that contains the characters I, B, M, C, C, S, I, D. The characters must be in uppercase.

Return Value

If successful, **iconv_open()** returns a conversion descriptor of data type `iconv_t`. This conversion descriptor must be passed unchanged as an input parameter to the **iconv()** and **iconv_close()** functions.

If unsuccessful, **iconv_open()** returns -1 in the return value of the conversion descriptor and sets **errno** to indicate the error.

Error Conditions

The following errors can be returned in **errno**:

[EFAULT]

Bad address

The system detected an address that was not valid when attempting to use an argument from the parameter list. An escape message may also be signaled as a result.

[EINVAL]

Parameter not valid.

The conversion specified in the *fromcode* and *tocode* parameters is not supported.

When an **errno** value of `EINVAL` is returned, check the *fromcode* and *tocode* parameters for CCSIDs that are not valid or unsupported alternatives and options.

[ENOMEM]

Not enough space.

Insufficient storage space is available.

[EUNKNOWN]

Undetected error

An undetected error occurred. Contact your service organization. An escape message may also be signaled as a result.

The following escape messages can be signaled:

Message ID	Escape Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Usage Notes

This API is threadsafe if threads that share a conversion descriptor do not attempt to preserve the shift state.

Related Information

- “**iconv()**—Code Conversion API” on page 105—Code Conversion API
- “**iconv_close()**—Code Conversion Deallocation API” on page 116—Code Conversion Deallocation API

API introduced: V3R1

Top | “National Language Support APIs,” on page 1 | APIs by category

QtqIconvOpen()—Code Conversion Allocation API

```
iconv_t QtqIconvOpen (tocode, fromcode)
```

```
    QtqCode_T  *tocode;  
    QtqCode_T  *fromcode;
```

Service Program: QTQICONV

Default Public Authority: *USE

Threadsafe: Yes

The **QtqIconvOpen()** function performs the necessary initializations to convert character encodings from the source CCSID identified by the *fromcode* to the CCSID identified by the *tocode*. It then returns a conversion descriptor of data type `iconv_t`. For EBCDIC mixed-byte encodings, the conversion descriptor is set to the initial single-byte shift state.

Note: This API performs the same function as the **iconv_open()** API except that the input type of *fromcode* and *tocode* is of data type `QtqCode_T`.

The conversion descriptor remains valid in a job until that job closes with **iconv_close()** or the job ends. Keeping unneeded conversion descriptors active results in storage being allocated for these. To reduce this storage, you should call the **iconv_close()** API if you no longer need this conversion descriptor. The number of active descriptors is limited to a maximum of 104 000 active descriptors per process. This number may be reduced if the job has many active threads.

Parameters

tocode INPUT

A pointer to the structure containing the CCSID to convert to. For the layout of this structure, see “Format of `QtqCode_T`.”

fromcode

INPUT

A pointer to the structure containing the CCSID to convert from, the type of conversion to perform, the substitution alternative, and the shift-state alternative. For the layout of this structure, see “Format of `QtqCode_T`.”

Format of `QtqCode_T`

The `QtqCode_T` structure consists of the following.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	CCSID
4	4	BINARY(4)	Conversion alternative
8	8	BINARY(4)	Substitution alternative
12	C	BINARY(4)	Shift-state alternative
16	10	BINARY(4)	Input length option
20	14	BINARY(4)	Error option for mixed data

Offset		Type	Field
Dec	Hex		
24	18	CHAR(8)	Reserved

Field Descriptions

CCSID. The CCSID to convert from or to. Valid CCSID values are in the range 1 through 65533. The following special value is supported on the *fromcode* and *tocode* parameters:

0 The conversion descriptor is created with the CCSID of the current job such that calls to **iconv()** with the conversion descriptor use the CCSID of the job at the time the conversion descriptor was opened by **iconv_open()**. If the CCSID of the current job is 65535 (indicating no conversion), the default CCSID from the DFTCCSID job attribute is used.

Conversion alternative. The conversion alternative that is selected to convert graphic character data. This value is only used on the *fromcode* parameter; it is ignored on the *tocode* parameter. The following values can be used:

0 The IBM-defined default conversion method and the associated conversion tables. Most of the default tables follow the round-trip conversion criterion. For the default tables that do not follow the round-trip conversion criterion, see the Globalization topic in the iSeries Information center.

57 The enforced subset match (substitution) criterion. For the CCSID conversion pairs that support this criterion, refer to the Globalization topic in the iSeries Information center.

102 The best-fit conversion criterion for character mismatch.

Error option for mixed data. Whether **iconv()** returns an error when it converts a character string from a mixed-byte encoding scheme and the input buffer contains double-byte character (DBCS) characters.

Note: This is only valid on the *fromcode* parameter and only used if the target CCSID is SBCS.

Valid values are:

0 An error is not returned when converting from a mixed-byte encoding scheme and the input buffer contains DBCS characters. Instead, **iconv()** converts the DBCS characters to the CDRA-standardized control character substitute of the target CCSID.

1 An ECONVERT error is returned in **errno** when **iconv()** encounters a DBCS character in the input buffer when converting from a mixed-byte encoding scheme.

Input length option. Whether **iconv()** determines the number of bytes to the end of the input buffer being converted. This value is only used on the *fromcode* parameter. The possible values follow:

- 0 The conversion descriptor is created such that the `iconv()` function does not determine the number of bytes; therefore, a valid value for the `inbytesleft` parameter must be specified.
- 1 The conversion descriptor is created such that the `iconv()` function determines the number of bytes to the end of the input buffer to be converted; therefore, the input buffer must end with a NULL character. The resulting converted output buffer is ended with a NULL character by `iconv()`. The `inbytesleft` parameter must point to a value of zero.

Reserved. A reserved field that must be set to hexadecimal zeros.

Shift-state alternative. Whether the shift state should be restored to its initial shift state before conversion is performed by `iconv()`. This value is only used on the `fromcode` parameter; it is ignored on the `toctype` parameter. The following values can be used:

- 0 The conversion descriptor is not returned to its initial shift state.

Note: The conversion descriptor can still be returned to its initial shift state by making a separate call to `iconv()`. The `inbuf` must be equal to a null pointer or pointing to a null pointer.
- 1 The conversion descriptor is always returned to its initial shift state.

Substitution alternative. Whether the number of substitution characters encountered in the converted output data is returned by `iconv()`. This value is only used on the `fromcode` parameter; it is ignored on the `toctype` parameter. The following values can be used:

- 0 The number of substitution characters encountered is not returned.
- 1 The number of substitution characters encountered is returned. This number may be equal to or greater than the actual number of nonidentical conversions (substitutions) performed. The substitution character for a particular encoding scheme is defined in the Character Data Representation Architecture (CDRA).

Note: This value for the substitution alternative is valid only when using the enforced subset conversion criterion for the conversion alternative. Performance is decreased when using this alternative.

Return Value

If successful, `QtqIconvOpen()` returns a conversion descriptor of data type `iconv_t`. This conversion descriptor must be passed unchanged as an input parameter to the `iconv()` and `iconv_close()` functions.

If unsuccessful, `QtqIconvOpen()` returns -1 and in the return value of the conversion descriptor and sets `errno` to indicate the error.

Error Conditions

The following errors can be returned in `errno`:

[EFAULT]

Bad address

The system detected an address that was not valid when attempting to use an argument from the parameter list. An escape message may also be signaled as a result.

[EINVAL]

Parameter not valid.

The conversion specified in the *fromcode* and *toctype* parameters is not supported.

When an errno value of EINVAL is returned, check the *fromcode* and *toctype* parameters for CCSIDs that are not valid or unsupported alternatives and options.

[ENOMEM]

Not enough space.

Insufficient storage space is available.

[EUNKNOWN]

Undetected error

An undetected error occurred. Contact your service organization. An escape message may also be signaled as a result.

The following escape messages can be signaled:

Message ID	Escape Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Related Information

- “[iconv\(\)](#)—Code Conversion API” on page 105—Code Conversion API
- “[iconv_close\(\)](#)—Code Conversion Deallocation API”—Code Conversion Deallocation API

API introduced: V3R1

[top](#) | [“National Language Support APIs,” on page 1](#) | [APIs by category](#)

iconv_close()—Code Conversion Deallocation API

Syntax

```
#include <iconv.h>
```

```
int iconv_close (cd)
```

```
    iconv_t    cd;
```

Service Program: QTQICONV

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 117.

The `iconv_close()` function closes the conversion descriptor `cd` that was initialized by the `iconv_open()` or `QtqIconvOpen()` function.

Parameters

`cd` INPUT

The conversion descriptor returned by a previous successful call to `iconv_open()` or `QtqIconvOpen()`.

Return Value

If an error occurs, `iconv_close()` returns a value of -1 and `errno` is set to indicate the error. If `iconv_close()` completes successfully, a value of zero is returned.

Error Conditions

The following errors can be returned in `errno`:

[EBADF]

Descriptor not valid.

The conversion descriptor (`cd`) parameter is not valid.

[EUNKNOWN]

Undetected error

An undetected error occurred. Contact your service organization. An escape message may also be signaled as a result.

The following escape messages can be signaled:

Message ID	Escape Message Text
CPF3C90 E	Literal value cannot be changed.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Usage Notes

This API is threadsafe if threads that share a conversion descriptor do not attempt to preserve the shift state.

Related Information

- “`iconv()`—Code Conversion API” on page 105—Code Conversion API
- “`iconv_open()`—Code Conversion Allocation API” on page 109—Code Conversion Allocation API
- “`QtqIconvOpen()`—Code Conversion Allocation API” on page 113—Code Conversion Allocation API

API introduced: V3R1

Top | “National Language Support APIs,” on page 1 | APIs by category

Convert Data (QDCXLATE) API

Required Parameter Group:	
1	Length of data being converted
Input	Packed(5,0)
2	Conversion data
I/O	Char(*)
3	SBCS conversion table name
Input	Char(10)
Optional Parameter Group:	
4	SBCS conversion table library name
Input	Char(10)
5	Output data
Output	Char(*)
6	Length of output buffer
Input	Packed(5,0)
7	Length of converted data
Output	Packed(5,0)
8	DBCS language
Input	Char(10)
9	Shift-out and shift-in characters
Input	Char(1)
10	Type of conversion
Input	Char(10)
Default Public Authority: *USE	
Threadsafe: Yes	

The Convert Data (QDCXLATE) API converts data through the use of a table object. (If you need to convert the case of your data, it is recommended that you use the “Convert Case (QLGCNVCS, QlgConvertCase) API” on page 5 (QLGCNVCS, QlgConvertCase API.) You also can use the equivalent API QTDXLATE to achieve the same function. The call interface to QTDXLATE is identical to Convert Data (QDCXLATE).

You can create the conversion table that QDCXLATE uses for the conversion, or you can use an IBM-supplied table. The IBM-supplied tables can be found in the QUSRSYS library. For a list of both the conversion tables and the casing tables, see the Globalization topic in the iSeries Information Center. These tables are not the same as those used by the intersystem communications function (ICF) for conversion support. For information on this, see Sockets Programming.

You can create your own conversion tables using the Create Table (CRTTBL) command.

When the QDCXLATE API is called with parameters 1, 2, 3, and 4, it converts single-byte data. When all parameters are specified, DBCS conversion is taking place.

The QDCXLATE API can distinguish double-byte from single-byte characters when converting from EBCDIC to ASCII and from ASCII to EBCDIC if the proper parameters have been supplied. The QDCXLATE API converts data byte for byte and returns the converted data to your program.

When only single-byte data is converted, the input (unconverted) data is replaced with the converted data. When double-byte data is converted, the converted data is placed in the output data parameter.

The QDCXLATE API is thread safe only when converting single-byte data or T.61 data.

Authorities and Locks

Table Authority

*USE

Table Library Authority

*USE

Required Parameter Group

Length of data being converted

INPUT; PACKED(5,0)

The length of the data being converted. This value cannot exceed 32 767.

Conversion data

I/O; CHAR(*)

The data to be converted. This buffer also contains the output data after conversion when the API is called with only the required parameter group.

SBCS conversion table name

INPUT; CHAR(10)

The name of the single-byte character set (SBCS) conversion table to be used. The table may be a system-supplied or user-supplied conversion table. The table name must be left-justified.

Note: This parameter is ignored when the DBCS language parameter is set to *BG5, *KSC, *SCGS, *J90X5026, *J90X5035, or *SCGBK.

Optional Parameter Group

SBCS conversion table library name

INPUT; CHAR(10)

The name of the library that contains the SBCS conversion table. The library name must be left-justified. If this parameter is not specified, the library list is used to locate the conversion table. This parameter is ignored when the DBCS language parameter is set to *BG5, *KSC, *SCGS, *J90X5026, *J90X5035 or *SCGBK.

Output data

OUTPUT; CHAR(*)

The output buffer that contains the double-byte character set (DBCS) data that was converted.

Because of the insertion of shift-out and shift-in characters, it is possible that the converted data is longer than the source data. If this is the case, it is not possible to do the conversion in place, as is done when you use only a required-parameter-group call. The converted data is then placed in the area pointed to by this parameter.

Length of output buffer

INPUT; PACKED(5,0)

The size of the output data buffer. The maximum length should match the actual size of the output data parameter. If the converted output is longer than the length of output buffer parameter, an exception is signaled.

Length of converted data

OUTPUT; PACKED(5,0)

The actual length of the converted output in the output data parameter.

DBCS language

INPUT; CHAR(10)

The DBCS language that is being converted. All values must be padded on the right with blanks. The possible values follow:

*JPN	IBM Japanese graphic character set
*KOR	IBM Korean graphic character set
*CHS	IBM Simplified Chinese graphic character set
*CHT	IBM Traditional Chinese graphic character set
*BG5	Taiwan industry standard graphic character set (BIG-5)
*KSC	Korean industry standard graphic character set (KS)
*SCGS	The People's Republic of China National standard graphic character set (GB)
*J90X5026	The Japanese JIS X 0208 1990 standard mapped using CCSID 5026.
*J90X5035	The Japanese JIS X 0208 1990 standard mapped using CCSID 5035.
*SCGBK	The People's Republic of China National standard graphic character set extended (GBK)

Shift-out and shift-in characters

INPUT; CHAR(1)

Whether shift-out and shift-in characters should be inserted during the conversion. This parameter is ignored when the DBCS language parameter is set to *BG5, *KSC, *SCGS, *J90X5026, *J90X5035 or *SCGBK. The possible values follow:

Y	Insert shift-out and shift-in characters
N	Do not insert shift-out and shift-in characters

Type of conversion

INPUT; CHAR(10)

The type of DBCS conversion being done. The possible values follow:

*AE	Convert ASCII to EBCDIC
*EA	Convert EBCDIC to ASCII

Note: You are responsible for specifying the correct SBCS table name for the type of conversion being done by this DBCS request except when the DBCS language parameter is set to *BG5, *KSC, *SCGS, *J90X5026, *J90X5035 or *SCGBK.

Error Messages

Message ID Error Message Text


CPF2143 E	Cannot allocate object &1 in &2 type *&3.
CPF2144 E	Not authorized to &1 in &2 type *&3.
CPF24B4 E	Severe error while addressing parameter list.
CPF2619 E	Table &1 not found.
CPF264D E	Double-byte character set language not valid.

Message ID	Error Message Text
CPF264E E	Shift-in and shift-out value of double-byte character set not valid.
CPF264F E	Translation type of double-byte character set not valid.
CPF2647 E	Buffer length not valid.
CPF265E E	Number of parameters specified not valid.
CPF265F E	Translation length exceeded maximum length.
CPF2651 E	Table &1 not found.
CPF2669 E	Double-byte character set source not valid.
CPF269A E	Library parameter is not set to "QSYS" on call.
CPF269B E	T.61 conversion table not found.
CPF269C E	Error in input data.
CPF269D E	A nonspacing underline character was found in the input data.
CPF3C90 E	Literal value cannot be changed.
CPF6309 E	Not authorized to library &1.
CPF9802 E	Not authorized to object &2 in &3.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API existed prior to V1R3

[Top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

Character Data Representation Architecture (CDRA) APIs

Note: You should have access to the *Character Data Representation Architecture Reference* book, SC09-1390-01, before using this set of APIs. This book is not available online, but can be ordered from the IBM Publications Center .

The Character Data Representation Architecture (CDRA) APIs are:

- "Convert a Graphic Character String (CDRCVRT, QTQCVRT) API" on page 123 (CDRCVRT, QTQCVRT) converts a graphic character data string of the identified string type (ST1) represented in a specified from CCSID (CCSID1) to a graphic character data string of the required string type (ST2) that is represented in another specified to CCSID (CCSID2).
- "Get CCSID for Normalization (CDRGCCN, QTQGCCN) API" on page 128 (CDRGCCN, QTQGCCN) assists in determining the CCSID for normalization given two CCSIDs. The returned CCSID may equal one or both the input CCSIDs. When certain operations, such as concatenation or comparison are performed on graphic character strings, the two strings are both in the same CCSID, or they are normalized first to a single CCSID before concatenation.
- "Get Control Function Definition (CDRGCTL, QTQGCTL) API" on page 130 (CDRGCTL, QTQGCTL) gets a requested control function definition associated with a given CCSID.
- "Get Encoding Scheme, Character Set, and Code Page Elements (CDRGESP, QTQGESP) API" on page 133 (CDRGESP, QTQGESP) returns the value of the encoding scheme associated with CCSID1 in ES, and the values of CS and CP elements in CSCPL.
- "Get Related Default CCSID (CDRGRDC, QTQGRDC) API" on page 135 (CDRGRDC, QTQGRDC) allows the caller to get a nearest equivalent or best-fit related CCSID.
- "Get Short Form (CCSID) from Specified ES and (CS, CP) Pair (CDRSCSP, QTQSCSP) API" on page 137 (CDRSCSP, QTQSCSP) gets the CCSID associated with the specified (CS, CP) pairs and ES. It aids in coexistence and migration for products that have to deal with the short form (CCSID) of identification on one side and the intermediate form (CGCSGID) on the other.
- "Get Short Form with Maximal CS for Specified ES and CP (CDRSMXC, QTQSMXC) API" on page 140 (CDRSMXC, QTQSMXC) gets the CCSID with the maximal character set (CS) for a given code page (CP) value.

- “Get Short Form with Maximal CS for Specified ES and CP (QTQSMXC2) API” on page 142 (QTQSMXC2) gets the CCSID with the maximal character set (CS) for a given code page (CP) value. The QTQSMXC2 API acts as an extension of the functions of the CDRSMXC API in that you can specify more code pages.
- “Resolve Client Server CCSID (QTQRCSC) API” on page 145 (QTQRCSC) returns the CCSID that will enable the proper transfer of data between the server and the client.

The CDRA APIs listed above are in library QSYS2. You can use equivalent QTQ APIs, stored in library QSYS, to achieve the same function. The API name and library comparison is as follows:

CDRA API in Library QSYS2	QTQ API in Library QSYS
CDRCVRT	QTQCVRT
CDRGCCN	QTQGCCN
CDRGCTL	QTQGCTL
CDRGESP	QTQGESP
CDRGRDC	QTQGRDC
CDRSCSP	QTQSCSP
CDRSMXC	QTQSMXC

Top | “National Language Support APIs,” on page 1 | APIs by category

Convert a Graphic Character String (CDRCVRT, QTQCVRT) API

Required Parameter Group:	
1	CCSID1
Input	Int(4)
2	ST1
Input	Int(4)
3	S1
Input	Char(*)
4	L1
Input	Int(4)
5	CCSID2
Input	Int(4)
6	ST2
Input	Int(4)
7	GCCASN
Input	Int(4)
8	L2
Input	Int(4)
9	S2
Output	Char(*)
10	L3
Output	Int(4)
11	L4
Output	Int(4)
12	FB
Output	Feedback
	Default Public Authority: *USE
Threadsafe: Yes	

The Convert a Graphic Character String (CDRCVRT, QTQCVRT) API converts a graphic character data string of the identified string type (ST1) represented in a specified *from* CCSID (CCSID1) to a graphic character data string of the required string type (ST2) that is represented in another specified *to* CCSID (CCSID2).

The function assumes that the entire string to be converted is known and is passed to the function. Also, the caller has provided sufficient space for the returned converted string. In case of an overflow situation, an orderly truncation would result.

Required Parameter Group

CCSID1

INPUT; INT(4)

This variable contains the CCSID value for the input graphic character data string being converted. The valid range is 1 to 65,533.

ST1 INPUT; INT(4)

This variable contains the type of input string. The following types are defined:

Type	Explanation
0	A graphic character string, as semantically defined by CCSID1.
1	A graphic character string, as semantically defined by CCSID1, and null-terminated.

S1 INPUT; CHAR(*)

This variable contains the graphic character data to be converted.

L1 INPUT; INT(4)

This variable contains the length (in number of bytes) of:

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

The valid range is from 1 to 32,767.

CCSID2

INPUT; INT(4)

This variable contains the CCSID value for the converted graphic character data string. The valid range is 1 to 65,533.

ST2 INPUT; INT(4)

This variable contains the type of output string. The following types are defined:

Type	Explanation
0	A graphic character string, as semantically defined by CCSID2.
1	A graphic character string, as semantically defined by CCSID2, and null-terminated.
2	A graphic character string, as semantically defined by CCSID2, and SPACE-padded.

GCCASN

INPUT; INT(4)

This variable contains a number that identifies which conversion alternative is to be selected to convert graphic character data from (CCSID1, ST1) to (CCSID2, ST2). Valid values are:


Value	Nature of the Conversion Alternative Selected
0	Used to select the designated "installation default" conversion method and tables.
1	Used to select the CDRA-defined default method and associated conversion tables. The difference management criterion used in the creation of the selected tables is based on country or region requirements to serve the majority of applications using the selected CCSID pairs.

Value
57

Nature of the Conversion Alternative Selected
Enforced subset match.

Notes:

1. The values 0 and 1 are treated the same.
2. *Character Data Representation, SC09-1391*, contains the CDRA-defined default conversion tables.

This book is not available online, but can be ordered from the IBM Publications Center .

L2 INPUT; INT(4)

This variable contains the byte-length of the allocated area to contain the converted graphic character data. The valid range is 1 to 32,767.

S2 OUTPUT; CHAR(*)

The converted graphic character data. The area's allocated length is given in L2.

Under certain error conditions the output may contain the results of converting only a part of the input string.

L3 OUTPUT; INT(4)

This variable contains the byte-length of the converted string returned in S2.

The byte-length includes any null termination or padding characters necessary to retain the semantics of CCSID2 and ST2.

L4 OUTPUT; INT(4)

The implementation of this API does not support the function of this parameter. This value is always returned as 0.

FB OUTPUT; FEEDBACK

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	Requested conversion is not supported
0001	0005	The requested conversion algorithm specified by GCCASN does not support the specified (CCSID1, ST1) to (CCSID2, ST2) combination.
0001	0006	GCCASN value is 0; but an "installation default" was not found in the GCCST, for the pair (CCSID1, ST1) to (CCSID2, ST2).
0002	0001	CCSID1 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0002	0002	CCSID2 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 is 65,535.
0003	0002	CCSID2 is 65,535.

Status	Reason	Meaning
0004	0001	The length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to 0.
0004	0002	The encoding scheme of CCSID1 is X'1301' (mixed Host; single-byte/double-byte encoding). The length value in L2 allocated for area S2 was too small for the output data. A properly truncated and terminated converted string that fits within the allocated maximum is returned in the area starting at S2 with its byte-length in L3. The value in L4 is set to 0.
0005	0001	A pure double-byte CCSID1 was specified and either: <ul style="list-style-type: none"> • ST1=0 and L1 is odd, or • ST1=1 and an orphan byte was found.
0005	0004	ES of CCSID1 is X'1301', and a malformed string—an odd number of bytes between SO, SI bracket—was encountered.
0005	0005	A null-terminated input string was specified using ST1=1; however, there was no null-termination character in S1 within the length L1 specified.
0005	0006	A null-terminated output string was specified using ST2=1; however, the output string contains one or more characters matching the null-termination character, resulting from using the selected conversion tables and methods.
0005	0007	A SPACE-padded output string was specified using ST2=2; however, the definition for SPACE character could not be obtained. The CCSID resource definition did not have an entry for SPACE character definition, or the CCSID resource definition could not be found.
0005	0008	A pure double-byte CCSID2 with ST2=1 was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns only an even number of bytes (maximum L2-1 bytes), including the null-termination character in S2.
0005	0009	A pure double-byte CCSID2 with ST2=2 (SPACE-padded string) was specified, and an odd value was specified for length L2 of the output buffer. The convert function returns L2-1 bytes, including the SPACE-padding characters, in S2.
0005	000C	ES of CCSID1 is X'1301', and a trailing SI bracket is missing.
0005	000D	ES of CCSID1 is X'1301', and a trailing SI code point was encountered without first encountering its corresponding leading SO code point (the number of intervening code points may have been odd or even; the code points would have been treated as single-byte code points because the leading SO was missing).
0006	0001	The selection table (GCCST) was not found.
0006	0002	A CDRA resource is currently unavailable.
0006	0003	The conversion method identified in the GCCST for the specified selection is currently unavailable.
0006	0004	A conversion table identified in the GCCST for the specified selection is not found.
0007	0001	The system GCCST resource accessed by the function is found to be invalid in structure.
0007	0002	The system GCCT resource accessed by the function is found to be invalid in structure.

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

Usage Notes

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

API introduced: V2R2

Top | "National Language Support APIs," on page 1 | APIs by category

Get CCSID for Normalization (CDRGCCN, QTQGCCN) API

Required Parameter Group:	
1	CCSID1
Input	Int(4)
2	CCSID2
Input	Int(4)
3	CCSIDN
Output	Int(4)
4	HINTV
Output	Int(4)
5	FB
Output	Feedback Default Public Authority: *USE
Threadsafe: No	

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

Required Parameter Group

CCSID1

INPUT; INT(4)

This variable contains the first CCSID value referenced. Possible values are a positive number in the range 1 to 65,533.

CCSID2

INPUT; INT(4)

This variable contains the second CCSID value referenced. Possible values are a positive number in the range 1 to 65,533.

CCSIDN

OUTPUT; INT(4)

This variable contains the returned CCSID value for normalization. Possible values are a positive number in the range 1 to 65,533.

HINTV

OUTPUT; INT(4)

The function returns in this variable a number that conveys information to assist the calling function in its subsequent processing. The following values and meanings are defined:

HINTV

0

Meaning

No hints

HINTV

1

Meaning

CCSID1 and CCSID2 have both the same value for their CP element. The returned CCSIDN has a character set which is a superset of or equals the larger of the character sets of CCSID1 and CCSID2.

2

CCSIDN has the same CP element as CCSID2. The character set of CCSIDN is a superset of or equals the character set of CCSID2. Only the string with CCSID1 needs to be converted to CCSIDN.

3

CCSIDN has the same CP element as CCSID1. The character set of CCSIDN is a superset of or equals the character set CCSID1. Only the string with CCSID2 needs to be converted to CCSIDN.

FB OUTPUT; FEEDBACK

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	There is no entry in the resource—Normalization Support CCSID Table (NSCT)—for the pair CCSID1, CCSID2.
0002	0001	CCSID1 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0002	0002	CCSID2 value is 0, which is reserved to indicate defaulting to a higher level in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 is 65,535.
0003	0002	CCSID2 is 65,535.
0006	0001	The NSCT resource was not found.
0006	0002	The NSCT resource is currently unavailable.
0007	0001	The system NSCT resource accessed by the function is found to be invalid in structure.
0008	0001	CCSID1 value is not in the range 0 to 65,535.
0008	0002	CCSID2 value is not in the range 0 to 65,535.

Usage Notes

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

API introduced: V2R2

Get Control Function Definition (CDRGCTL, QTQGCTL) API

Required Parameter Group:	
1	CCSID1
Input	Int(4)
2	SEL
Input	Int(4)
3	N1
Input	Int(4)
4	N2
I/O	Int(4)
5	CTLFDF
Output	Array of Int(4)
6	FB
Output	Feedback Default Public Authority: *USE
Threadsafe: No	

The Get Control Function Definition (CDRGCTL, QTQGCTL) API gets a requested control function definition associated with a given CCSID. The following control function definitions are defined in the CCSID resource repository model:

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

Additionally, the following other definitions are included in this repository:

- Space
- String Type

Each control function definition is defined as a triplet consisting of:

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

Each control function may be defined for each of the possible code extension switching states associated with the CCSID. There is, at most, one code point definition for a control function within a switching state.

If the string type definition is selected it's value will be returned in the first value of the triplet. The remaining two values of the triplet will be returned as 0.

A selection parameter (SEL) is used to identify which control function definition is to be returned by the function.

Required Parameter Group

CCSID1

INPUT; INT(4)

This variable contains the CCSID value referenced; a positive number in the range 1 to 65,279.

SEL INPUT; INT(4)

This variable contains the selection specification, a non-negative number in the range 0 to 255. If the selected control function element is available in the resource definition for CCSID1, the triplets are returned in the area starting at CTLFDF. The following values are currently defined for SEL:

SEL	Selected Control Function
0	Space
1	Substitute
2	New Line
3	Line Feed
4	Carriage Return
5	End of File
100	String Type

N1 INPUT; INT(4)

This variable contains the size of the allocated area starting at CTLFDF to contain the returned data. N1 is specified as a number of elements. Each triplet is counted as 3 elements. It is a positive number whose minimum value is 3.

N2 I/O; INT(4)

This variable contains the number of values returned in CTLFDF. The first invocation of this function must have N2 initialized to zero. It is a non-negative integer and is a multiple of 3 (corresponding to each triplet in CTLFDF). If no definition is found in the CCSID resource for the requested element, a value of 0 is returned in N2.

CTLFDF

OUTPUT; INT(4)

The start of the area to contain the selected definition elements. Each element is a triplet of 3 binary(4) values. For each triplet, the first value is the code point, the second is the code point width, and the third value contains the switching state number. There is one triplet returned for each switching state for CCSID1. An undefined element is indicated by a zero state number in the corresponding CTLFDF entry.

If the string type definition is selected it's value will be returned in the first value of the triplet. The remaining two values of the triplet will be returned as 0. The following table provides a definition of each of the string types:

String Type	Text Type	Numeric Shaping	Orientation	Text Shaping	Symmetrical Swapping
4	Visual	Passthrough	LTR	Shaped	Off
5	Implicit	Arabic	LTR	Unshaped	On
6	Implicit	Arabic	RTL	Unshaped	On
7(*)	Visual	Passthrough	Contextual *	Unshaped-Ligatures	Off
8	Visual	Passthrough	RTL	Shaped	Off

String Type	Text Type	Numeric Shaping	Orientation	Text Shaping	Symmetrical Swapping
9	Visual	Passthrough	RTL	Shaped	On
10	Implicit		Contextual LTR		On
11	Implicit		Contextual RTL		On

(*) Field orientation is left to right (LTR), when first alphabetic character is a Latin one, and right to left (RTL) when it is a Bidi (RTL) character; characters are unshaped, but LamAlef ligatures are kept, and not broken into constituents.

FB OUTPUT; Feedback

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	CCSID1 value is not in the CCSID resource repository.
0001	0004	One or more of the requested control function definitions are undefined (as indicated by a zero value for its corresponding state number in CTLFDF).
0001	000A	The requested control function definition element in the CCSID resource for CCSID1 was not found.
0002	0001	CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 has one of the special-purpose CCSID values in the range 65,280 to 65,535.
0004	0001	The allocated length (value of N1) for the area to contain returned values was insufficient to contain all the output data that is to be returned.
0005	0002	N2 is greater than N1. However, the start of the next block of data to be returned is outside the valid range 1 to N2max.
0005	0003	The value specified in the SEL parameter is not supported.
0005	000A	N2 is less than or equal to N1, but is not 0.
0006	0001	The CCSID resource repository was not found.
0006	0002	The CCSID resource repository is currently unavailable.
0007	0001	The system CCSID resource repository accessed by the function was found to be not valid in the structure.
0008	0001	CCSID1 value is not in the range 0 to 65,535.
0008	0002	N1 is greater than the maximum permitted in this implementation. (The maximum permitted is 99.)
0008	000A	N1 is less than 3.
0008	000B	SEL value is not in the range 0 to 255.

Exceptions issued: CPF9872 - Program &1 in library &2 ended.

Usage Notes

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

For example, for CCSID 00037, the return value, if the space was asked for, would be X'00000040'. This is interpreted as a 1 byte value of X'40'.

API introduced: V3R6

Top | “National Language Support APIs,” on page 1 | APIs by category

Get Encoding Scheme, Character Set, and Code Page Elements (CDRGESP, QTQGESP) API

Required Parameter Group:	
1	CCSID1
Input	Int(4)
2	N1
Input	Int(4)
3	N2
I/O	Int(4)
4	ES
Output	Int(4)
5	CSCPL
Output	Array of Int(4)
6	FB
Output	Feedback
	Default Public Authority: *USE
Threadsafe: Yes	

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

Required Parameter Group

CCSID1

INPUT; INT(4)

This variable contains the CCSID value referenced; a positive number in the range 1 to 65,533.

N1 INPUT; INT(4)

This variable contains the size of the allocated area starting at CSCPL to contain the returned data. N1 is specified as a number of elements, each CS/CP pair is counted as 2 elements. It is an even number greater than or equal to 2.

N2 I/O; INT(4)

This variable contains the number of values (each pair of CS and CP is counted as two values) associated with CCSID1 and returned in CSCPL providing that sufficient space (N1) was allocated. The first invocation of this function must have N2 initialized to zero.

ES OUTPUT; INT(4)

This variable contains the ES value associated with CCSID1. It is a positive number in the range 4352 to 65,534.

CSCPL

OUTPUT; Array of INT(4)

Start of the area to contain the CS/CP elements returned. Each element is a pair of Binary(4) values. For each pair, the first value is the character set and the second value is the code page.

FB OUTPUT; FEEDBACK

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	CCSID1 value is not in the CCSID resource repository.
0002	0001	CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. The caller must resolve the default before calling this function.
0003	0001	CCSID1 is 65,535.
0004	0001	The allocated length (value of N1) for the area to contain returned values was insufficient to contain all the output data that is to be returned.
0005	0002	N2 is greater than N1. However, the start of the next block of data to be returned is outside the valid range 1 to N2max.
0005	000A	N2 is less than or equal to N1, but is not 0.
0006	0001	The CCSID resource repository was not found.
0006	0002	The CCSID resource repository is currently unavailable.
0007	0001	The system CCSID resource repository accessed by the function was found to be incorrect in the structure.
0007	0004	There was no ES element definition in the CCSID resource for CCSID1.
0007	0006	There was no definition for CS, CP elements in the CCSID resource for CCSID1.
0008	0001	CCSID1 value is not in the range 0 to 65,535.

Status	Reason	Meaning
0008	0002	N1 value is greater than the maximum allowed in this implementation, or N1 is odd.
0008	0003	N1 is less than 2.

Usage Notes

1. The maximum number of (CS, CP) values depends on the ES. Most CCSIDs have only one pair of (CS, CP) values. A calling function can set N1 to 32, to accommodate up to 16 pairs of (CS, CP) values without overflow.

API introduced: V2R2

Top | "National Language Support APIs," on page 1 | APIs by category

Get Related Default CCSID (CDRGRDC, QTQGRDC) API

Required Parameter Group:	
1	CCSID1
Input	Int(4)
2	ESIN
Input	Int(4)
3	SEL
Input	Int(4)
4	CCSIDR
Output	Int(4)
5	FB
Output	Feedback
	Default Public Authority: *USE
Threadsafe: No	

A given CCSID may not be directly usable in many situations. The Get Related Default CCSID (CDRGRDC, QTQGRDC) API allows the caller to get a nearest equivalent or best-fit related CCSID. The caller supplies an ES value as an additional key to select the appropriate related CCSID.

Required Parameter Group

CCSID1

INPUT; INT(4)

This variable contains the CCSID value referenced, a positive number in the range 1 to 65,533.

ESIN INPUT; INT(4)

This variable contains the ES value referenced; a positive number in the range 4352 to 65,534.

SEL INPUT; INT(4)

This variable is reserved to identify any specific selection criteria as additional input. For example, to select among two equally valid related defaults.

SEL **Meaning**
 0 Installation default

CCSIDR
 OUTPUT; INT(4)

This variable contains the returned CCSID value; a positive number in the range 1 to 65,533. If no related default is found, CCSIDR is set to CCSID1.

FB OUTPUT; FEEDBACK

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	No entry was found in the Related Default CCSID Table (RDCT) resource for the CCSID1, ESIN, and SEL combination specified. CCSID1 value is copied and returned in CCSIDR.
0002	0001	CCSID1 value is 0, which is reserved for indicating a default in a hierarchy. It must be resolved before this function is called.
0003	0001	CCSID1 is 65,535.
0005	0001	The value of SEL specified is not supported.
0006	0001	The RDCT resource was not found.
0006	0002	The RDCT resource is currently unavailable.
0007	0001	The system RDCT resource accessed by the function is found to be incorrect in the structure.
0008	0001	CCSID1 value is not in the range 0 to 65,635.
0008	0002	ESIN value is not in the range 4352 to 65,534. The CCSID1 value is copied and returned in CCSIDR.
0008	000B	SEL value is not in the range 0 to 255.

Usage Notes

1. CCSID is not locally supported:

A given CCSID is not one of the supported CCSIDs in that environment. For example, an iSeries server supporting only EBCDIC CCSIDs may be serving a PC user, where all the data generated is in one of the PC CCSIDs. A DB2 UDB for iSeries database server receiving the SQL statement CREATE TABLE from the PC user may not be able to create a table in any other CCSID than an EBCDIC CCSID supported in that installation. Before the table creation is completed, a CCSID value is needed to identify the table's CCSID. Any data from the PC that is placed in this table is converted to this CCSID. However, a selection from the locally supported CCSIDs of a single CCSID that can preserve the maximum number of PC graphic characters is needed. If the user does not supply this CCSID, the system defaults to a CCSID.

This function gets a CCSID that is predetermined to be the best fit. An ES value is supplied as a key to identify the local environment's needs or characteristic.

2. CCSID to match a specific data type is needed:

In a situation where a given CCSID is incompatible with the data type (for example, SBCS CCSID and graphic data type), and a group of CCSIDs (one for each of SBCS, DBCS, and mixed SB/DB encoding schemes) are used, it is necessary to pick the correct CCSID that matches the data type.

Most CCSIDs registered to date have only one CCSID per data type and ES match. Such CCSIDs share one or more CS and CP values among them, and differ only on the ES ID values. However, since some CS and CP values can be shared between different CCSIDs (with the same ES), more than one CCSID can qualify to be used.

For example, the CS, CP (00370, 00300) of DBCS Japanese Host CCSID 00300 is used in CCSID 05026 (with SBCS Katakana Extended CS 01172, CP 00290) and in CCSID 05035 (with SBCS Latin Extended CS 01172 and CP 01027). If only DBCS CCSID 00300 was specified, both the CCSIDs 05026 and 05035 qualify to be a related mixed CCSID. Also, both the SBCS CCSIDs 00290 and 01027 qualify to be the related SBCS CCSID.

However, a single default value selected from the multiple possible CCSIDs is predetermined and made available as a *related default* CCSID. This function gets this predetermined default for the caller-supplied CCSID value. The defaults are arranged with ES as the key to properly match the data type needed by the caller.

API introduced: V2R2

Top | "National Language Support APIs," on page 1 | APIs by category

Get Short Form (CCSID) from Specified ES and (CS, CP) Pair (CDRSCSP, QTQSCSP) API

Required Parameter Group:

1 CSCPL

Input Int(4)

2 N1

Input Int(4)

3 ESIN

Input Int(4)

4 CCSIDR

Output Int(4)

5 ESR

Output Int(4)

6 FB

Output Feedback

Default Public Authority: *USE

Threadsafe: No

The Get Short Form (CCSID) from Specified ES and (CS, CP) Pair (CDRSCSP, QTQSCSP) API gets the CCSID associated with the specified (CS, CP) pairs and ES. It aids in coexistence and migration for products that have to deal with the short form (CCSID) of identification on one side and the intermediate form (CGCSGID) on the other. The ES is further required to distinguish between usage of the same CS, CP with two different encoding schemes, (such as CS 00697 and CP 00850, with ES values of X'2100' and X'3100'), and when more than one CGCSGID is associated with a CCSID (such as with ES X'1301', X'2300', and X'3300')

Required Parameter Group

CSCPL

INPUT; INT(4)

This variable is an array whose format is CS1, CP1, CS2, CP2,...CSn, CPn. Each CS is a positive number in the range 1 to 65,535. Each CP is a positive number in the range 1 to 65,534. Each CP is placed in a single CSCPL array element.

N1 INPUT; INT(4)

This variable contains the number of elements in CSCPL, an even positive number in the range 2 to 32 (can accommodate up to 16 CS and CP pairs.)

ESIN INPUT; INT(4)

This variable contains the ES value referenced, zero or a positive number in the range 4352 to 65,534. The implementation of this API supports only a value of 0 for this parameter.

ESIN

0

Meaning

The caller does not know the ESID value. The CCSID returned is the first occurring in the CCSID resource repository whose CS and CP values match those specified in CSCPL.

CCSIDR

OUTPUT; INT(4)

This variable contains the returned CCSID value, a positive number in the range 1 to 65,534. A value of 65,535 is returned when the function could not find the requested CCSID.

ESR OUTPUT; INT(4)

This variable contains the ES value of the returned CCSID, a zero, or a positive number in the range 4352 to 65,534. The implementation of this API does not support the function of this parameter.

FB OUTPUT; FEEDBACK

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

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0001	No entry was found in the CCSID resource repository for the specified ESIN and CS, CP pairs.

Status	Reason	Meaning
0001	0002	A single CCSID value with the specified CSCPL was found, but not with the specified ESIN. The values of CCSID found and its associated ES are returned.
0002	0001	A CP value in CSCPL is zero.
0002	0002	A CS value in CSCPL is zero.
0003	0001	A CP value in CSCPL is 65,535.
0005	0001	N1 is odd.
0006	0001	The CCSID resource repository was not found.
0006	0002	The CCSID resource repository is currently unavailable.
0007	0001	The system CCSID resource repository accessed by the function was found to be invalid in structure.
0008	0001	A CS or CP value in CSCPL array is not in the range 0 to 65,535.
0008	0002	N1 is greater than the maximum permitted in this implementation.
0008	0003	N1 is less than 2.
0008	0004	ESIN value is nonzero and not in the range 4352 to 65,534.

Usage Notes

1. Often it is required to find the CCSID when the ES and (CS,CP) values are known. CS, CP (also known as CGCSGID or GCID) is used in many existing IBM architectures, data streams, and supporting products. This function aids in coexistence and migration for products that have to deal with the short form (CCSID) of identification on one side and the intermediate form (CGCSGID) on the other. Intermediate forms are by themselves incomplete, that is, without an encoding scheme a single CCSID cannot be identified in all cases. The function can return only a default value as defined in the installation's resources when the ESIN information is not known.
2. For the CCSIDs defined to date, the maximum number of CS, CP pairs is 2 (up to four values can be specified in CSCPL). There may be additional CCSIDs which meet the specified criteria.

API introduced: V2R2

[top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

Get Short Form with Maximal CS for Specified ES and CP (CDRSMXC, QTQSMXC) API

Required Parameter Group:	
1	CPIN
Input	Int(4)
2	ESIN
Input	Int(4)
3	CCSIDR
Output	Int(4)
4	ESR
Output	Int(4)
5	FB
Output	Feedback
	Default Public Authority: *USE
Threadsafe: Yes	

The Get Short Form with Maximal CS for Specified ES and CP (CDRSMXC, QTQSMXC) API gets the CCSID with the maximal character set (CS) for a given code page (CP) value. The encoding scheme (ES) parameter can be specified to distinguish between usage of the same CP with two different encoding schemes, such as PC Display compared to PC Data.

The function works only with those CCSIDs that have only one (CS, CP) pair associated with them. That is, pure single-byte or pure double-byte CCSIDs (for those registered to date).

The CS element associated with the returned CCSID is the largest in size of all the character sets that are contained in its associated code page in the CCSID resource installed in the system.

Notes:

1. Not all the registered code pages are full, leaving room for possible future additions. The term maximal is used to distinguish it from a full code page. Over time, when new larger character sets are registered for a code page (this can happen until the code page is full), the maximal character set also changes, leading to a new CCSID returned by this function.
2. This API does NOT support any mixed or multibyte CCSIDs.

Required Parameters

CPIN INPUT; INT(4)

This variable contains the CP value referenced, a positive number in the range 1 to 65,534.

ESIN INPUT; INT(4)

This variable contains the ES value referenced;

ESIN
0

Meaning

The caller does not know the ES value, and expects the first CCSID encountered in the CCSID repository, with the specified CP and the Full or Maximal CS, to be returned.

Other

The caller specifies the ESID value, a positive number in the range 4352 to 65,534. Only ESIDs that have a single (CS, CP) pair associated with them are valid for this function.

CCSIDR

OUTPUT; INT(4)

This variable contains the returned CCSID value, a positive number in the range 1 to 65,279.

ESR OUTPUT; INT(4)

This variable contains the ES value of the returned CCSID, a zero or a positive number in the range 4352 to 65,534.

FB OUTPUT; FEEDBACK

The function returns, in this feedback array, the processing status (and any associated reason) for this function the field type is array of three 32-bit two's complement binary values (12 bytes, or 96 bits); the status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in hexadecimal) contained in the first 32 bits of FB, (the other 64 bits are reserved).

Feedback Codes and Conditions

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

Usage Notes

1. Often it is required to find the CCSID when the ES, CS, and CP values are known. CS and CP (also known as CGCSGID or GCID) is used in many existing IBM architectures and data streams and supporting products. Together with the CDRSCSP API, this function aids in coexistence and migration for products that have to deal with the short form (CCSID) of identification on one side and the intermediate form (CPGID) on the other. Because the intermediate forms are, by themselves, incomplete when used in some encoding schemes, the function can return only a default value as defined in the installation's resources, when the ESIN information is unknown.
2. Some code page identifiers in use in the Far East refer to "pseudo" or "combined" code pages with the PC mixed and host mixed encoding schemes. These identifiers are to be used as CCSIDs rather than CPGIDs. The CDRSMXC function does not return the corresponding CCSIDs for these combined code page identifier values. Please refer to the documentation for the QTQSMXC2 API for support of some of these code page identifiers.
3. When an ESIN value of zero is specified, the feedback code is never returned as zero. Absence of an error results in a 0001 0003 feedback code.

API introduced: V3R6

[top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

Get Short Form with Maximal CS for Specified ES and CP (QTQSMXC2) API

Required Parameter Group:	
1	CPIN
Input	Int(4)
2	ESIN
Input	Int(4)
3	CCSIDR
Output	Int(4)
4	ESR
Output	Int(4)
5	FB
Output	Feedback
	Default Public Authority: *USE
Threadsafe: Yes	

The Get Short Form with Maximal CS for Specified ES and CP (QTQSMXC2) API gets the CCSID with the maximal character set (CS) for a given code page (CP) value. The encoding scheme (ES) parameter can be specified to distinguish between usage of the same CP with two different encoding schemes, such as PC Display compared to PC Data.

The CS element that is associated with the returned CCSID is the largest in size of all the character sets that are contained in the associated code page for that CS element in the CCSID resource installed on the system.

The QTQSMXC2 API acts as an extension of the functions of the CDRSMXC API in that you can specify more code pages. The additional code pages are as follows:

- 932
- 934
- 936
- 938
- 942
- 944
- 946
- 948
- 949
- 950
- 1381

Note: Not all the registered code pages are full, leaving room for possible future additions. The term maximal is used to distinguish it from a full code page. Over time, when new larger character sets are registered for a code page (this can happen until the code page is full), the maximal character set also changes, leading to a new CCSID returned by this function.

Required Parameters

CPIN INPUT; INT(4)

A variable that contains the CP value referenced. This value must be a positive number in the range 1 to 65 534.

ESIN INPUT; INT(4)

A variable that contains the ES value referenced.

Possible values follow:

0

The caller does not know the ES value, and expects the first CCSID encountered in the CCSID repository, with the specified CP and the full or maximal CS, to be returned.

4352-65 534

The caller specifies the encoding scheme value. Only encoding schemes that have a single (CS, CP) pair associated with them are valid for this function.

CCSIDR

OUTPUT; INT(4)

A variable that contains the returned CCSID value. This value must be a positive number in the range 1 to 65 279.

ESR OUTPUT; INT(4)

A variable that contains the ES value of the returned CCSID. This value must be zero or a positive number in the range 4352 to 65 534.

FB OUTPUT; FEEDBACK

In this feedback array this function returns the processing status (and any associated reason) for this function. The field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a non-negative number in the first 16 bits, and the reason code is a

non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in hexadecimal) contained in the first 32 bits of FB. The other 64 bits are reserved.

Feedback Codes and Conditions

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

Usage Notes

- Often it is required to find the CCSID when the ES, CS, and CP values are known. CS and CP (also known as CGCSGID or GCID) are used in many existing IBM architectures, data streams, and supporting products. Together with the CDRSCSP API, this function aids in the coexistence and migration for products that have to deal with the short form (CCSID) of identification on one side and the intermediate form (the code page global identifier) on the other. Because the intermediate forms are, by themselves, incomplete when used in some encoding schemes, the function can return only a default value as defined in the installation's resources, when the ESIN information is unknown.
- When an ESIN value of zero is specified, the feedback code is never returned as zero. Absence of an error results in a 0001 0003 feedback code.

API introduced: V4R1

[Top](#) | ["National Language Support APIs," on page 1](#) | [APIs by category](#)

Resolve Client Server CCSID (QTQRCSC) API

Required Parameter Group:	
1	Server CCSID
Input	Int(4)
2	Client CCSID
Input	Int(4)
3	Selection
Input	Int(4)
4	Resolved CCSID
Output	Int(4)
5	Feedback code
Output	Feedback Default Public Authority: *USE
Threadsafe: No	

The Resolve Client Server CCSID (QTQRCSC) API returns the CCSID that will enable the proper transfer of data between the server and the client.

Required Parameters

Server CCSID

INPUT; INT(4)

A variable that contains the CCSID value for the server. This value must be a positive number in the range 1 to 65 533.

Possible values follow:

0

The caller does not know the server job value. The default job CCSID value will be used.

1-65 533

The server CCSID value.

Client CCSID

INPUT; INT(4)

A variable that contains the CCSID value for the client. This value must be a positive number in the range 1 to 65 533.

Selection

INPUT; INT(4)

A variable that allows the user to specify what CCSID to return. If the resolved CCSID is not supported as a job CCSID for the system on which the API is being called, the API returns either that CCSID or finds the closest match for the system it is on. If the exact match is to be returned, but the CCSID is not supported on the host system, the feedback code is set.

Possible values follow:

0	Return the closest match for this system. If the client CCSID is mixed and the server system only handles single byte, find the closest match SBCS CCSID value.
1	Return an exact match for the resolved CCSID. The API will not take into account the available support on the server. The feedback code will be set if the CCSID value returned is not compatible with the system on which the API is being called.

Resolved CCSID

OUTPUT; INT(4)

A variable that contains the resolved CCSID value. This value must be a positive number in the range 1 to 65 279.

Feedback code

OUTPUT; FEEDBACK

In this feedback array, the function returns the processing status (and any associated reason) for this function. The field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The following are specific meanings of the status code and associated reason code values (in hexadecimal) contained in the first 32 bits of the feedback code. The other 64 bits are reserved.

Feedback Codes and Conditions

Status	Reason	Meaning
0000	0000	The function completed successfully.
0001	0007	The user requested a CCSID for a mixed system, and the API is being called on an SBCS system. An exact match was requested, so a warning is returned to indicate that the CCSID returned is not valid as a job CCSID for the system on which this API is running.
0006	0001	The CCSID resource repository was not found.
0007	0001	The system CCSID resource repository accessed by the function was found to be incorrect in the structure.

Usage Notes

1. This API uses the Get Related Default CCSID (CDRGRDC) and Get Encoding Scheme, Character Set, and Code Page Elements (CDRGESP) APIs to perform its functions. If any errors are found on the calls to these APIs, the feedback code is set to those values. Refer to the documentation for these two APIs for more feedback codes.

API introduced: V4R1

Top | "National Language Support APIs," on page 1 | APIs by category

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Advanced 36
Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI

DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and conditions for downloading and printing publications

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

Personal Use: You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM^(R).

Commercial Use: You may reproduce, distribute and display this information solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of this information, or reproduce, distribute or display this information or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the information or any data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the information is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THIS INFORMATION. THE INFORMATION IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

All material copyrighted by IBM Corporation.

By downloading or printing information from this site, you have indicated your agreement with these terms and conditions.

Code disclaimer information

This document contains programming examples.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM^(R), ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.



Printed in USA