Mainframe code page conversion customization in an IBM z/OS system

Hong Liang Han  
Xiao Lin Zhang  
Si Bin Fan  

February 02, 2012

Encoding in mainframes is EBCDIC, and encoding in distributed system is ASCII. Mainframe computers need to communicate with distributed systems that include Microsoft® Windows®, UNIX®, and other platforms. The EBCDIC and ASCII code page conversion happens in each communication between the IBM® z/OS® operating system and the distributed system. However, some characters cannot go round trip between EBCDIC and ASCII in some non-English languages. As a result, the application doesn't work correctly. One solution is to modify application code, but that is often complicated and time consuming. This article presents a lightweight solution as an alternative. No application code change is needed, and you can resolve the issue quickly.

Introduction

Over decades, the IT systems in many enterprises have evolved into multi-platform environments. The mainframe often plays a key role in the business, and many other business applications run on distributed platforms such as Microsoft Windows, UNIX, and Linux. As a result of this evolution, communication between the heterogeneous systems has become a mandatory and common need. However, because some characters cannot go round trip between EBCDIC and ASCII code pages in some non-English languages, problems often arise during the code page conversion between the mainframe and the distributed platforms. This can result in the applications not functioning properly. This article presents a solution to resolve this issue quickly with no need to change the application code, and which will benefit the application developers and the end users who are involved with both the distributed side and the mainframe side.

The customization process of Unicode conversion service in IBM z/OS is introduced in the first step, and the non-round-trip conversion issues are illustrated in the file transferring between z/OS and distributed systems.

With the solution provided in this article, you will learn how to customize the code page conversion to support characters transferring from the mainframe side to the distributed side, and vice versa.
Prerequisites

You will need an intermediate level of knowledge to follow this article, and it is based on the assumption that you have knowledge of Unicode and z/OS skills.

Business scenario

Enterprise users often need to transfer files from the mainframe to the distributed system side. More often, they need to edit the files on the distributed side and transfer them back to the mainframe later. The distributed side, which includes Windows, UNIX and Linux platforms, uses ASCII code pages. The mainframe side, typically running IBM z/OS, uses EBCDIC as the encoding. The conversion of some characters cannot travel round trip between EBCDIC and ASCII code pages. Even the Unicode and EBCDIC code pages can not be converted in a round trip way. The non-round-trip conversion characters vary from code page to code page.

For example, IBM z/OS users usually prefer using characters such as $, #, and @ in the data set names, PDS member names, and variable names in the source codes. For z/OS users in non-English environments, they often need to use non-English characters in the source codes. However, some characters can be messed up during the round trip transfer between the mainframe and the distributed side. For instance, the ¥ sign in the CP1388 code page is converted to a non-displayable character in CP5488 code page in the IBM AIX® system Chinese environment. This is a problem for the end users because they see unreadable codes in both the distributed side and in the mainframe side after the round trip.

Now let’s start to explore a practical solution for this scenario.

The Unicode conversion service (UCS)

Unicode is a coding standard that supports all known characters in all of the languages in the world, plus the classical and historic text. The code page conversion converts the character from one coded character set identifier (CCSID) to another.

In IBM z/OS, the code page conversion is based on Unicode environment to provide the infrastructure.

- The code page with a pure single-byte is called a single-byte character set, or SBCS.
- The double-byte character set, or DBCS, contains only double-byte code points. The UCS-2 is a kind of DBCS code page. They are all simple code pages.
- MBCS, the multi-byte character set code pages, consist of two or more sub-code pages.

The z/OS code page conversion can be either a direct or indirect conversion. The direct conversion usually contains SBCS to SBCS, and DBCS to DBCS within the EBCDIC, ASCII, and Unicode. The indirect conversion is to use the CCSID 1200 as the intermediate code pages. The conversion of MBCS characters is a composite conversion. An MBCS input data stream is decomposed into SBCS and DBCS parts. The conversion of MBCS uses an SBCS conversion table for the SBCS data and a DBCS conversion table for the DBCS data.
The EBCDIC DBCS code page is converted to a Unicode code page, and then it is converted from the Unicode code page to the ASCII code page in the indirect conversion. The conversion from an ASCII code page to an EBCDIC code page is the same when using the indirect conversion. Both the direct conversion and the indirect conversion can be customized by the method introduced in this article.

**The UCS customization process**

The UCS customization process includes the following six steps:

1. Get the original conversion table in text mode.
2. Customize the conversion table in text mode.
3. Generate the binary conversion table.
4. Apply the conversion table to the system.
5. Update the z/OS system for the next IPL.
6. Update the application runtime environment to pick up the new conversion table.

Now you will use a concrete example to illustrate the detailed steps in the UCS customization.

An IBM Rational ClearCase customer in China is using some Chinese characters in their COBOL source codes.

One of the characters is the ¥ (\(<0x5B>\)) in IBM-1388, which is entered by using the CTRL + backslash in IBM Personal Communications when IBM-1388 code page is specified. When the source file is transferred from the mainframe to the AIX Chinese environment for some actions, such as compare and merge, the ¥ is converted to a non-displayable character. This makes the codes unreadable for compare and merge, so the customer expects the ¥ to be converted to $ (\(<0x24>\)) in AIX so that they can merge the file and transfer it back to the mainframe to achieve the round trip without messing up the codes.

The UCS customization solution can resolve the above issue perfectly. First, you need to figure out the code pages involved in the conversions in this scenario. You can find out the sub-CCSIDs of the MBCS CCSID in the MBCS CCSID decomposition table in the Related topics section.

CP13124 is the SBCS code page of CP1388, and CP13488 is the Unicode code page to which the SBCS characters in CP13124 are converted. Then you need to perform the following steps for the customization.

1. Allocate the data set of the JCL, the text map, and the binary map. The JCL data set should be DCB(RECFM=FB,LRECL=80). The text map data set should be DCB(RECFM=FB,LRECL=80). The binary map data set should be DCB(RECFM=FB,LRECL=256).
2. Customize the conversion between CP13124 and CP13488 as shown in Listings 1, 2, and 3.
Listing 1. JCL to get the original text conversion table

```
//MP1DXPG  JOB ,CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1),NOTIFY=&SYSUID
  /*JOBPARM  S=ADCD
//CUNMRATB EXEC PGM=CUNMITG1,PARM='13124,13488,R'
//TABIN DD DISP=SHR,DSN=SYS1.SCUNTBL
//CHAROUT DD DISP=SHR,DSN=IBMUSER.CUNUNI.TEXTMAP(MAP1DXPG)
//SYSPRINT DD SYSOUT=* 
//CUNTXT   EXEC PGM=CUNMITG1,PARM='13488,13124,E'
//TABIN DD DISP=SHR,DSN=SYS1.SCUNTBL
//CHAROUT DD DISP=SHR,DSN=IBMUSER.CUNUNI.TEXTMAP(MAP1PGDX)
//SYSPRINT DD SYSOUT=* 
/* THE END
```

Modify the MAP1DXPG member in IBMUSER.CUNUNI.TEXTMAP as shown in Listing 2.

**Listing 2. MAP1DXPG member**

```
%  <5B>  <00A5>   % % This is the comment line
  <5B>  <0024>
%  <B2>  <005C>   % % This is the comment line
  <B2>  <00A5>
%  <E0>  <0024>   % % This is the comment line
  <E0>  <005C>
```

Modify the MAP1PGDX member in IBMUSER.CUNUNI.TEXTMAP as shown in Listing 3.

**Listing 3. Map1PGDX member**

```
%  <0024>  <E0>
%  <0024>  <5B>
%  <005C>  <B2>
%  <005C>  <E0>
%  <00A5>  <5B>
%  <00A5>  <B2>
```

3. Create the binary files and copy those to SYS1.SCUNTBL. Submit the following JCL to generate the binary conversion table, as shown in Listing 4.

**Listing 4. Submit JCL**

```
//TOBIN    JOB UNI,'TEST',MSGCLASS=H,NOTIFY=&SYSUID,REGION=1600M
//TOBDXPG  EXEC PGM=CUNMITG2,PARM='13124,13488,1'
//CHARIN   DD DISP=SHR,DSN=IBMUSER.CUNUNI.TEXTMAP(MAP1DXPG)
//TABOUT   DD DISP=SHR,DSN=IBMUSER.CUNUNI.BINMAP
//SYSPRINT DD SYSOUT=* 
//TOBPGDX  EXEC PGM=CUNMITG2,PARM='13488,13124,1'
//CHARIN   DD DISP=SHR,DSN=IBMUSER.CUNUNI.TEXTMAP(MAP1PGDX)
//TABOUT   DD DISP=SHR,DSN=IBMUSER.CUNUNI.BINMAP
//SYSPRINT DD SYSOUT=* 
```

And then copy the two members, MAP1DXPG and MAP1PGDX, from IBMUSER.CUNUNI.BINMAP to SYS1.SCUNTBL.

4. Load the new tables as shown in Listings 5 through 7. First, issue the command, shown in Listing 5, in the SDSF command line to delete the current loaded conversion tables:
Listing 5. Issue first SDSF command

SETUNI DEL ALL FORCE=YES

Second, issue the commands shown in Listing 6 in SDSF command line.

Listing 6. Issue second and third SDSF commands

SETUNI ADD, FROM(13124), TO(13488), TECHNIQUE(1), DSNAME(SYS1.SCUNTBL)
SETUNI ADD, FROM(13488), TO(13124), TECHNIQUE(1), DSNAME(SYS1.SCUNTBL)

Finally, issue the command shown in Listing 7 to display the current loaded conversion table.

Listing 7. Current loaded conversion table

DISPLAY UNI,CONVERSION

The following two tables shown in Listing 8 should be in the output.

Listing 8. Output

01200(13488)-13124-1
13124-01200(13488)-1

5. Update the system for the next IPL by running the following JCL to create the CUNIMG00, as shown in Listing 9.

Listing 9. JCL code

//*******************************************************************
//*                                                                 *
//* LICENSED MATERIALS - PROPERTY OF IBM                             *
//*                                                                 *
//* 5637-A01                                                         *
//*                                                             @L1C*
//* (C) COPYRIGHT IBM CORP. 2004, 2005                             @L1C*
//*                                                                 *
//* STATUS = HUN7720                                                 *
//*                                                                 *
//* $L1=MG03727 HUN7720 040422 JR: Change current program number to 5637-A01 @L1A*
//* //**                                                                   *
//* //** IMAGE GENERATOR                                                *
//* //**                                                                   *
="/*******************************************************************
//CUNMIUTL EXEC PGM=CUNMIUTL
//SYSPRINT DD   SYSOUT=*                                           *
//TABIN    DD   DSN=SYS1.SCUNTBL,DISP=SHR
//         DD   DSN=IBMUSER.CUNUNI.BINMAP,DISP=SHR
//SYSIMG   DD   DSN=IBMUSER.CUNUNI.IMAGES(CUNIMG00),DISP=SHR
//SYSIN    DD   *
/*****************************/
*      INPUT STATEMENTS FOR THE IMAGE GENERATOR   *
*****************************/
NORMALIZE;       /* ENABLE NORMALIZATION       */
COLLATE;         /* ENABLE COLATION            */
CASE NORMAL;     /* ENABLE TOUPPER AND TOLOWER */
CASE LOCALE; /* ENABLE LOCALE */
CASE SPECIAL; /* ENABLE SPECIAL */
CONVERSION 13124,13488,1; /* EBCDIC -> ASCII */
CONVERSION 13488,13124,1; /* ASCII -> EBCDIC */

6. To ensure that the table shown previously in Listing 9 is included on future IPLs, update the CUNUNIxx parmlib member by doing the following.

   a. Copy the member CUNIMG00 from IBMUSER.CUNUNI.IMAGES to SYS1.PARMLIB.
   
   b. CUNUNIxx contains information that Unicode Services uses to define its environment.

   Select a CUNUNIxx parmlib member by specifying the UNI=xx keyword in IEASYSxx.

**Listing 10. Example**

```plaintext
  /**********************************************************************
  /* CUNUNIXX - UNICODE CONVERSION CONTROL PARAMETERS */
  /**********************************************************************
  /* ESTABLISH A NEW ENVIRONMENT */
  /**********************************************************************
  /* REQUIRED KEYWORD REALSTORAGE */
  /* MAXIMAL USED PAGES OF REAL STORAGE, MIN=0 MAX=524287 */
  /* WHERE 0 MEANS NO EXPLICITE LIMIT (=524287) */
  /**********************************************************************
  REALSTORAGE 51200; /* E.G. 200 MB */
  /**********************************************************************
  /* REQUIRED KEYWORD IMAGE WITH */
  /* REQUIRED PARAMETER: MEMBER NAME */
  /* THIS MEMBER MUST BE PLACED IN A DATA SET FROM THE */
  /* LOGICAL PARMLIB CONCATENATION (DEF'D IN LOADXX) */
  IMAGE CUNIMG00;
  /**********************************************************************

7. Update the application runtime environment to use the new tables. To load the new added conversion table, you need to add the two environment variables, shown in Listing 11, in your application runtime environment.

**Listing 11. Output**

```plaintext
_ICONV_TECHNIQUE
_ICONV_MODE
```

You can use z/OS UNIX profile to export the two ENVARS in system level, or set it in the specific application runtime. You can allocate the EDCENV sequential data set with following contents shown in Listing 12.

**Listing 12. EDCENV**

```plaintext
_ICONV_TECHNIQUE=1LMREC
_ICONV_MODE=C
```

The example shown in Listing 13 shows the usage of SELECT in an REXX application.
Listing 13. SELECT in REXX

```
"ALLOC FI(EDCENV) DA('"HLQ".EDCENV") SHR "
ENVARS='ENVAR("_CEE_ENVFILE=DD:EDCENV")'
"SELECT PGM(YourPGM) , PARM('"ENVARS"/"Your Other Parameters").
```

The example shown in Listing 14 shows using CALL.

**Listing 14. Call output**

```
"Call '"HLQ".MLQ.LLQ(YourPGM)' '"ENVARS"/"Your Other Parameters"
```

The example shown in Listing 15 shows using batch JCL.

**Listing 15. JCL output**

```
//JOBCARD ...
//RUNPGM EXEC PGM=YourPGM,
// PARM='ENVAR("_CEE_ENVFILE=DD:EDCENV")'
//EDCENV DD DSN=&HLQ..EDCENV,DISP=SHR
//OtherDD DD ...
```

**Other considerations**

The method in this article can be used when your application can be customized by the REXX or JCL to transfer the technique to your application runtime environment. It is convenient and easy to satisfy your specific requirement.

But when your application can accept only the CCSIDs, or only the fixed technique, you must define a new CCSID and create new conversion tables to the FROM- CCSID and TO- CCSID.

For example, the Enterprise COBOL uses the default technique search order (RECLM), while IBM DB2 uses the ER technique search order. If your application is written by Enterprise COBOL, and accesses DB2, you must define the new CCSIDs. But contact IBM to check whether they already have the CCSIDs for your requirement.

**Conclusion**

The solution introduced in this article applies to all of the conversions in different code pages and different platforms. You have now learned how to customize the code page conversion to support correctly transferring characters from the mainframe side to the distributed side, and vice versa.

**Acknowledgement**

Special thanks to Pat Glenski who gave technical support on the Chinese and Korean code page customization in the actual customer situation. Thanks also to Xue Ming Zuo for the review and the refinement suggestions for this article.
Related topics

- Find out the sub-CCSIDs of the MBCS CCSID in the "MBCS CCSID decomposition table."
- Read the z/OS Unicode Services "User's Guide and Reference."
- Review the "z/OS XL C/C++ Programming Guide."
- Learn more about "z/OS installation" from the DB2 Version 9.1 for z/OS Installation Guide.
- Read the Enterprise COBOL for z/OS "Customization Guide."
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.