Banner
Pro*C Toolkit

Release 8.0
May 2008

SUNGARD HIGHER EDUCATION

What can we help you achieve?

SunGard Higher Education

4 Country View Road
Malvern, Pennsylvania 19355
United States of America
(800) 522 - 4827

Customer Support Center Website

http://connect.sungardhe.com

Documentation Feedback

http://education.sungardhe.com/survey/documentation.html

Distribution Services E-mail Address

distserv@sungardhe.com

Other Services

In preparing and providing this publication, SunGard Higher Education is not rendering legal, accounting, or other similar professional services. SunGard Higher Education makes no claims that an institution's use of this publication or the software for which it is provided will insure compliance with applicable federal or state laws, rules, or regulations. Each organization should seek legal, accounting and other similar professional services from competent providers of the organization's own choosing.

Trademark

Without limitation, SunGard, the SunGard logo, Banner, Campus Pipeline, Luminis, PowerCAMPUS, Matrix, and Plus are trademarks or registered trademarks of SunGard Data Systems Inc. or its subsidiaries in the U.S. and other countries. Third-party names and marks referenced herein are trademarks or registered trademarks of their respective owners.

Revision History Log

Publication Date	Summary
May 2008	New version that supports 8.0 software.

Contents

Notice to licensee

By downloading the software and any other materials mentioned in this guide, you, on behalf of your institution ("Licensee") are agreeing to the following terms, conditions, and limitations. Do not download or otherwise access these files unless you have read and understand the following terms, conditions, and limitations. By downloading or otherwise accessing any of these files, you, on behalf of the Licensee, are acknowledging and agreeing that you in fact have read, do understand, and are agreeing to the following terms, conditions, and limitations, and that the Licensee is bound by the following terms, conditions, and limitations:

- Use of Files Subject To License Agreement. The software and any other materials contained in this file ("Licensed Materials") are licensed for use by the Licensee, subject to the written, fully executed Software License and Services Agreement ("License Agreement") between the Licensee and SunGard Higher Education.
- Use of Files Subject To License Agreement. As between the Licensee and SunGard Higher Education, SunGard Higher Education is the owner of all right, title and interest in and to the Licensed Materials, including all copyright in and to the Licensed Materials. The Licensed Materials are Confidential Information of SunGard Higher Education, in whole and in part.
- Licensed Materials Are Not Supported By SunGard Higher Education. The Licensed Materials are not supported by SunGard Higher Education, are not subject to any warranty or maintenance program provided by or endorsed by SunGard Higher Education.
- Licensed Materials Provided on an "as is" basis; no liability on the part of SunGard Higher Education. Licensee acknowledges and agrees that the Licensed Materials are made available to Licensee on an "as is" basis; that SunGard Higher Education makes no warranties regarding the accuracy of the Licensed Materials, in whole or in part; that SunGard Higher Education hereby disclaims any and all warranties, express or implied, including without limitation any implied warranties of merchantability, non-infringement and/or of fitness for a particular purpose with regard to the Licensed Materials, in whole and in part; and that in no event will SunGard Higher Education have any liability whatsoever in connection with the Licensed Materials for the use thereof, in whole or in part, whether for direct, indirect, special, incidental, consequential, or any other type of damages.
- Authority To Bind. The person accessing the Licensed Materials on behalf of Licensee has the power and authority bind Licensee to these terms, conditions and limitations.

Disclaimer

This software and the accompanying files are distributed "as is" and without any warranties whether expressed or implied. No responsibilities for possible damages or even functionality can be taken. The user must assume the entire risk of using this program.

Limitations

This software is only intended to be used to assist in the SunGard Higher Education Banner 8.0 conversion process.Use of this software with non-Banner compliant code may produce undesired results or may not work at all.

Introduction

Since the Banner 8 environment uses a Unicode/UTF-8 character set, the Pro*C processes have been modified. International Component for Unicode (ICU) is a set of C/C++ libraries that have been created by IBM for use in developing globalized software. The new libraries created for Banner 8 call the ICU libraries, which mandates changes in coding practices for the Pro*C processes.

To assist in updating Pro*C processes to conform to the new standards, SunGard Higher Education is providing you with a rekey script. This script is a batch process used to renumber G$_NLS.Get keys that are used to identify message strings that can be isolated for easier translation.

This guide describes what the rekey script does, how to install the rekey script, and how to use it to validate your Pro*C processes so that they will compile correctly for the new Banner 8 database. This guide also provides information on how you update your Pro*C processes prior to running the script and how to troubleshoot compilations errors.

Updating your Pro*C processes

Before you run rekey, you should update your Pro*C processes to review the string literals to make sure they adhere to proper standards for TM_NLS_GET. You also need to make sure each header in a Pro*C process includes the following code:

```
#include <stdio.h>
   #include "tmcilib.h"
   #define SCT_DEBUG_LEVEL 3
   struct TMBundle tmBundle={<filename>,NULL,NULL,NULL,NULL};
int globalCount=1;
```

Finally, you need to review the variable declarations that have the following format:

```
char15 var[35]
```

These need to use the following new format:

```
TMCHAR var[35][15]
```

The following table outlines how hardcoded strings in C code should be modified for I18N support. You'll notice that a number of examples illustrate the need for a more elegant approach than simply making the string argument of *TM_NLS_GET(...)*.

Before I18N	After I18N
`char str1;`	`TMCHAR str1;`
`char str3[256]="B string";`	`TMCHAR TMCHARRYT_GLOB_DCL(str3,256,TM_NLS_Get ("0001","B String"));`
`strcpy(str2,"E string");`	`tmstrcpy(str2,TM_NLS_GET(&tmBundle,"0004","E string");`
`fprintf("A number: %d\n", num);`	`tmfprintf("A number: {0,number,integer}\n", num);`
`FILE *infile;`	`UFILE *infile;`
`infile=fopen(fname, openmode);`	`infile=tmfopen(fname,openmode);`

`static const char *messages[] = {`
`"some very meaningful message",`
`"and another one"`
`};`
	`static TMCHAR *messages = { NULL,NULL };`
	`TMCHRPTRARRY_GLOB (messages)={`
	`Messages[0]=TM_NLS_Get("X", "some very meaningful message");`
	`Messages[1]=TM_NLS_Get("X","and another one");`
	`}`

Before I18N	After I18N
#include <stdio.h> #define SCT_DEBUG_LEVEL 3 int globalCount=1;	#include <stdio.h> #include “tmcilib.h” #define SCT_DEBUG_LEVEL 3 struct TMBundle tmBundle={<filename>,NULL,NULL,NULL,NULL}; int globalCount=1;
printf (cis_sh_msg("GUAPARM-0011", &message)?message:"RETRIEVED: param RUNDATE = %s, status = %d\n",param,status);	tmprintf (TM_NLS_Get(&tmBundle,”0011”,"RETRIEVED: param RUNDATE = %s, status = %d\n"),param,status);
CHAR15 var[35]	TMCHAR var[35][15];

Variable Declaration and Initialization examples

As you are updating your Pro*C code, you will encounter a number of common issues related to variable declarations and initialization that need to be modified to better support I18N.

The following table illustrates how variables were formerly declared and initialized and how they should be handled now.

Pre I18N	Post I18N
static NUMSTR ask_one_up_no_var="";	static BANNUSTR(ask_one_up_no_var)={0};
static CHAR2 continue_sw="";	static TMCHAR continue_sw[2]={0};

The following table illustrates the incorrect way to write variable declarations (on a single line) and the correct way.

Incorrect	Correct
CHAR2 var1, var2,var3;	CHAR2 var1; CHAR2 var2; CHAR3 var3;

The following table illustrates how a static storage class should not be used if the variable is being initialized with an output of a method in local scope.

Incorrect	Correct
Static TMCHAR *str[] =	TMCHAR *str[] =
{ _TMC(“1”), _ TMC(“2”) .. };	{ _TMC(“1”), _ TMC(“2”) .. };

The parser needs to push the static string initializations (char strx[..] =”string”;) on a FIFO stack until the end of the local variable declaration/initialization. Then the strings from the resource bundles need to be *tmstrcpy’d* to static strings before the beginning of the processing.

The number between the square brackets must be filled with a value giving space for longer translations. For example, in the following code you could use a formula like “t = 20 + 1.3 * s” where s=length of original string and t is the new string length:

```
Parse (char str3[256]=”B string”;)
    push on stack (name,string)=(“str3”,”B string”)
    newlength=20+1.3*256
    Current line to be replaced with:
    TMCHAR str3[352];
Parse (char str4[]=”C string”;)
    push on stack (name, string)=(“str4”, “C string”)
    newlength=20 +1.3*strlen(“C string”)
    Current line to be replaced with:
    TMCHAR str4[30];
…
If end local variable declaration/initialization found then
    Pull first entry from stack:  (name, string)=(“str3”, “B string”)
    Insert the initialization of the string:
    tmstrcpy(str3,TM_NLS_GET(….., “B string”));
    Pull second entry from stack: (name, string)=(“str4”, “C string”)
    Insert the initialization of the string:
    tmstrcpy(str4,TM_NLS_GET(….., “C string”));
    …
End if
```

Now consider the following example:

```
{
  static const char *messages[] = {
    “some very meaningful message”,
    “and another one”
  };
```

```
  const char *string;
  …
  string
    = index > 1 ? “a default message” : messages[index];
  fputs (string);
  …
}
```

While it is not a problem to internationalize the “`a default message`” string, it is not possible to internationalize the string initializers for `messages`. The strings have to be translated at runtime before printing them.

One possible solution is to define a no-op *TM_NLS_Init* and find out all access points to a string from the array, which would look like the following:

```
#define TM_NLS_Init(String) (String)

{
  static const char *messages[] = {
    TM_NLS_Init(“0000.some very meaningful message”),
    TM_NLS_Init(“0001.and another one”)
  };
  const char *string;
  …
  string
    = index > 1 ? TM_NLS_Get(“0002”,”a default message”) :
TM_NLS_Get_Static(&tmBundle, messages[index]);
  fputs (string);
  …
}
```

In *TM_NLS_Init* and *TM_NLS_Get_Static*, the key and the default string are separated by a period. At position 5 `messages[index]` cannot contain a key and a default string separately.

Another way to solve the problem of using `static const` is to manually move code into compilation using an initialization routine so it is no longer static, in which case you could then use the standard *TM_NLS_Get*.

Installing and running the rekey script

Rekey is a batch process that automates the renumbering of G$_NLS.Get keys that are used when character strings have to be displayed in the user interface which may need translation. The rekey script does not add in any G$_NLS.Get calls.

The *G$_NLS.Get* calls use a format similar to the following:

```
g$_nls.get( <key>, <type>, <message> [,variables] )
```

The rekey script replaces the `<key>` element with a filename of the following format:

```
FILENAME-NNNN
```

The `FILENAME` uses uppercase without an extension for known type. The `NNNN` portion is a sequence number, starting with 0000. This sequence number is increased after each replacement of a key in the `g$_nls.get` statement. The `<key>` can be any text literal. The `<type>` portion of the `g$_nls.get` statement is not touched, as it is assumed to be correctly entered by the programmer. The `<message>` portion needs to be a translatable text literal presented between single quotes.

The rekey script also replaces incorrect usage of `g$_nls.get` with the more efficient `g$_nls.formatMsg`, which is a function that does not do a message translation look up, but instead substitutes the variable placeholders (like %01%) with the variable values. The following usages are considered incorrect.

- **<message> is a variable.** Variables are only known at runtime, so TranMan cannot extract the text value to present it to the translators for translation. If the variable will contain translatable text, `g$_nls.get` should be used where the variable gets the value from a text literal.
- **<message> is '%01%'.** In this trivial case, no translation will be needed. This code may have been created by older versions of TranMan when `g$_nls.get` was used with a variable (as in the previous bullet).

Finally, rekey adds an audit trail as defined in tm.ini for the current installation.

Installing rekey

To install rekey, use the following procedure:

1. Unzip `rekey1.1` to your computer or a network mapping that can be accessed by all team members.

   For example, you may want to unzip it to the root of your C drive, as illustrated below:

```
C:\rekey1.1
```

For purposes of these instructions, we will refer to this location. As you configure rekey in the steps that follow, change the path to the actual location where you extracted `rekey1.1` on your system.

2. Access the directory where you extracted `rekey1.1`.

3. Using an editor, open the `rekey_fd.bat` file.

4. Locate the following lines and set them appropriately for the location where you extracted `rekey1.1`.

```
TM_HOME=<extraction path>\tmhome
REKEY_HOME=<extraction path>\rekey
```

For example, if you extracted rekey to `c:\rekey1.1`, you'd edit these lines as follows:

```
TM_HOME=C:\rekey1.1\tmhome
REKEY_HOME=C:\rekey1.1\rekey
```

5. If you are using forms 10g release 2, locate DV=90 and change it as follows:

```
DV=101
```

6. If you want to make use of the context sensitive SendTo menu in Windows Explorer do the following:

   6.1. Open the Windows Explorer and navigate to the following directory:

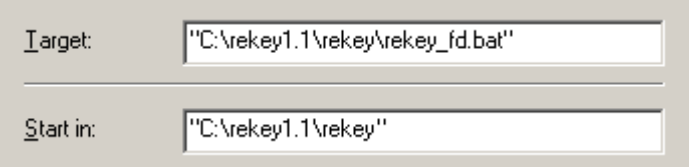
   ```
   <rekey extraction path>\i18nCtrMenu\
   ```

   6.2. Right-click the rekey shortcut and select properties.

   6.3. Select the Shortcut tab and change the paths for the Target and Start in properties to specify the rekey extraction location as follows:

   ```
   Target: <rekey extraction path>\rekey\rekey_fd.bat
   Start in: <rekey extraction path>\rekey
   ```

   For example, if you had extracted rekey to `C:\rekey1.1`, these paths would be as follows:



6.4. Using an editor, open the UpdateMenu.bat file and change the path for the i18CtrMenu to specify the rekey extraction location as follows:

```
"<rekey extraction path>\i18nCtrMenu"
```

For example, with rekey extracted to C:\rekey.1.1, you would use the following path:

```
"C:\rekey1.1\i18nCtr Menu"
```

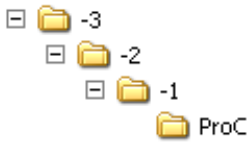
Once these configuration are done, users can install the SendTo menu by double clicking on UpdateMenu.bat

Running rekey

Any environment variables needed while processing your files can be specified in files with the name `fcmpenv.bat`. These files can be created from the following template:

```
<rekey extraction path>\fcmpenv_template.bat
```

The possible locations for the *fcmpenv* files that are being used are relative to the file you are processing. The locations can be in the directory of the file or up to three levels higher in the file tree as illustrated below:



In this structure, the order in which `fcmpenv.bat` files would be executed would be -3, -2, -1, then ProC.

If you have multiple branches in your source, you can specify common settings at the higher level and specific settings at the lower level.

When processing forms, make sure to create the `fcmpenv.bat` files in the relevant places.

The *rekey_fd* command is able to process files and directories as outlined below:

- rekey_fd file1 file2 file3
- rekey_fd dir1 dir2
- rekey_fd file1 dir1

You can run rekey by adding it to your PATH. Alternately, you can use the full path to call the commands.

When running rekey, note the following restrictions:

- The number of files and directories that can be passed is limited by Windows.
- Wildcards are not allowed.
- The paths of files and directories to process should not contain spaces.

Reviewing rekey output

After you run the rekey script, you should review the output for possible errors. The output will be in `<directory>_rekey`, which will be created if it does not exist already.

If you notice errors, you will need to resolve the issues and recompile your Pro*C processes. The next section provides information about troubleshooting compilations errors.

Troubleshooting compile errors

The following section provides a list of potential errors that you might receive while trying to compile your Pro*C processes and probable solutions for each.

Semantic error

The error message for a semantic error will appear similar to the following, where brackets indicate variable information:

```
Semantic error at line 2174, column 30, file <filename>.pcc:
   EXEC SQL PREPARE S4 FROM :xx_stmt;
```

The following table illustrates problematic code and the corrections that should be made:

Before	After

`EXEC SQL PREPARE S4 FROM :xx_stmt;`	`TMCHAR8 xx_stmt_c[2000];`
	`tmstrcpy8(xx_stmt_c, tmtochar8(xx_stmt));`
	`EXEC SQL PREPARE S4 FROM :xx_stmt_c;`

`tmmemset(&tmpvariable[0], 0, sizeof(tmpvariable[0]));`	`tmmemset((TMCHAR*)&tmpvariable[0], 0, tmcharsizeof(tmpvariable[0]));`

`struct parms_VALUES{`
`int total;`
`CHAR31 value[20];`
`short Ind_00[20];`
`int required;`
`} para[250], temp_para;`
	`struct parms_VALUES{`
	`int total;`
	`CHAR20 value[31];`
	`short Ind_00[20];`
	`int required;`
	`} para[250], temp_para;`

`dberror(__FILE__,__LINE__);`	`dberror(_TMV(__FILE__),__LINE__);`

Initializer element error

An initializer element error will be advertised through an error message similar to the following, where brackets indicate variables:

```
<filename>.pcc: In function `cmdhelp':
<filename>.pcc:1080: error: initializer element is not constant
<filename>.pcc:1080: error: (near initialization for `msg[0]')
<filename>.pcc:1081: error: initializer element is not constant
<filename>.pcc:1081: error: (near initialization for `msg[1]')
<filename>.pcc:1082: error: initializer element is not constant
<filename>.pcc:1082: error: (near initialization for `msg[2]')
```

For example, the following code is formatted incorrectly:

```
static TMCHAR *msg[]={
              _TMC("[-t] [-o output_file] [userid[/password]]"),
              _TMC("      -t Turn on sql_trace"),
              _TMC("      -o Specify output file "),
              NULL};
```

It should not be specified as static, as illustrated below:

```
  TMCHAR *msg[]={
              _TMC("[-t] [-o output_file] [userid[/password]]"),
              _TMC("      -t Turn on sql_trace"),
              _TMC("      -o Specify output file "),
              NULL};
```

Incompatible pointer type error

The error message for these conditions appears similar to the following, where brackets indicate variable information:

```
<filename>.pcc:10060: warning: passing arg 1 of `remove' from
incompatible pointer type
```

An incompatible pointer type error can be caused by code similar to the following:

```
remove(ps_input_file.fname);
```

This code should be written as follows to resolve issues:

```
remove((TMCHAR*)ps_input_file.fname);
```

Variable formatting errors

You may encounter errors relative to variables that appear similar to the following:

```
warning: passing arg 2 of `u_strcmp_2_8' makes pointer from integer
without a cast
```

To fix this code, you should break the declaration of the concerned variable into multiple lines. For an example, refer to the table on page 9.

Unsupported or undefined value errors

You may find errors related to unsupported or undefined values, similar to the following, where brackets are used to indicate variable information:

```
<filename>.pcc:160: warning: passing arg 1 of `u_strncpy_2_8' makes
pointer from integer without a cast
<filename>.pcc:160: error: subscripted value is neither array nor
pointer
```

These types of errors can result from code similar to the following:

```
static TMCHAR TMCHARRAY_GLOB_DCL(print_control_header,2,_TMC("N"));
```

This code should be modified as follows:

```
static CHAR2 TMCHARRYT_GLOB_DCL(print_control_header,2,_TMC("N"));
```

You could also perform this same correction in the GLOBAL STRING INITIALIZATION section, where you see code similar to the following:

```
TMCHARRAY_GLOB_INIT(print_control_header);
```

This code should be modified as follows:

```
TMCHARRYT_GLOB_INIT(print_control_header);
```

Segmentation fault errors

To resolve segmentation fault errors, you can use the following tactics:

- Replace calls to `str2uc` with `toUpper` as illustrated below:

```
static TMCHAR *toUpper(TMCHAR *utmp);   /*I18N*/
```

```
static TMCHAR *toUpper(TMCHAR *utmp)
  {
    TMCHAR  tmp_rtn[32];
    TMCHAR tmp_input[32];
    tmstrcpy(tmp_input,utmp);

    EXEC SQL SELECT UPPER(:tmp_input)
                INTO :tmp_rtn
                FROM DUAL;

    tmstrcpy(utmp,tmp_rtn);
    return utmp;

  }
```

- Change occurrences of `sizeof` to `tmcharsizeof`.
- Search for occurrences of `malloc` and increase the size to 3 times the current size.

These solutions are illustrated in the following table that provides before and after code samples:

Before	After
`static CHAR34 print_name[25];`	`static CHAR25 print_name[34];`
`static NUMSTR pell_percent_sched[3];`	`#define BANNMSTR(key,length) TMCHAR key[length][27] --ADD THIS IN THE DECLARATION SECTION` `static BANNMSTR (pell_percent_sched,3);`

`struct parms_VALUES{`
`int total;`
`CHAR31 value[20];`
`short Ind_00[20];`
`int required;`
`} para[250], temp_para;`
	`struct parms_VALUES{`
	`int total;`
	`CHAR20 value[31];`
	`short Ind_00[20];`
	`int required;`
	`} para[250], temp_para;`

`dberror(__FILE__,__LINE__);`	`dberror(_TMV(__FILE__),__LINE__);`

Commonly asked questions

The following section provides a list of common questions regarding coding practices to support I18N and how rekey is used.

Q: What happens if the code is not correctly internationalized?

The rekey utility relies upon correctly internationalized code. It is now a programming standard to write software that encapsulates user interface display strings in *G$_NLS.Get*. The rekey utility is passive and only inserts valid message numbers. Programs must be properly internationalized by inserting *G$_NLS.Get* in the places where it is missing.

Q: How does the key value differ from previous releases.

The previous key is only the value from the code when it is a proper key (MODULE_NAME-NNNN). In rekey the previous key has no use, it is only used when connected to a TranMan database where it serves as a hint to preserve translations after key renumbering. The translations are used with a minimal difference between previous key and stored key.

For example, you may see an error similar to the following:

```
Message: "*Error* - Go to key-block and provide all data for the
creation of the new record."
    Key: RVAIACA-0049, previous=RVAIACA-0049, max len=0, info=Key
update
```

The input that caused this error is as follows:

```
MESSAGE( G$_NLS.Get('X', 'FORM','*Error* - Go to key-block and provide
all data for the creation of the new record.'));
                RAISE FORM_TRIGGER_FAILURE;
```

Q: Why has rekey changed G$_NLS.Get to G$_NLS.FormatMsg, and why did it not use a valid message key with a sequence number?

This will only occur when standards are not followed. The rekey utility replaces *G$_NLS.Get* with the more efficient *G$_NLS.FormatMsg*, which does not try to do a database lookup that will never find a translation. If you want to avoid *FormatMsg*, the code can be fixed manually to use the variables directly. The message key is not needed in *FormatMsg* because there is no need to look up translations. To avoid confusion with existing message numbers, the *x* is used.

You can grep the log file for the following text:

```
"*CHANGE* G$_NLS.Get to G$_NLS.FormatMsg"
```

Q: Is it correct that the tool renumbers all the messages? Shouldn’t it just renumber the 'X' / new messages?

It is intentional behavior that all messages get a new message key. This was done to keep the processing straightforward and alleviate the need for multiple code scans or methods to sort out duplicate keys.

Q: What is fcmpenv.bat?

This sets up the forms environment for your product (primarily FORMS90_PATH) to avoid issues with missing objects from reference forms.

Q: Do I need to customize fcmpenv.bat?

Not if you are processing dbprocs only. If the Banner library forms that can be found via `fcmpenv.bat` contain all the referenced objects in your code, you don't need to customize the file. But you need to be aware of any errors encountered during processing. Processing forms with the wrong FORMS90_PATH may result in losing the object references in the form (the same as with the forms builder tool).

Q: The audit trails contain a line with odd hexadecimal numbers like “MSGSIGN : #0316eb8b744e93ae.” What is this?

The hexadecimal number is a digital signature (or hashcode) of all message keys and messages in the file. This number will change if the messages change, the purpose is to add the same signature in the .tmm's in the future to validate that they match.