

[print](#) | [close](#)

APIs by Example: Locales, APIs, and Time Zones

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 10/12/2006 (All day)

In previous articles, I covered the time zone support introduced in V5R3 and the APIs available to exploit this feature (links to these past articles are at the end of this article). However, the native i5/OS time zone support has not been fully implemented yet. Currently the job time zone attribute is merely copied from the time zone system value, so the system currently has no support for setting up different time zones for different users or subsystems.

However, i5/OS already supports locales. These provide an alternative to control time zone at job or user level, but this alternative involves some programming effort. Locales are the focus of this week's APIs by Example.

Locales are intended to enable an application for internationalization, also known as application globalization, in which you ensure that the application can be run independently of language, script, culture, and coded character set.

Locales and, more specifically, the POSIX locales are thoroughly introduced in the article "Hello World!: Globalization and POSIX Locales" (*System iNEWS*, February 1998, article ID 2599). The article also describes in detail the steps involved in creating and setting up locales and provides a comprehensive discussion of locale operands and configuration. Here is a link to the article:

<http://www.SystemiNetwork.com/article.cfm?id=2599>

Two flavors of locales exist on the System i: C and POSIX locales, object type *CLD and *LOCALE, respectively. Because the POSIX version is more versatile and more widely used, this article deals with the POSIX variant. The C compiler maps to the correct versions of the C locale functions based on a compiler directive, but for use in RPG IV, we simply specify the appropriate locale function with a `_C_PSX_` prefix, as in `_C_PSX_setlocale()`. More about this later.

i5/OS includes a number of system-supplied locales and sources. The locale sources are in the source file QLOCALESRC in library QSYSLOCALE. If you have no luck finding the QSYSLOCALE library, please verify that 5722-SS1 option 21 - Extended NLS Support is installed on your system. The Display Software Resources (DSPSFWRSC) command lets you check that easily. If option 21 is missing, look here for information about how to install and enable locales:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/nls/rbagsinstalllocales.htm>

As a matter of fact, i5/OS already lets you take advantage of locales when setting up user profiles to reflect the cultural and regional settings appropriate for the location of the individual user. The following article gives you all the details:

<http://www.SystemiNetwork.com/article.cfm?id=16812>

The IBM Info Center section titled "Work with locales" describes locale creation, locale programming, and the different locale categories, and it contains a lot of other information about

locales:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/nls/rbagslocale.htm>

To change a locale, you must copy the appropriate system locale source to a source file in a user library and perform the necessary changes to the copy. You can use the following command to create a locale source file in library QGPL:

```
CRTSRCPF FILE( QGPL/QLOCALESRC )
        RCDLEN( 112 )
        TEXT( 'Locale source file' )
```

Next, copy the locale source from QSYSLOCALE/QLOCALESRC to the newly created locale source file. After making the required modifications, use the Create Locale (CRTLOCALE) command to create the new locale:

```
CRTLOCALE LOCALE( '/QSYS.LIB/QGPL.LIB/EN_GB_285.LOCALE' )
          SRCFILE
( '/QSYS.LIB/QGPL.LIB/QLOCALESRC.FILE/EN_GB_285.MBR' )
          CCSID( 285 )
```

This command creates a locale named EN_GB_285 in library QGPL, based on the locale source member EN_GB_285 in file QLOCALESRC in library QGPL. The CRTLOCALE command's Coded Character Set Identifier (CCSID) parameter is particularly important. The locale source specifications defining character sets, collating sequence, and similar definitions are resolved to the actual characters and signs at compilation time. So the specified CCSID must support all the characters and signs defined in the locale source.

Because no command to display locale attributes exists, IBM simply specifies the CCSID used for the system-supplied locales in the locale object description. Running the command:

```
WRKOBJ *ALL/*ALL *LOCALE
```

displays a list of all currently available locales and their descriptions. The Info Center provides documentation about the system-supplied locales and their recommended CCSID for each release. Please follow the links at the end of this article to learn more.

The system-supplied locales' naming convention defines the locale name as LG_TT, where LG is a two-letter language abbreviation and TT is a two-letter territory abbreviation. Euro-enabled locales add an E to that naming scheme — LG_TT_E. For example, the U.S. system locale is named EN_US (English, USA), and the Euro-enabled locale for Great Britain is named EN_GB_E (English, Great Britain, Euro).

For user-created locales, adding the locale's CCSID as the final name extension is recommended. For example, a user-created version of the U.S. locale would be EN_US_37.

Some standard locales span more than one time zone. Because the API example in this article exploits the locale time zone support, I've extended the locale naming standard to include the time zone instead of the CCSID. For Mountain Standard Time in the U.S., for example, that would lead to a name of EN_US_MST. I then specify the CCSID in the locale object description, as does IBM for the system-supplied locales.

Locales define seven different categories of locale information:

LC_COLLATE	- Describes collating sequence for all characters provided by the CCSID used to create the locale
LC_CTYPE	- Defines locale character classes and mappings
LC_MESSAGES	- Defines the formats and values for affirmative and negative responses
LC_MONETARY	- Describes rules and symbols for the region's monetary numeric data
LC_NUMERIC	- Provides formats and symbols for nonmonetary numeric data
LC_TIME	- Defines date and time formats used by time-formatting functions
LC_TOD	- Extends the LC_TIME category, defining the time difference to Greenwich Mean Time (GMT) and the rules applying to Daylight Saving Time (DST)

For an in-depth description of these categories, check out the IBM documentation and POSIX locale article referenced earlier in this article.

One noticeable aspect of locales is their influence on the behavior of certain C/C++ runtime library functions, as documented by the *ILE C/C++ Programmer's Guide* — Chapter 19: International Locale Support:

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/books/co92712331.htm>

More specific documentation is in the Locale-Sensitive Run-Time Functions table:

http://publib.boulder.ibm.com/iserics/v5r2/ic2924/books/co92712331.htm#Header_490

The behavior and result of the functions in this table are controlled by the locales and categories currently set for the runtime executing these functions.

The LC_TOD locale category is an IBM extension to the LC_TIME category, which defines the rules and settings that apply to the locale's time zone and DST. The LC_TOD category includes the following operands:

tzdiff	- Time zone difference to Coordinated Universal Time (UTC) (GMT) in minutes
tname	- Time zone name (abbreviation)
dstname	- DST name (abbreviation)
dststart	- Start date for DST
dstend	- End date for DST
dstshift	- DST shift time in seconds.

The system-supplied locales, however, specify no values for this category's operands. So to activate the LC_TOD category, some research about locale programming is required. Using the Google search engine, I easily found a number of references to the LC_TOD category on different IBM sites, but unfortunately none of the different documents agreed about how to specify the various operands! Applying the trial-and-error method, I arrived at the conclusion that follows. My investigation most likely did not cover all aspects of this issue, though, so if you find reason to disagree, please let me know. Here's what I've come up with so far:

```

tzdiff    - Time zone difference to UTC in minutes:
            A positive number indicates a location west of GMT
            (e.g., the U.S.)

            A negative number indicates a location east of GMT
            (e.g., Italy, Denmark, Russia)

tname     - Time zone name:
            The abbreviated form (e.g., EST, MST, CET, AEST)
            (Run the WRKTIMZON command to verify)

dstname   - Daylight Saving Time:
            The abbreviated form (e.g., EDT, MDT, CEST, AEDT)
            (Run the WRKTIMZON command to verify)

dststart  - Start date for DST:
            Four integers specifying the month, week, day, and time
            that DST starts:
            Month: 1-12,
                    1=January, 12=December

            Week:  0, 1 to 4, -1 to -4
                    0=The Day field represents the day of the month
rather
                    than the day of the week.
                    1=First week of month
                    4=Fourth week of month
                    -1=Last week of month
                    -2=Second to last week of month
                    -4=Fourth to last week of month

            Day:   1-7, 1-31
                    1=Monday, 7=Sunday (when Week field is not 0)
                    1-31=Day of month (when Week field is 0)

            Time:  0-86399 seconds
                    The number of seconds after midnight
                    the shift takes place

dstend    - End date for DST:
            Four integers specifying the month, week, day, time
            that DST ends. Same rules as for dststart.

dstshift  - DST shift time in seconds:
            An integer value specifying the DST
            shift in seconds.

```

Please note that in the southern hemisphere, DST is observed (where applicable) at the opposite time of year than in the northern hemisphere. When winter rules in the U.S., it's summertime in Australia, and vice versa.

To help clarify the preceding rule set, and based on the values that the WRKTIMZON command currently provides for the different time zones, I created some example LC_TOD's for the following time zone locales:

```
EN_AU_AEST (Australian Eastern Standard Time):
```

```
LC_TOD
```

```
tzdiff    -600
```

```
tname     ""
```

```
dstname   ""
```

```
dststart  10,-1,7,7200
```

```
dstend     4,-1,7,7200
```

```
dstshift  3600
```

```
END LC_TOD
```

```
DA_DK_277 (Central European Time - Denmark):
```

```
LC_TOD
```

```
tzdiff    -60
```

```
tname     ""
```

```
dstname   ""
```

```
dststart  3,-1,7,7200
```

```
dstend    10,-1,7,10800
```

```
dstshift  3600
```

```
END LC_TOD
```

```
EN_GB_285 (Greenwich Mean Time - Great Britain):
```

```
LC_TOD
```

```
tzdiff    0
```

```
tname     ""
```

```
dstname   ""
```

```
dststart  4,-1,7,3600
```

```
dstend    10,-1,7,7200
```

```
dstshift  3600
```

```
END LC_TOD
```

```
EN_US_EST (Eastern Standard Time):
```

```
LC_TOD
```

```
tzdiff    300
```

```
tname     ""
```

```
dstname   ""
```

```
dststart  4,1,7,7200
```

```
dstend    10,-1,7,7200
```

```
dstshift  3600
```

```

END LC_TOD

EN_US_CST (Central Standard Time):

LC_TOD
tzdiff 360
tname ""
dstname ""
dststart 4,1,7,7200
dstend 10,1,7,7200
dstshift 3600
END LC_TOD

EN_US_MST (Mountain Standard Time):

LC_TOD
tzdiff 420
tname ""
dstname ""
dststart 4,1,7,7200
dstend 10,1,7,7200
dstshift 3600
END LC_TOD

EN_US_PST (Pacific Standard Time):

LC_TOD
tzdiff 480
tname ""

dstname ""

dststart 4,1,7,7200
dstend 10,1,7,7200
dstshift 3600
END LC_TOD

```

As I mentioned earlier, some confusion exists about the correct specification of the various operands, so be sure to test and verify thoroughly before implementing any of these LC_TODs in a production environment.

To enable you to verify your success as a locale programmer, I wrote the Display Locale Time of Day (DSPLOCTOD) command. The DSPLOCTOD command's CPP also serves as an example of how to use the C/C++ runtime library functions that make calculating a local timestamp based on a locale possible—provided that the locale has a valid LC_TOD category specified.

Here's the DSPLOCTOD command prompt:

```


```

```

-----Display Locale Time of Day (DSPLOCTOD)-----
Type choices, press Enter.
Locale . . . . . Name, *USRPRF
Library . . . . . *LIBL Name, *LIBL, *CURLIB
User profile . . . . . Character value

```

You can specify either a qualified locale object name or the name of a user profile that has a specific, named locale defined. That locale is then retrieved from the user profile. In both events, the locale in question is used to calculate a local timestamp, based on the time zone and DST information in the locale. The locale name and the local timestamp are then displayed in a window on the screen:

```

.....
+:
+:
+: Locale object . . . . : /QSYS.LIB/QGPL.LIB/EN_US_MST.LOCALE +:
+: Current local time . : 2006-10-07-11.55.15.060688 +:
+:
+:
+:
+:
+:
+: Bottom +:
+: F12=Cancel +:
+:
+:
+: .....

```

The command's CPP performs the following actions:

1. If *USRPRF is specified for the locale parameter, the locale path name is retrieved from the user profile specified as the second parameter.
2. The locale is set using the `_C_PSX_setlocale` function. This returns a pointer to the previous locale in effect, and this locale name is resolved and saved to be restored later.
3. The current UTC (GMT) timestamp is retrieved using the `gettimeofday()` API. This function returns two data structure parameters named `timeval` and `timezone`. The `timeval` structure contains an epoch-1970 value (seconds and microseconds since 1 January 1970, 00:00:00 UTC) and is used as input for the next step.
4. The `_C_PSX_localtime` function (which is locale sensitive) takes the seconds part of the `timeval` data structure as input and returns the local time in a `tm` (short for "time") data structure. The `tm` structure is then used to calculate the actual timestamp, and the microsecond part of the `timeval` structure is added to arrive at an exact offset of the initial UTC timestamp.
5. The locale name and calculated timestamp is displayed in a window for verification.
6. Finally, the initially retrieved previous locale is restored.

You assign a locale to a user profile using the Change User Profile (CHGUSRPRF) command. The locale must be specified in path name format:

```
CHGUSRPRF USRPRF( JAN )
        LOCALE( '/QSYS.LIB/QGPL.LIB/DA_DK_277.LOCALE' )
```

After the preceding command is run, any job started using the specified user profile has the DA_DK_277 locale set, and the job's LANG environment variable specifies that same locale as well. This behavior applies to both batch and interactive jobs.

If you're looking for more information and documentation about time zones, daylight saving time, and the like, the following site is a great place to start your search:

<http://www.greenwichmeantime.com/>

This APIs by Example includes the following sources:

```
CBX163      Display Locale Time of Day  CPP
CBX163B    -- Display Locale Time of Day - Binder source
CBX163H    -- Display Locale Time of Day - Help
CBX163S    -- Display Locale Time of Day - Services
CBX163V    -- Display Locale Time of Day - VCP
CBX163X    -- Display Locale Time of Day
CBX163M    -- Display Locale Time of Day - Build command
```

To create all these objects, compile and run CBX163M. Compilation instructions are in the source headers, as usual.

The API date and time zone support article is here:

<http://www.SystemiNetwork.com/article.cfm?id=51703>

Parts one, two, and three of the related User Application Information APIs article are here:

<http://www.SystemiNetwork.com/article.cfm?id=52288>

<http://www.SystemiNetwork.com/article.cfm?id=52387>

<http://www.SystemiNetwork.com/article.cfm?id=52457>

System-supplied locales and recommended CCSIDs V5R3:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/nls/rbagssysuplocalesourcedef.htm>

System-supplied locales and recommended CCSIDs V5R4:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/nls/rbagssysuplocalesourcedef.htm>

This article demonstrates the following APIs:

Get Current UTC Time (gettimeofday()) function:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/gettod.htm>

Set Locale (setlocale()) function (page 322 of *ILE C/C++ Run-Time Library Functions, SC41-5607-02*):

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/books/sc415607.pdf#SETLOC>

~~Convert Time (localtime()) function (page 173 of *ILE C/C++ Run-Time Library Functions, SC41-5607-02*):~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/books/sc415607.pdf#LOCALT>~~

~~Retrieve User Information (QSYRUSRI) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qsyrusri.htm>~~

~~Retrieve Job Information (QUSRJOBI) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusrjobi.htm>~~

~~Convert a Graphic Character String (QTQCVRT) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/CDRCVRT.htm>~~

~~Retrieve Message (QMHRTVM) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHRTVM.htm>~~

~~Send Program Message (QMHSNDPM) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHSNDPM.htm>~~

~~Retrieve Object Description (QUSROBJD) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusrobid.htm>~~

~~Display Long Text (QUILNGTX) API:~~

~~<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/quilngtx.htm>~~

~~You can retrieve the source code for this API example from~~

~~http://www.pentontech.com/IBMContent/Documents/article/53355_124_LocaleTimeZone.zip.~~

~~**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-locales-apis-and-time-zones>~~