BlackBerry Software Development Kit

Version 2.5

Ribbon and Options API Reference Guide

BlackBerry Software Development Kit Version 2.5 Ribbon and Options API Reference Guide
Last revised: 25 June 2002

Part number: PDF-04640-001

At the time of publication, this documentation complies with RIM Wireless Handheld version 2.5.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Published in Canada

Contents

About this guide **5**

Related documentation 5

CHAPTER 1 **Ribbon API functions** **7**

Ribbon 7

Functions 7

CHAPTER 2 **Options API functions** **17**

Options 17

Structures 17

Functions 18

Index of functions **33**

Index **35**

About this guide

The Ribbon and Options application programming interface (API) contains functions for modifying the handheld's Ribbon and Device Options screens and attributes. The Ribbon is the RIM Wireless Handheld's Graphical User Interface (GUI). It can contain links to applications, tools, and web pages.

Related documentation

Before you use this guide, you should be familiar with the following documentation. These other resources can help you develop C++ applications for the BlackBerry Wireless Handheld.

- *BlackBerry SDK Developer Guide*

  The *BlackBerry SDK Developer Guide* explains how to use the BlackBerry SDK, with tutorials and sample code to demonstrate how to write handheld applications. For additional information, visit the BlackBerry Developer Zone at `http://www.blackberry.net/developers`.

- *UI Engine API Reference Guide*

  This guide includes a complete listing of all structures and functions used by the UI Engine.

- `README.txt`

  The `README.txt` file is installed with the BlackBerry Software Development Kit (SDK). It provides information on any known issues and workarounds, as well as last-minute documentation updates and release notes.

Chapter 1
Ribbon API functions

Ribbon

The Ribbon displays all registered applications along with bitmaps in a functions list on the Home screen. The Ribbon is the base of the RIM Wireless Handheld user interface; in most cases, to make an application available to a user, you must place it on the ribbon.

Functions

The following functions are listed alphabetically.

RibbonAddBitmap 7
RibbonAddBitmapToElement 8
RibbonAddElement 8
RibbonGetPrevApplication 9
RibbonModifyApplication 9
RibbonRegisterApplication 10
RibbonRegisterFunction 11
RibbonRemoveBitmap 12
RibbonSetApplicationString 13
RibbonSetBitmap 14
RibbonSetCursorPosition 14
RibbonSetNextApplication 15
RibbonSetPrevApplication 15
RibbonShowRibbon 15
RibbonUnregisterApplication 15

RibbonAddBitmap

Associates ribbon bitmaps (icons) with an application.

```
const int RibbonAddBitmap(
    const char * applicationName,
    const BitMap * bitmap)
```

Parameters

`applicationName`	The name of the application to associate with the bitmap.
`Bitmap`	The bitmap resource.

Returns The index into the bitmap array in which the bitmap is placed. If the bitmap array for the application is full, the return value is –1.

RibbonAddBitmapToElement

Adds a bitmap to an element.

```
const int RibbonAddBitmapToElement(
    const char * applicationName,
    const BitMap * bitmap)
```

Parameters

`applicationName`	The name of the application to associate with this element.
`bitmap`	A pointer to the bitmap for this element.

RibbonAddElement

Adds an element to the ribbon.

```
void RibbonAddElement(const BitMap * const bitmapPtr,
    int applicationData,
    int position,
    void (*entry)(void) = NULL,
    DWORD stacksize = 0,
    TASK task = (TASK)-1)
```

Parameters

`bitmapPtr`	Pointer to the bitmap for this application.
`applicationData`	When the ribbon sends a message to the registering application, the information in this parameter is included in the `SubMsg` field of the `message` data structure. The same application can then register multiple icons, if required.
`position`	The position of the new element on the ribbon. In the case of applications that are set the the same cursor position, the application that called `RibbonSetCursorPosition` prevails. A lower value of position means that the application is placed to the left. Positions 1 to 20 are reserved.

Parameters

`bitmapPtr`	Pointer to the bitmap for this application.
`(*entry)(void)`	Pointer to an entry function for the application.
`stacksize`	The stacksize required by the application.
`task`	The application that is currently running.

RibbonGetPrevApplication

Retrieves the name of the application that had focus previous to the current application.

```
const char * RibbonGetPrevApplication()
```

Returns The name of the application that was previously in focus.

Description An application relinquishing focus should call this function to determine which application should receive focus.

RibbonModifyApplication

Enables an application to change its appearance in the ribbon by updating its bitmap, changing the application name, or both.

```
void RibbonModifyApplication(
    const char * applicationName,
    const BitMap * bitmap,
    const char * const newApplicationName)
```

Parameters

`applicationName`	This parameter is the name of the application to modify. (The string found here is the same as that found in `applicationName` of `RibbonRegisterApplication`.) Only the thread that has registered its name can modify the application. **Note:** If you are calling this function, make sure that the string from the `applicationName` parameter of `RibbonRegisterApplication` has any hot key letters removed. These identifiers are denoted by an ampersand (&)
`bitmap`	This parameter is a pointer to the bitmap that will replace the currently displayed bitmap. The maximum bitmap size is 16 by 16 pixels. This parameter can be NULL.
`newApplicationName`	This parameter is the new name of the application. A hot key identifier can be used in this string. This parameter can be NULL.

Returns `RibbonModifyApplication` has no return value. It always succeeds.

Description RibbonModifyApplication enables changes to the application appearance on the ribbon. The radio control icon uses this functionality to change its appearance according to the current state of the radio.

Icon manipulation in RibbonModifyApplication will eventually be removed. Only use RibbonModifyApplication to modify the name and hot key. To change, add, or delete bitmaps, use the functions RibbonSetBitmap, RibbonAddBitmap and RibbonRemoveBitmap.

Currently RibbonModifyApplication does not return a value that indicates where the new icon was placed. If a new icon is passed in using RibbonModifyApplication, then the function removes the current icon (if any), adds the new icon, and sets it. The position of the new icon can be returned (it is guaranteed to always have a position because the old one is removed) at your request.

RibbonRegisterApplication

Retrieves a bitmap and name so that an application can be displayed in the ribbon.

```
void RibbonRegisterApplication(
    const char * applicationName,
    const BitMap * bitmap,
    int applicationData,
    int position)
```

Parameters

applicationName	This string uniquely identifies the application, along with the task ID of the calling thread. If hot key access to the application is required, an ampersand (&) can be used to identify the character that should act as a hot key. If more than one application registers with the same hot key, the application with the lowest position starts.

`bitmap`	A pointer to the bitmap to be displayed in the ribbon. The maximum bitmap size is 16 by 16 pixels. If this parameter is NULL, then the bitmap stored in the application PID is used. If the application has not registered a PID, then a portion of the ribbon display is used (results are undefined). Refer to the *OS API Reference Guide* for more information on PID.
`applicationData`	When the ribbon sends a message to the registering application, the information in this parameter is included in the `SubMsg` field of the `message` data structure. The same application can then register multiple icons, if required.
`position`	This parameter determines the application's position on the ribbon. In the case of applications that are set to the same cursor position, the application that called `RibbonSetCursorPosition` prevails. A lower `position` value means that the application is placed to the left. Positions 1 to 20 are reserved.

Returns `RibbonRegisterApplication` has no return value. It always succeeds.

Description To be displayed in the Ribbon, an application must call this function and provide a bitmap and name. If a bitmap and a name are not provided, they can be taken from the application PID.

RibbonRegisterFunction

Retrieves a facility for dynamically launching a new application when the ribbon icon is clicked.

```
void RibbonRegisterFunction(
    const char * applicationName,
    const BitMap * bitmapPtr,
    int applicationData,
    int position,
    void (*entry)(void),
    DWORD stacksize)
```

Parameters

`applicationName`	This string (with the task ID of the calling thread) uniquely identifies the application. If hot key access to the application is required, an ampersand (&) can be used to identify the character that should act as a hot key. If more than one application registers with the same hot key, the application with the lowest `position` starts.
`bitmapPtr`	This parameter is a pointer to the bitmap that is to be displayed in the ribbon. The maximum bitmap size is 16 by 16 pixels. If this parameter is NULL, then the bitmap that is stored in the application PID is used. If the application has not registered a PID, then a portion of the ribbon display is used (results are undefined). Refer to the *OS API Reference Guide* for more information on PID.
`applicationData`	When the ribbon sends a message to the registering application, the information in this parameter is included in the `SubMsg` field of the `message` data structure. The same application can then register multiple icons, if required.
`position`	The application's position on the ribbon. In the case of applications that are set at the same cursor position, the application that called `RibbonSetCursorPosition` prevails. A lower value for `position` means that the application is placed to the left. Positions 1 to 20 are reserved.
`(*entry)(void)`	A pointer to an entry function for the application.
`stacksize`	The stacksize required by the application.

Returns `RibbonRegisterFunction` has no return value. It always succeeds.

Description This method dynamically launches the application when the ribbon icon is clicked. This new form permits better memory management because the application is not in memory unless activated by the ribbon.

This function is recommended because it increases the memory resources that are available to applications by unloading an application after it is no longer in focus. Use caution, because after the entry point function returns, any stack memory is freed and associated data lost.

RibbonRemoveBitmap

Removes a bitmap from the application bitmap array.

```
bool RibbonRemoveBitmap(
    const char * applicationName,
    const int bitmapPosition)
```

Parameters

`applicationName`	The name of the application.
`bitmapPosition`	An index into the application bitmap array. The bitmap at the given index is removed. If equal to –1, all bitmaps in the array are used.

Returns True if the bitmap is found and freed; false otherwise.

RibbonSetApplicationString

Display a short string if the application needs user interaction or intervention.

```
void RibbonSetApplicationString(
    const char * const appName,
    const char * const string,
    const BitMap * const bitMapPtr,
    int priority)
```

Parameters

`appName`	This parameter identifies the application. (The string found here is the same as that found in `applicationName` of `RibbonRegisterApplication`.) Only the thread that has registered its name is allowed to modify the application. **Note:** If you are calling this function, make sure that the string from `appName` parameter of `RibbonRegisterApplication` has hot key identifiers removed. These identifiers are denoted by an ampersand (&).
`string`	A pointer to a string to be displayed on the Home screen and the Security screen. A maximum of five characters of the string are copied into an internal buffer for display. Setting this string to NULL clears the application display and no messages will be sent.
`bitMapPtr`	A pointer to a bitmap that is the size of a character. If this points to a bitmap (that is, is not NULL), the bitmap is displayed instead of one of the characters in `string`, limiting `string` to four characters. This bitmap is a useful way to display an image that is not in the standard font set.
`priority`	This parameter determines the relative notification priority. When the handheld is removed from the holster, a `ribbon_holster_launch` message is sent to the application with the highest priority that has called `RibbonSetApplicationString()`.

Description Using `RibbonSetApplicationString` before calling `Options::NotifyUser` enables the ribbon and options to launch an application as soon as the handheld is removed from the holster. If the password is set, it must be entered before the `ribbon_holster_launch` message is sent.

The application string is displayed in the user alert section of the Home screen. Applications that need user interaction or intervention can call this function to display a short string (such as displaying new messages or new alarms on the Ribbon). Only the highest two priority strings are displayed. If an alert is triggered and then the handheld is removed from the holster, the highest priority application is launched.

RibbonSetBitmap

Chooses a ribbon bitmap (icon) to display for an application.

```
const int RibbonSetBitmap(
    const char * applicationName,
    const int bitmapPosition)
```

Parameters

`ApplicationName`	The name of the application.
`bitmapPosition`	An index into the bitmap array. If equal to –1, the bitmap that is used is the biggest bitmap in the array that fits into the area that is allocated for the device ribbon icons. If equal to 0, a dummy icon is used (it resembles the Folder icon). Otherwise, the bitmap at the position that is noted is used as the ribbon icon.

Returns The index of the bitmap used.

Description Use this function to select the ribbon bitmap to display for an application. Passing –1 as the `bitmapPosition` parameter uses the bitmap that best fits the area allocated for ribbon bitmaps on the handheld.

RibbonSetCursorPosition

Sets the cursor position.

```
void RibbonSetCursorPosition(int position)
```

Parameters

`position`	The position in which to place the cursor. In the case of applications that are set at the same cursor position, the application which called `RibbonSetCursorPosition` prevails. A lower value of position means that the application is placed to the left. Positions 1 to 20 are reserved.

RibbonSetNextApplication

Specifies the next application to launch after the ribbon returns to the foreground.

```
void RibbonSetNextApplication(const char * const nextAppName)
```

Parameters

`nextAppName`	This parameter is the next application to run after the current application returns. This string is the same as that found in the `applicationName` parameter of `RibbonRegisterApplication`.) Only the thread that has registered its name is allowed to modify the application. **Note:** If you are calling this function, make sure that the string from `applicationName` parameter of `RibbonRegisterApplication` has any hot key identifiers removed. These identifiers are denoted by an ampersand (&).

Description If this function is called, the ribbon launches the application that is named in `nextApplicationName` instead of returning to the foreground. The next application information is only used once, and `RibbonSetNextApplication` must be called again to repeat the action.

If a bitmap and a name are not provided, they can be taken from the application PID.

RibbonSetPrevApplication

Specifies the application name string that is returned by a call to `RibbonGetPrevApplication`.

```
void RibbonSetPrevApplication(const char * const prevAppName)
```

Parameters

`PrevAppName`	The name of the application to set as the previous application.

Description Because of dynamic application launching, the application that is losing focus might want to call this method with its own application name as the parameter. By specifying the parameter in this way, the application that is gaining focus can determine to which application to return the focus.

RibbonShowRibbon

Puts the ribbon in the foreground after an application is complete.

```
void RibbonShowRibbon()
```

Description The foreground application should call this function when it is ready to return to the Home screen.

RibbonUnregisterApplication

Removes the calling application from the Home screen.

```
void RibbonUnregisterApplication(const char * applicationName)
```

Parameters

`applicationName`	Name of the application to remove from the ribbon. **Note:** If you are calling this function, make sure that the string from the `applicationName` parameter of `RibbonRegisterApplication` has any hot key identifiers removed. These identifiers are denoted by an ampersand (&).

Description Applications that are about to terminate should call `RibbonUnregisterApplication` to make sure that they are removed from the ribbon before exiting.

Chapter 2
Options API functions

The Options API contains functions that enable you to manage new and existing entries on the Device Options menu on the handheld, as well as the internal processes of these menu items.

Options

To include the Options API functions in your programs, you must include `<options.h>` in your code.

Structures

The Device Options API uses the following structures.

TZDayInfo 17
TZInfo 18

TZDayInfo

`TZDayInfo` is the structure that is used to describe the start or end date of a shift to Daylight Savings Time.

```
struct TZDayInfo {
    WORD month : 4;
    WORD positionOfWeek : 3;
    WORD day : 5;
    WORD hour : 4;
};
```

Field	Description
`month`	month of shift (1 to 12); o = invalid/absent date
`positionOfWeek`	0 for set day of month, otherwise 1 to 5 for first to last week of month
`day`	day of week (0 to 6) if `positionOfWeek` > 0, day of month (1 to 31) otherwise
`hour`	hour of day (0 to 23)

TZInfo

`TZInfo` is the structure that is used to describe a time zone.

```
structure TZInfo {
    unsigned short id;
    signed char bias;
    TZDayInfo standardDate;
    TZDayInfo daylightDate;
};
```

FIeld	Description
`id`	unique identifier for time zone; identifiers are not necessarily consecutive
`bias`	standard bias (in 15 minute units; for example, `offset_from_UTC * 60/15 EST = -5 * 60/15`)
`standardDate`	date when standard time starts (`month` = 0 if no DST)
`daylightDate`	date when daylight time starts (`month` = 0 if no DST)

Functions

The following functions are listed alphabetically.

AddOption 19
ConvertTime 20
DisplayAlarmFunction 20
EnterSecurityPassword 20
FormatDate 21
FormatDay 22
FormatDayDate 22
FormatTime 22
GetOwnerInformation 23

GetOwnerName 23
GetPassword 24
NotifyUser 24
OptionsEntry 24
OptionsGetCurrentTimeZone 25
OptionsGetCurrentTimeZoneId 25
OptionsGetCurrentTimeZoneInfo 25
OptionsGetDSTData 25
OptionsGetIndexFromTimeZoneId 26
OptionsGetTimeZoneDescriptions 26
OptionsGetTimeZoneIdFromIndex 26
OptionsGetTimeZoneInfo 27
OptionsIsDateInDST 27
OptionsIsDaylightSavingsActive 27
OptionsIsDaylightSavingsEnabled 27
OptionsRegisterTimeChangeCB 28
OptionsSetCurrentTimeZone 28
OptionsSetCurrentTimeZoneId 28
OptionsSetDateTime 28
OptionsSetDateTimeDST 29
OptionsSetDaylightEnabled 29
OptionsSetDaylightSavings 29
RegisterApplication 29
RegisterNotifyChoice 30
RegisterOptions 30
RegisterPasswordChange 30
RegisterSecurity 31
RemoveOption 31

AddOption

Adds an option to the options list.

```
char const * AddOption(
    void(*func)(void),
    char const * name,
    void(*entry)(DWORD) = NULL)
```

Parameters

`func`	This callback function is executed when the user selects `name` from the options list.
`name`	Specifies the name of the option, displays in the Device Options list.
`(*entry)(DWORD)`	A pointer to an entry function for the option.

Returns The `optionDLL`'s title, or NULL if the option failed to register.

Description If your application needs to display or configure its own options as part of the handheld options set, you can call this function add your application to the registration list. The `name you` supply is displayed in the Device Options list. When a user selects that option, the function is called. For more information, refer to `RemoveOption`.

ConvertTime

Converts time to Daylight Savings Time.

```
bool OptionsConvertTime(TIME & t,
    unsigned short tzFrom,
    unsigned short tzTo)
```

Parameters

`t`	A pointer to the time to convert.
`tzFrom`	The time zone of `t`.
`tzTo`	The time zone to which to convert `t`.

Returns The converted time.

DisplayAlarmFunction

This function is only included for binary comapatibility. You should *not* call this function.

```
void DisplayAlarmFunction(int index)
```

EnterSecurityPassword

Launches the password UI dialog.

```
int EnterSecurityPassword(
    UIEngine & ui,
    Screen & screen,
    int anchor,
    void (*callback)(MESSAGE & msg) = NULL,
    int one_try = 0,
    int allow_exit = 1)
```

Parameters

`ui`	A reference to an initialized UI Engine.
`screen`	A reference to a screen UI component.
`anchor`	The dialog anchor point on the y-axis, in pixels.

`(*callback)(MESSAGE & msg)`	A pointer to a callback function that the password process calls if it receives a message while active that it cannot handle.
`one_try`	Indicates whether the password dialog box should stay active after a failed attempt. If set to true, the dialog box exits if the user enters an incorrect password on the first try. The failed attempts counter is not increased.
`allow_exit`	If set to 0, the password dialog box does not exit until the user has entered the correct password. Full security measures are in place in this case, that is, if the user fails 10 times, critical resources are erased to avoid compromising data.

Returns An integer corresponding to one of the following fields.

Field	Value
PASSWORD_USER_OK	1
PASSWORD_FAILED	0
PASSWORD_ABORTED	-1
PASSWORD_ERASED	-2

FormatDate

Formats the date passed in the `TIME` structure according to the current user settings and puts the result into the supplied buffer.

```
void FormatDate(
    TIME * date,
    char * buffer,
    int form)
```

Parameters

`date`	A pointer to the date.
`buffer`	Specifies the buffer in which to store the formatted date.
`form`	The form, either `FORMAT_SHORT` (for example, “2001/01/31”) or `FORMAT_LONG` (for example, “January 31, 2001”).

Description The date and time require a 20-byte buffer length and `form` is long or short form. If `time` is NULL, `FormatDate` returns the current system time and date.

FormatDay

Formats the date passed in the `TIME` structure according to the current user settings and puts the result into the supplied buffer.

```
void FormatDay(
    TIME * time,
    char * buffer,
    int form)
```

Parameters

`time`	A pointer to the time.
`buffer`	Specifies the buffer in which to store the formatted day.
`form`	The form, either `FORMAT_SHORT` (for example, “Thu”) or `FORMAT_LONG` (for example, “Thursday”).

Description The date and time require a 20-byte buffer length and `form` is long or short form. If `time` is NULL, `FormatDay` returns the current system date as a day of the week.

FormatDayDate

Formats the day and date passed in the `TIME` structure according to the current user settings and puts the result into the supplied buffer.

```
void FormatDayDate(TIME * date,
    char * buffer,
    int form)
```

Parameters

`date`	A pointer to the day and date.
`buffer`	Specifies the buffer in which to store the formatted day and date.
`form`	The form, either `FORMAT_SHORT` (for example, “THU, JAN 24”) or `FORMAT_LONG` (for example, “Thursday, January 24th”).

Description The day and date require a 20-byte buffer length.

FormatTime

Formats the time passed in the `TIME` structure according to the current user settings and puts the result into the supplied buffer.

```
void FormatTime(
    TIME * time,
    char * buffer,
    int form)
```

Parameters

`time`	A pointer to the time.
`buffer`	Specifies the buffer in which to store the formatted time.
`form`	The form, either `FORMAT_SHORT` (for example, "2:14p") or `FORMAT_LONG` (for example, "2:14 PM").

Description The date and time require a 20-byte buffer length and `form` is long or short form. If `time` is NULL, `FormatTime` returns the current system time.

GetOwnerInformation

Retrieves the owner information for the handheld.

```
int GetOwnerInformation(
    char * buffer,
    int len)
```

Parameters

`buffer`	A pointer to a buffer to which details from the Owner screen are copied.
`len`	The number of bytes, including the terminating NULL, that can be copied into the buffer.

Returns The actual length of the `OwnerInformation` string, including the NULL.

Description `GetOwnerInformation` copies the current owner information (address, and so on) from the Owner screen into the supplied buffer.

GetOwnerName

Retrieves the owner name for the handheld.

```
int GetOwnerName(
    char * buffer,
    int len)
```

Parameters

`buffer`	A pointer to a buffer to which details from the Owner screen are copied.
`len`	The number of bytes, including the terminating NULL, that can be copied into the buffer.

Returns The actual length of the `OwnerName` string, including the NULL.

Description `GetOwnerName` copies the current owner name from the Owner screen into the supplied buffer.

GetPassword

Retrieves a hash of the handheld password.

```
int GetPassword(
    unsigned char * hash,
    int * length)
```

Parameters

`hash`	A pointer to the hashed password.
`length`	A pointer to the length of the password hash.

Returns A flag indicating the status of the password: 0 indicates the password is disabled, 1 indicates the password is enabled.

NotifyUser

Notifies the user, using a tune, that an event has occurred.

```
void NotifyUser(
    int tune = 0,
    int repetitions = -1,
    char const * const name = NULL)
```

Parameters

`tune`	Tune to play to notify the user. Select from tunes 1 to 6. If set to 0, the default tune is used, as specified in the notify settings.
`repetitions`	The Number of times to play the tune, if set to –1, the default number of repetitions is used as specified by the system.
`name`	The name of the application to which to give focus.

Description Call this function when the application needs to signal the user that an event has occurred. If `tune` is set to 0, the configured tune is used. This function also gives focus to the specified application when the handheld is removed from the holster.

OptionsEntry

Adds an options to the options list. Deprecated – use `AddOption.`

```
char * OptionsEntry(
    void (*func)(void),
    char * name)
```

Parameters

`(*func)(void)`	This callback function is executed when the user selects `name` from the options list.
`name`	Specifies the name of the option, displays in the Device Options list.

OptionsGetCurrentTimeZone

Retrieves the current time zone in character format. Deprecated - use `OptionsGetCurrentTimeZoneInfo`.

```
int OptionsGetCurrentTimeZone()
```

Returns `OptionsGetCurrentTimeZone` returns the current time zone in character format. Acceptable values of time zones range from 120 (GMT +12 hours) to –120 (GMT –12 hours). For example, GMT -3.5 hours is equal to -35. Eastern Standard Time (GMT –5 hours) is represented as –50.

OptionsGetCurrentTimeZoneId

Retrieves unique time zone identifier.

```
unsigned short OptionsGetCurrentTimeZoneId()
```

Returns The unique identifier for the current time zone.

Description Each time zone is identified by a unique ID.

OptionsGetCurrentTimeZoneInfo

Retrieves time zone information.

```
bool OptionsGetCurrentTimeZoneInfo(TZInfo * pInfo)
```

Parameters

`pInfo`	Pointer to the time zone information for the current time zone.

Returns Time zone information for the current time zone.

OptionsGetDSTData

Retrieves Daylight Savings Time starting and ending times.

```
bool OptionsGetDSTData(unsigned short tzID,
    unsigned int year,
    TIME * tStandard,
    TIME * tDaylight)
```

Parameters

`tzID`	The unique identifier of the time zone.
`year`	The year for which to retrieve DST starting and ending times.
`tStandard`	A pointer to the time in Standard time.
`tDaylight`	A pointer to the time in Daylight Savings Time.

Description `OptionsGetDSTData` retrieves the DST start and ending times for a particular time zone in a particular year.

This function replaces the deprecated `OptionsIsDaylightSavingsEnabled`.

OptionsGetIndexFromTimeZoneId

Retrieves the position of a time zone in the list of time zone descriptions.

```
int OptionsGetIndexFromTimeZoneId(unsigned short id)
```

Parameters

`id`	The unique identifier of the time zone for which to retrieve the index position.

Returns The position of the time zone in the list of time zone description.

OptionsGetTimeZoneDescriptions

Retrieves a list of time zone descriptions.

```
const char ** OptionsGetTimeZoneDescriptions()
```

Description The time zone descriptions are stored internally on the handheld. They cannot be modified.

Returns The list of time zone descriptions.

OptionsGetTimeZoneIdFromIndex

Retreives the unique identifier for a time zone from the time zone list.

```
unsigned short OptionsGetTimeZoneIdFromIndex(int index)
```

Parameters

`index`	The time zone position in the list for which to retrieve the unique identifier.

Returns The unique identifier of the desired time zone.

OptionsGetTimeZoneInfo

Retrieves the time zone information for a time zone.

```
OptionsGetTimeZoneInfo(unsigned short tzID, TZInfo * pInfo)
```

Parameters

`tzID`	The unique identifier of the time zone for which to retrieve information.
`pInfo`	Pointer to the time zone information to retrieve.

Description Call `OptionsGetTimeZoneInfo` to retrieve time zone information for an arbitrary time zone. To retrieve the information for the current time zone, call `OptionsGetCurrentTimeZoneInfo`.

OptionsIsDateInDST

Determines if a date is running in Daylight Savings Time mode for a time zone.

```
bool OptionsIsDateInDST(unsigned short tzID, const TIME * t)
```

Parameters

`tzID`	The unique identifier of the time zone.
`t`	Pointer to the date to query.

Returns True if the date is running in Daylight Savings Time mode; false otherwise.

OptionsIsDaylightSavingsActive

Determines if Daylight Savings Time mode is running.

```
bool OptionsIsDaylightSavingsActive()
```

Returns 1 if Daylight Savings Time mode has been enabled on the handheld and is currently active; 0 otherwise.

Description `OptionsIsDaylightSavingsActive` determines if the handheld is currently in Daylight Savings Time mode (that is, between the first Sunday of April until the last Sunday of October, the handheld is one hour ahead).

OptionsIsDaylightSavingsEnabled

Determines if the handheld is in a time zone where Daylight Savings Time is allowed. Deprecated - use `OptionsGetDSTData`.

```
bool OptionsIsDaylightSavingsEnabled()
```

Returns 1 if Daylight Savings Time mode has been enabled on the device; 0 otherwise.

OptionsRegisterTimeChangeCB

Registers for callback when the system time changes.

```
bool OptionsRegisterTimeChangeCB(
    void (*callback)(int days, int secs))
```

Parameters `(*callback)(int days, int secs)` A pointer to a void function that accepts two integer parameters: the days delta and the seconds during the current day. (For example, a time change that results in moving the date back 5 days and 30 seconds corresponds to `days = -5` and `secs = -30`)

Returns True if the registration was successful; false otherwise.

OptionsSetCurrentTimeZone

Sets the current time zone. Deprecated - use `OptionsSetCurrentTimeZoneId`.

```
bool OptionsSetCurrentTimeZone(char time_zone)
```

Parameters `time_zone` The time zone that the handheld is to use. `time_zone` ranges from 120 (GMT +12 hours) to -120 (GMT –12 hours), in half-hour increments. For example, GMT +1 hour is represented as 2.

Returns 1 if successful or 0 if unsuccessful.

OptionsSetCurrentTimeZoneId

Sets the unique identifier for the current time zone.

```
bool OptionsSetCurrentTimeZoneId(unsigned short tzID)
```

Parameters `tzID` The unique time zone identifier to set.

Returns 1 if successful or 0 if unsuccessful.

Description `OptionsSetCurrentTimeZoneId` switches the handheld to a different time zone, and adjusts the current time.

OptionsSetDateTime

Sets the date and time.

```
void OptionsSetDateTime(TIME * NewDateTime)
```

Parameters `NewDateTime` A pointer to a `TIME` structure that is populated with the new time.

OptionsSetDateTimeDST

Sets the date and time if time is adjusted for Daylight Savings Time.

```
void OptionsSetDateTimeDST(TIME * NewDateTime, bool bInDST)
```

Parameters

`NewDateTime`	A pointer to a `TIME` structure that is populated with the new time.
`bInDST`	Boolean indicating if the time is in Daylight Savings Time mode.

Description `OptionsSetDateTimeDST` is useful in instances in which Daylight Savings Time and standard time overlap.

OptionsSetDaylightEnabled

Enables Daylight Savings Time on the handheld. Deprecated, ignored.

```
void OptionsSetDaylightEnabled(int enabled)
```

Parameters

`enabled`	If not set to 0, DST is enabled; if set to 0, DST is disabled.

Description `OptionsSetDaylightEnabled` enables Daylight Savings Time on the handheld, but does not change the state of the handheld.

OptionsSetDaylightSavings

Enables applications to put the handheld into Daylight Savings Time. Deprecated, ignored.

```
void OptionsSetDaylightSavings(int state)
```

Parameters

`state`	Determines if DST is active. If not set to 0, the state of DST is set to active; if set to 0, DST is disabled.

RegisterApplication

Registers the name and build version of the application to display on the Status screen of Device Options.

```
int  RegisterApplication(
    char * name,
    int major,
    int minor,
    int revision)
```

Parameters

`name`	Specifies the name of the application.
`major`	Specifies the major integer (build).
`minor`	Specifies the minor integer (build).
`revision`	The integer that specifies the build version. For example, `1.2.0.0`.

Returns 1 if successful; 0 otherwise.

RegisterNotifyChoice

Adds an option to the notify menu.

```
bool RegisterNotifyChoice(Choice * ChoicePtr
    const char * ChoiceDescription)
```

Parameters

`ChoicePtr`	A pointer to the choice to register.
`ChoiceDescription`	A pointer to the choice description.

Returns True is successful; false otherwise.

Description `RegisterNotifyChoice` adds a new option to the **Profiles** menu on the handheld.

RegisterOptions

Registers a single choice field in one of the Options sections.

```
int RegisterOptions(
    int section,
    Choice * field)
```

Parameters

`section`	Specifies the section that is to be registered.
`field`	Specifies the different field choices.

Returns 0 if false or 1 if true.

Description This function enables you to add options to the existing options screen.

RegisterPasswordChange

Registers for notification when the password changes.

```
void RegisterPasswordChange(
    void (*func)(unsigned char * hash, int length, int enable))
```

Parameters

`(*func)(unsigned char * hash, int length, int enable)`	This parameter is a pointer to the function to be called when the password changes. The function is passed a hash of the password, the length of the password hash, and a flag that indicates whether or not security is enabled (1 == enabled).

Description Only one application can register to receive notification of password changes. The last application to call this function is given notification.

RegisterSecurity

Registers application for callback on security failure.

```
int RegisterSecurity(
    void (*func)(void),
    char * message)
```

Parameters

`(*func)(void)`	A callback function if security has failed (such as, an incorrect password has been entered more than ten times).
`message`	Reserved for future development.

Returns 1 if registers properly, 0 if does not register properly.

Description Up to 32 applications may register to receive notification on a security failure. After ten incorrect attempts to enter a password are made, a security failure occurs.

RemoveOption

Removes an option entry from the options list.

```
void RemoveOption(
    void (*func)(void),
    char const * name)
```

Parameters

`(*func)(void)`	A pointer to a function that accepts no parameters. This pointer must be the same as the one passed to the `AddOption` function.
`name`	The name of the option entry item. The same pointer that is passed to `AddOption` is expected.

Index of functions

O

Option
- AddOption(), 19
- ConvertTime(), 20
- DisplayAlarmFunction(), 20
- EnterSecurityPassword(), 20
- FormatDate(), 21
- FormatDay(), 22
- FormatDayDate(), 22
- FormatTime(), 22
- GetOwnerInformation(), 23
- GetOwnerName(), 23
- GetPassword(), 24
- NotifyUser(), 24
- OptionsEntry(), 24
- OptionsGetCurrentTimeZone(), 25
- OptionsGetCurrentTimeZoneId(), 25
- OptionsGetCurrentTimeZoneInfo(), 25
- OptionsGetDSTData(), 25
- OptionsGetIndexFromTimeZoneId(), 26
- OptionsGetTimeZoneDescriptions(), 26
- OptionsGetTimeZoneIdFromIndex(), 26
- OptionsGetTimeZoneInfo(), 27
- OptionsIsDateInDST(), 27
- OptionsIsDaylightSavingsActive(), 27
- OptionsIsDaylightSavingsEnabled(), 27
- OptionsRegisterTimeChangeCB(), 28
- OptionsSetCurrentTimeZone(), 28
- OptionsSetCurrentTimeZoneId(), 28
- OptionsSetDateTime(), 28
- OptionsSetDateTimeDST(), 29
- OptionsSetDaylightEnabled(), 29
- OptionsSetDaylightSavings(), 29
- RegisterApplication(), 29
- RegisterNotifyChoice(), 30
- RegisterOptions(), 30
- RegisterPasswordChange(), 30
- RegisterSecurity(), 31
- RemoveOption(), 31

R

Ribbon
- RibbonAddBitmap(), 7
- RibbonAddBitmapToElement(), 8
- RibbonAddElement(), 8
- RibbonGetPrevApplication(), 9
- RibbonModifyApplication(), 9
- RibbonRegisterApplication(), 10
- RibbonRegisterFunction(), 11
- RibbonRemoveBitmap(), 12
- RibbonSetApplicationString(), 13
- RibbonSetBitmap(), 14
- RibbonSetCursorPosition(), 14
- RibbonSetNextApplication(), 15
- RibbonSetPrevApplication(), 15
- RibbonShowRibbon(), 15
- RibbonUnregisterApplication(), 15

Index

A

adding
- bitmaps, 7
- options, 19

API functions
- Options, 17
- Ribbon, 7

application
- registering, 29

B

bitmaps
- adding, 7
- removing, 12
- setting, 14

D

date and time
- setting, 28

date formatting, 21
day formatting, 22
daylight savings time, 27
displaying
- options entry, 24

E

enabling daylight savings time, 29

F

Formats, 21
formatting
- date, 21
- day, 22
- time, 22

G

getting
- owner information, 23
- owner name, 23
- the current time zone, 25
- the previous application, 9

M

modifying applications, 9

N

notifying the user, 24

O

options
- adding, 19
- displaying the options, 24
- registering, 30
- removing from options list, 31

P

password
- entering, 20
- registering change, 30
- retrieving hash, 24

R

registering
- application, 29
- options, 30
- password change, 30
- time change, 28

registering applications, 10
registering functions, 11
removing
- bitmaps, 12
- option from options list, 31

ribbon
- api functions, 7

S

security
- registering, 31

setting
- application strings, 13
- bitmaps, 14
- date and time, 28
- daylight savings time, 29
- the current time zone, 28
- the next application, 15
- the previous application, 15

showing the ribbon, 15

T

time formatting, 22

U

unregistering applications, 15