QUALCOMM®

Binary Runtime Environment for Wireless™

BREW API REFERENCE

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA. 92121-1714
U.S.A.

This manual was written for use with the BREW SDK™) for Windows, software version 1.0.1. This manual and the BREW SDK software described in it are copyrighted, with all rights reserved. This manual and the BREW SDK software may not be copied, except as otherwise provided in your software license or as expressly permitted in writing by QUALCOMM Incorporated.

BREW API Reference

80-D4995-1 Rev. J

July 30, 2001

Contents

Introducing the BREW API Reference

This document provides developers with the information about Binary Runtime Environment for Wireless™ (BREW™) functions, and data structures needed to develop applications for BREW-enabled mobile platforms.

In this reference

This remainder of the BREW API Reverence contains the following sections:

BREW API Interfaces (See Contents)	Lists the BREW interfaces and functions contained in them in alphabetical order
Helper Functions	Lists the helper functions in alphabetical order
Data Structures	Lists the data structures used by the BREW interfaces in alphabetical order

Each Function is listed with the following Information:

Description	An explanation of the function's use
Prototype	A sample of the structure of a call
Parameters	The items needing to be input and items returning

NOTE: Parameter lists will show [in], [in/out] and [out] only when the tables of parameters have a mixture of types. If the tables are all input parameters, the "[in]" is omitted.

Return Values	The items returning from the function call which include a wide variety of types, messages, values, structures, and descriptions
Comments	Any special considerations and extra information to assist in understanding the function's use, limitations, and boundaries.
Side Effects	Any behavior that the function exhibits which may not be normally considered when using a function call.
See Also	A cross reference to any related function or data structure.

BREW documentation set

The BREW documentation set contains the following documents:

Document	Description
BREW SDK User's Guide	Introduces the components of the BREW Software Development Kit (BREW SDK™) and their relationships to one another. The document also contains general instructions for developing your own BREW applications.
BREW API Reference	Provides developers with information about BREW functions and data structures needed to develop applications for BREW-enabled mobile platforms.
BREW Device Configurator Guide	Describes the purpose and features of the BREW Device Configurator and its relationship with the BREW Emulator. The document also provides instructions for creating effective wireless devices for use in the BREW Emulator to facilitate application development.
BREW Resource Editor Guide	Describes how to use the BREW Resource Editor to create three types of resources that are utilized in BREW applications. The types of resources you can create are UI text strings, images, and dialogs.
BREW MIF Editor Guide	Describes how to use the BREW Module Information File (MIF) Editor to create and modify MIFs — a special type of BREW resource file that contains information about the classes and applets supported by particular BREW modules.

For more information

Online information and support is available for BREW application developers. Please visit the BREW web site for details: *www.qualcomm.com/brew.*

IApplet Interface

IApplet is the interface that represents an Application Execution Environment (AEE) Applet. The interface is derived from IBase and is generated by the associated IModule Interface. The IApplet Interface implements a simple HandleEvent routine. This routine is called by the AEE Shell in response to events generated by the system, other components, or applets. All applets in BREW must implement this interface. Stated in other words, “A BREW applet is a class that implements the IApplet Interface.” The IAPPLET_HandleEvent() function is used by the AEE Shell for sending events to the applet.

> **NOTE:** IAPPLET_HandleEvent() can only be called by the AEE Shell. Events sent by other applets or components can be sent via the ISHELL_SendEvent() function.

List of functions

Functions in this interface include:

IAPPLET_AddRef()

IAPPLET_HandleEvent()

IAPPLET_Release()

Return to the Contents.

IAPPLET_AddRef()

Description:

This function increments the reference count of IApplet Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IAPPLET_Release().

Prototype:

```
uint32 IAPPLET_AddRef(IApplet * pIApplet)
```

Parameters:

pIApplet	Pointer to the IApplet Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

IAPPLET_Release()
Return to the List of functions.

IAPPLET_HandleEvent()

Description:

This function provides the main event processing for a BREW applet. It is called when any event is passed to the applet. Events can include system-level notifications, keypress events, and so forth

System alarms or system notifications call this function to respond. In those cases, if the applet is not currently running, the applet is loaded and the event is sent to the applet. In such cases, the EVT_APP_START event is not sent to the applet and unless the applet starts itself, the applet terminates after the completion of the event. The **IAPPLET_HandleEvent()** function must be implemented by all applets.

The IAPPLET_HandleEvent() function is also used to support applet startup, shutdown, suspend, and resume.

Prototype:

```
boolean IAPPLET_HandleEvent(IApplet * pIApplet, AEEEvent evt, uint16 wp,
uint32 dwp)
```

Parameters:

pIApplet	Pointer to the IApplet Interface object
evt	Event code
wp	16-bit event-specific parameter
dwp	32-bit event-specific parameter

Return Value:

TRUE	If the event was handled by the applet
FALSE	If otherwise

Comments:

This function can only be called by the AEE Shell

Side Effects:

None

See Also:

ISHELL_SendEvent(),
AEE Events,
Return to the List of functions.

IAPPLET_Release()

Description:

This function decrements the reference count of IApplet Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IAPPLET_Release(IApplet * pIApplet)
```

Parameters:

pIApplet	Pointer to the IApplet Interface object

Return Value:

reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IAPPLET_AddRef()
Return to the List of functions.

IAStream Interface

The IAStream Interface reads data from an asynchronous stream. It is an abstract interface that is implemented by classes that provide access to data that may not all be available at once and must be retrieved asynchronously. At present, the IFile and ISocket classes implement the IAStream Interface. The IImage Interface and ISoundPlayer Interface implement SetStream functions that allow an IAStream Interface to be supplied as the source of image or sound input.

The function IASTREAM_Read() reads data from the stream and returns the number of bytes read. If no data is available for reading, this function returns the value AEE_STREAM_WOULDBLOCK. In the latter case, you can call the function IASTREAM_Readable() to schedule a callback function that is invoked when there is more data available. The function IASTREAM_Cancel() cancels a callback that was scheduled with IASTREAM_Readable().

To use an IAStream instance to retrieve data asynchronously, perform the following steps:

1. Call ISHELL_CreateInstance() to create an instance of a class that implements the IAStream interface.
2. Call IASTREAM_Read() to read the required number of bytes of data from the stream. Since IAStream Interface is an abstract interface, you can also call IFILE_Read() for files, or ISOCKET_Read() for sockets.
3. If IASTREAM_Read() returns AEE_STREAM_WOULDBLOCK, call IASTREAM_Readable() to schedule a callback to try again later.
4. If IASTREAM_Read() reads fewer than the required number of bytes, call the function again to read the remaining data.
5. Repeat steps 2-4 until all the data has been received on the stream.

To use an asynchronous stream as the source of image or sound data, perform the following steps:

1. Call ISHELL_CreateInstance() to create an instance of a class that implements the IAStream Interface.
2. Call IAStream Interface to create an instance of IImage Interface or ISoundPlayer Interface.

3 Call IIMAGE_Notify() or ISOUNDPLAYER_RegisterNotify() to schedule a callback that is invoked when the image or sound data has been completely retrieved.

4 Call IIMAGE_SetStream() or ISOUNDPLAYER_SetStream() to associate the stream created in step 1 with the IImage or ISoundPlayer instance created in step 2. This initiates the retrieval of image or sound data on the stream.

5 When retrieval is complete, the callback registered in step 3 is invoked. You can then use functions in the IImage Interface or the ISoundPlayer Interface to access the retrieved image or sound data.

List of functions

Functions in this interface include:

IASTREAM_AddRef()

IASTREAM_Cancel()

IASTREAM_Read()

IASTREAM_Readable()

IASTREAM_Release()

Return to the Contents.

IASTREAM_AddRef()

Description:

This function increments the reference count of IAStream Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IASTREAM_Release().

Prototype:

```
uint32 IASTREAM_AddRef(IAstream * pIAstream)
```

Parameters:

pIAstream	Pointer to the IAStream Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

IASTREAM_Release()
Return to the List of functions

IASTREAM_Cancel()

Description:

This function cancels a callback that was scheduled with IASTREAM_Readable.

Prototype:

```
void IASTREAM_Cancel(IAStream * pIAStream, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIAStream	Pointer to an IAStream Interface object
pfn	Pointer to the callback function of the callback to be cancelled
pUser	Pointer to user-specified data that is passed as a parameter to the callback function and is used to identify which callback must be cancelled.

Return Value:

None.

Comments:

None

Side Effects:

None

See Also:

IASTREAM_Read()
IASTREAM_Readable()
Return to the List of functions

IASTREAM_Read()

Description:

This function attempts to read data from a stream, and returns the number of bytes read. If no data is available for reading, it returns the value AEE_STREAM_WOULDBLOCK.

Prototype:

```
int32 IASTREAM_Read (IAStream * pIAStream, void * pBuffer, uint32 dwCount)
```

Parameters:

pIAStream	Pointer to an IAStream Interface object
pBuffer	Pointer to the buffer into which the data will be read
dwCount	Number of bytes to read

Return Value:

Number of bytes read	If data is available for reading
0 (zero)	If all available data has been read AEE_STREAM_WOULDBLOCK if no data is currently available

Comments:

None

Side Effects:

None

See Also:

IASTREAM_Readable()
IASTREAM_Cancel()
Return to the List of functions

IASTREAM_Readable()

Description:

This function registers a callback that checks whether data is available to be read. The IASTREAM_Readable() is called when the IASTREAM_Read() returns AEE_STREAM_WOULDBLOCK.

Prototype:

```
void IASTREAM_Readable(IAStream * pIAStream, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIAStream	Pointer to an IAStream Interface object
pfn	Pointer to the callback function
pUser	Pointer to user-specified data that is passed as a parameter to the callback function

Return Value:

None.

Comments:

None

Side Effects:

None

See Also:

IASTREAM_Read()
IASTREAM_Cancel()
Return to the List of functions

IASTREAM_Release()

Description:

This function decrements the reference count of IAStream Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IASTREAM_Release(IAstream * pIAstream)
```

Parameters:

pIAstream	Pointer to the IAStream Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IASTREAM_AddRef()
Return to the List of functions

IBase Interface

IBase is the base level interface from which all other BREW interfaces are derived. It supplies the object reference counting mechanisms that allow objects to manage their own memory instances.

List of functions

Functions in this interface include:

IBASE_AddRef()

IBASE_Release()

Return to the Contents

IBASE_AddRef()

Description:

This function increments the reference count of IBase Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IBASE_Release().

Prototype:

```
uint32 IBASE_AddRef(IBase * pIBase)
```

Parameters:

pIBase	Pointer to the IBase Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

IBASE_Release()
Return to the List of functions

IBASE_Release()

Description:

This function decrements the reference count of IBase Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IBASE_Release(IBase * pIBase)
```

Parameters:

pIBase	Pointer to the IBase Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IBASE_AddRef()
Return to the List of functions

IControl Interface

The IControl Interface is an abstract interface that is implemented by each of the BREW control interfaces (IDateCtl Interface, IMenuCtl Interface, IStatic Interface, ITextCtl Interface and ITimeCtl Interface). Because the interface is abstract, it is not possible to create an instance of the IControl Interface directly. Given a pointer to an instance of a control interface, you can invoke an IControl Interface function using either an IControl function or a function in that control's interface. For example, a menu control can be redrawn by calling either ICONTROL_Redraw() or IMENUCTL_Redraw(). The IControl functions are useful if you want to perform the same operation on many different types of control. For example, suppose that the array **pControls[numControls]** contains pointers to controls of different types that must be displayed together on the screen. The following loop can be used to redraw all the controls:

```
for (i = 0; i < numControls; i++)
ICONTROL_Redraw(pControls[i])
```

Each BREW control implements all of the functions in the IControl Interface. However, the behavior of each function may be different in each interface; refer to the function descriptions of each control interface for details. Each BREW control also implements control-specific functions that are not part of the IControl Interface.

The IControl Interface functions include:

- ICONTROL_HandleEvent() is called to pass events to a control. The BREW controls process various key events to allow a user to enter a text, time or date value or choose an item from a menu. Refer to the descriptions of each control for the events it handles. A control can

receive events only when it is active. Your applet must pass a control any events it handles when it is active, unless the control is part of a dialog.

- ICONTROL_Redraw() draws the control on the screen. This function can be used to re-display a control after it has been overwritten.

- ICONTROL_SetActive() sets the activity state (active or inactive) of the control. Only an active control can receive and process events. In the case of a multicontrol dialog, only the control that currently has the focus is active.

- ICONTROL_IsActive() retrieves the activity state of a control.

- ICONTROL_SetRect() sets the pixel dimensions of the screen rectangle in which the control will be displayed. The pixel dimensions are specified in an AEERect structure that is passed as a parameter to the function. Some controls provide a scrolling mechanism that is used when the contents of the control will not fit in the rectangle. You can display multiple controls on the screen at the same time by setting their rectangle sizes appropriately.

- ICONTROL_GetRect() retrieves the current rectangle of a control.

- ICONTROL_SetProperties() allows you to change a control's property values, which are used to customize the control's behavior and appearance by enabling some optional features. Each control can have up to 32 properties, with each represented by a bit in a 32-bit variable. ICONTROL_SetProperties() allows you to set the value of this variable. These properties are all unset initially, and a property is set by turning its bit on. Each BREW control has a different set of properties; refer to the descriptions of each control for a list of the properties it supports. The header file for each BREW control has a set of bit-mask constants that can be used to test and set the values of the bits corresponding to each property.

- ICONTROL_GetProperties() retrieves the current value of the control's property value. To set a property without changing the values of the other properties, you can do the following:

```
dwProps = ICONTROL_GetProperties(pIControl);
ICONTROL_SetProperties(pIControl,(dwProps | PROP_BITMASK));
```

  where **pIControl** is a pointer to the control and **PROP_BITMASK** is a bit-mask for the property to be set.

- ICONTROL_Reset() frees all the resources associated with the control, but does not release its interface pointer. This function can be used to re-initialize a control.

List of functions

Functions in this interface include:

ICONTROL_AddRef()

ICONTROL_GetProperties()

ICONTROL_GetRect()

ICONTROL_HandleEvent()

ICONTROL_IsActive()

ICONTROL_Redraw()

ICONTROL_Release()

ICONTROL_Reset()

ICONTROL_SetActive()

ICONTROL_SetProperties()

ICONTROL_SetRect()

Return to the Contents

ICONTROL_AddRef()

Description:

This function increments the reference count of the IControl Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ICONTROL_Release().

Prototype:

```
uint32 ICONTROL_AddRef(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

ICONTROL_Release()
Return to the List of functions

ICONTROL_GetProperties()

Description:

This function returns the control-specific properties or flags.

Prototype:

```
uint32 ICONTROL_GetProperties(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

32-bit properties	For the control

Comments:

None

Side Effects:

None

See Also:

ICONTROL_SetProperties()
Return to the List of functions

ICONTROL_GetRect()

Description:

This function fills a pointer to an input AEERect structure with the active screen coordinates for the control. This is particularly useful after a control is created to determine its optimal/default size and position.

Prototype:

```
void ICONTROL_GetRect(IControl * pIControl, AEERect * prc)
```

Parameters:

pIControl	Pointer to the IControl Interface object
prc	Rectangle to be filled with the coordinates of the control

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ICONTROL_SetRect()
AEERect
Return to the List of functions

ICONTROL_HandleEvent()

Description:

This function provides the main event processing for a control. It is called when any event is passed to the control. Events mainly include keypress events. This function must be implemented by all controls.

Prototype:

```
boolean ICONTROL_HandleEvent(IControl * pIControl, AEEEvent evt, uint16 wp,
uint32 dwp)
```

Parameters:

pIControl	Pointer to the IControl Interface object
evt	Event code
wp	16-bit event data
dwp	32-bit event data

Return Value:

TRUE	If the event was processed by the control
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ICONTROL_IsActive()

Description:

This function returns the active or focus state of the control.

Prototype:

```
boolean ICONTROL_IsActive()(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

TRUE	If the control is active
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ICONTROL_Redraw()

Description:

This function instructs the control to redraw its contents. Under normal conditions, user interface controls do not redraw their contents when the underlying data behind the control changes. This allows several data updates to occur while minimizing screen flashes. For example, several items can be added to a menu with no visible effect until the Redraw function is called.

Prototype:

```
boolean ICONTROL_Redraw(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

TRUE	If the event was processed by the control
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ICONTROL_Release()

Description:

This function decrements the reference count of IControl Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ICONTROL_Release(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

ICONTROL_AddRef()
Return to the List of functions

ICONTROL_Reset()

Description:

This function instructs the control to reset (free/delete) its contents as well as to immediately leave active/focus mode. This is useful in freeing all underlying memory in text or menu controls or removing all menu items in a single call.

Prototype:

```
void ICONTROL_Reset(IControl * pIControl)
```

Parameters:

pIControl	Pointer to the IControl Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ICONTROL_SetActive()
Return to the List of functions

ICONTROL_SetActive()

Description:

This function instructs the control to enter/leave focus or selected mode. The concept of focus is left somewhat to the control. In the case of menus, focus indicates that the menu is active. In terms of text controls it means the control is active and in edit mode. This call usually results in the underlying control redrawing its contents. It is important to know that controls still have their HandleEvent function called even when they are inactive. This allows them to process special events such as scrolling multiline text controls.

Prototype:

```
void ICONTROL_SetActive(IControl * pIControl,boolean bActive)
```

Parameters:

pIControl	Pointer to the IControl Interface object
bActive	Specifies whether to activate (TRUE) or deactivate (FALSE) the control

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ICONTROL_SetProperties()

Description:

This function sets control-specific properties or flags. Although some properties are defined across controls (CP_FRAME, and so forth) , most controls expose a range of properties to allow more specific control over the object.

Prototype:

```
void ICONTROL_SetProperties(IControl * pIControl, uint32 dwProps)
```

Parameters:

pIControl	Pointer to the IControl Interface object
dwProps	32-bit set of flags/properties

Return Value:

None

Comments:

None

Side Effects:

May redraw the control

See Also:

ICONTROL_GetProperties()
Return to the List of functions

ICONTROL_SetRect()

Description:

This function sets the active screen coordinates of the control. This may result in the control redrawing its contents.

Prototype:

```
void ICONTROL_SetRect(IControl * pIControl, const AEERect * prc)
```

Parameters:

pIControl	Pointer to the IControl Interface object
prc	The bounding rectangle for the control

Return Value:

None

Comments:

None

Side Effects:

May redraw the control

See Also:

ICONTROL_GetRect()
Return to the List of functions

IDatabase Interface

The IDatabase Interface functions allow you to create and access records in databases created and opened with the IDBMgr Interface. To obtain an instance of the IDatabase Interface, you call IDBMGR_OpenDatabase() to open the desired database. You then use the IDatabase Interface pointer returned by this function to access the database with the operations described later in this section. You can also use functions in the IDBRecord Interface to access the fields of individual database records. When you have completed access to the database, you call IDATABASE_Release() to close it.

> **CAUTION:** Your application must have a privilege level of File or All to be able to invoke the functions in this interface that modify the contents of the database.

The IDATABASE_CreateRecord() function creates a new record and adds it to your database (the function IDBRECORD_Remove() is used to remove a record from the database). Each record contains one or more fields. Each field is defined by the AEEDBField structure, which includes the following elements:

The field **name** is a descriptor of the field's contents, (name, phone number, email address, and so forth) The AEEDBFieldName enumerated type contains constants for commonly used field names.

The field **type** gives the data type of the field (byte, word, double-word, character string, binary, phone number or bitmap).

The field **buffer** pointer is a pointer to the actual contents of the field.

The field **length** is the length in bytes of the field contents.

When creating a record, you populate an array of AEEDBField structures to specify the name, type, contents and length of each field in the record. You then call IDATABASE_CreateRecord(), supplying a pointer to this array and the number of AEEDBField structures it contains as input. IDATABASE_CreateRecord() returns a pointer to an instance of the IDBRecord Interface that can be used to access and update fields of the record. The number and type of fields are specified on a per-record basis when the record is created or updated; there is no requirement that all records in a given database have the same structure.

Once you have created a database and added records to it, you can use other IDatabase functions to retrieve records from the database.

To retrieve every record in the database

1 call IDATABASE_Reset() to set the record index to 0 (zero).

2 Repeatedly call IDATABASE_GetNextRecord() to obtain an IDBRecord Interface pointer for each record in the database. IDATABASE_GetNextRecord() returns NULL when all the records have been enumerated.

Each database record is assigned a unique ID when it is created (the ID of a record can be obtained with the IDBRECORD_GetID() function). The function IDATABASE_GetRecordByID() lets you retrieve a database record with a given ID, returning an IDBRecord pointer that can be used to access the record. The function IDATABASE_GetRecordCount() returns the number of records in the database.

To use functions in the IDatabase Interface

1 Call ISHELL_CreateInstance() if necessary to obtain an instance of the IDBMgr Interface.

2 Call IDBMGR_OpenDatabase() or IDBMGR_OpenDatabaseEx() to obtain an IDatabase Interface pointer to a new or existing database.

3 Call IDATABASE_CreateRecord() to create new records and add them to the database opened in step 2.

4 Call IDATABASE_Reset() and IDATABASE_GetNextRecord() if you need to enumerate all the records in the database, for example, to find all records that match certain criteria. Call IDATABASE_GetRecordByID() to retrieve a particular record given its ID.

5 Call IDATABASE_Release() to close the database when you have completed accessing it.

List of functions

Functions in this interface include:

IDATABASE_AddRef()

IDATABASE_CreateRecord()

IDATABASE_GetNextRecord()

IDATABASE_GetRecordByID()

IDATABASE_GetRecordCount()

IDATABASE_Release()

IDATABASE_Reset()

Return to the Contents

IDATABASE_AddRef()

Description:

This function increments the reference count of IDatabase Interface object.

Prototype:

```
uint32 IDATABASE_AddRef(IDatabase * pIDatabase)
```

Parameters:

pIDatabase Pointer to IDatabase Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IDATABASE_Release()
Return to the List of functions

IDATABASE_CreateRecord()

Description:

This function creates a new database record with the fields specified by **pDBFields** in the database specified by **pIDatabase**.

Prototype:

```
IDBRecord * IDATABASE_CreateRecord( IDatabase * pIDatabase, AEEDBField *
pDBFields, int iNumfields)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object in which the record is being created
pDBFields	Pointer to the database fields that need to be placed in a new record created by this function
iNumFields	Number of fields in the record

Return Value:

Pointer	Pointer to the database record created, if successful
NULL	If unsuccessful

Comments:

A new record is added to the database. The IDBRecord must be released using the IDBRECORD_Release() before the database is released. If records are not released, IDATABASE_Release() cannot close the database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDATABASE_GetNextRecord()

Description:

This function returns the next IDBRecord from the database.

Prototype:

```
IDBRecord * IDATABASE_GetNextRecord(IDatabase * pIDatabase)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object whose next record is requested

Return Value:

Pointer	Pointer to the database record, if successful
NULL	If otherwise

Comments:

The IDBRecord must be released using the IDBRECORD_Release() before releasing the database. If records are not released, IDATABASE_Release() cannot close the database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDATABASE_GetRecordByID()

Description:

This function returns a pointer to the record whose record ID is specified.

Prototype:

```
IDBRecord * IDATABASE_GetRecordByID( IDatabase * pIDatabase, uint16
u16RecID)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object whose record is requested
u16RecID	Index of record to get

Return Value:

Pointer	Pointer to the database record whose index is specified, if successful
NULL	If unsuccessful

Comments:

The IDBRecord must be released using the IDBRECORD_Release() before releasing the database. If records are not released, IDATABASE_Release() cannot close the database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDATABASE_GetRecordCount()

Description:

This function returns the number of records in the database specified by **pIDatabase**.

Prototype:

```
uint32 IDATABASE_GetRecordCount(IDatabase * pIDatabase)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object whose record count is requested

Return Value:

Number	Number of records in the database, if successful
0 (zero)	If **pIDatabase** is pointing to IDatabase Interface object with invalid database handle

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDATABASE_Release()

Description:

This function decrements the reference count of the IDatabase Interface object. If the reference count reaches 0 (zero), this function closes the database.

Prototype:

```
uint32 IDATABASE_Release(IDatabase * pIDatabase)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object whose reference count needs to be decremented

Return Value:

The updated reference count for the object

Comments:

Before closing a database all the records of the database must be released using the IDBRECORD_Release(). If records are not released, IDBRECORD_Release() cannot close the database.

Side Effects:

None

See Also:

IDATABASE_AddRef()
IDBRECORD_Release()
IDBMGR_OpenDatabase()
IDBMGR_OpenDatabaseEx()
Return to the List of functions

IDATABASE_Reset()

Description:

This function resets the record index of the database specified by **pIDatabase**.

Prototype:

```
void IDATABASE_Reset(IDatabase * pIDatabase)
```

Parameters:

pIDatabase	Pointer to the IDatabase Interface object whose record index needs to be reset

Return Value:

None

Comments:

Calling IDATABASE_GetNextRecord() after IDATABASE_Reset() gets the first record in the database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDateCtl Interface

Date controls are used in interfaces that require the device user to choose a date. There are two types of date controls (you choose the type you want by specifying its ClassID when you create an instance of the date control):

A standard date control (ClassID AEECLSID_DATECTL) allows the user to use the UP, DOWN, LEFT and RIGHT keys to choose the desired month, day and year.

A date pick control (ClassID AEECLSID_DATEPICKCTL) displays a monthly calendar; the arrow keys can be used to choose the day of the month, or to scroll to the next or previous month.

IDATECTL_HandleEvent() function handles the AVK_SELECT, AVK_UP, AVK_DOWN, AVK_LEFT and AVK_RIGHT keys. When it receives AVK_SELECT, the date control sends an EVT_COMMAND to signal your application that the user has selected a date, provided you have enabled command sending (see later in this section); you can then use one of the IDateCtl functions to retrieve the selected date value. For standard date controls, control tabbing events (EVT_CTL_TAB) are sent when the user presses AVK_LEFT while selecting the month or AVK_RIGHT while selecting the year. You can use these events to allow the user to navigate between controls in a multicontrol screen (dialogs handle EVT_CTL_TAB events and change control focus as needed).

At present, there are no properties specific to date controls, so the functions IDATECTL_SetProperties and IDATECTL_GetProperties() are not used.

IDateCtl Interface implements several functions in addition to those in the IControl Interface.

IDATECTL_SetTitle() is used to specify a title that appears above the date control.

IDATECTL_SetDate() sets the date stored in the control to the value specified in the function's integer month, day, and date parameters. IDATECTL_GetDate() retrieves the control's current date in the same format. IDATECTL_SetJulianDay() and IDATECTL_GetJulianDay() are similar, except that the control's date is set and retrieved in JulianType date format (number of seconds since January 1, 1980 GMT). The function IDATECTL_GetDayOfWeek() returns the day of the week corresponding to the control's current date.

The date control's current date, day, or month can also be stored into a character string with the functions IDATECTL_GetDateString(), IDATECTL_GetDayString(), and IDATECTL_GetMonthString(). The date string can be in any of several different formats (for example, you can specify the order of the date, month and year, and whether the full month name or a three-letter abbreviation is used).

For date pick controls, IDATECTL_SetActiveDayMask() causes specified days of the month to be displayed in reverse video on the date pick calendar display; this can be used to designate holidays, For example, the function is called with a 32-bit parameter that specifies which of the days of the month is displayed in this way.

IDATECTL_EnableCommand() allows the disabling or enabling of command sending when the user presses the AVK_SELECT key while the date control is active (command sending is disabled by default). This function also lets you specify the command ID (for example, the **wParam** value that is sent along with the EVT_COMMAND when your application's IAPPLET_HandleEvent() function is called).

To use a date control

1. Call ISHELL_CreateInstance() to create an instance of a date control, specifying the ClassID of either a standard or date-pick date control.
2. Call IDATECTL_SetRect() to specify the screen rectangle that contains the control.
3. Call IDATECTL_SetDate() or IDATECTL_SetJulianDay() to specify an initial date value for the control if necessary (if you do not specify one, the control's date fields appears blank initially).
4. Call IDATECTL_SetActive() to make the control active so that it can receive key events generated as the user chooses a date value. Your application must send the control these events using the IDATECTL_HandleEvent() while the control is active.
5. When the user has chosen a date, you can call one of the date-retrieval functions mentioned above to access its value in the desired format. The user's selection of a date may be signaled via an EVT_COMMAND event if command sending is enabled, or by the user exiting the screen that contains the date control.
6. When you no longer need the date control, you can call IDATECTL_Release() to release it.

List of functions

Functions in this interface include:

IDATECTL_AddRef()

IDATECTL_EnableCommand()

IDATECTL_GetDate()

IDATECTL_GetDateString()

IDATECTL_GetDayOfWeek()

IDATECTL_GetDayString()

IDATECTL_GetJulianDay()

IDATECTL_GetMonthString()

IDATECTL_GetProperties()

IDATECTL_GetRect()

IDATECTL_HandleEvent()

IDATECTL_IsActive()

IDATECTL_Redraw()

IDATECTL_Release()

IDATECTL_Reset()

IDATECTL_SetActive()

IDATECTL_SetActiveDayMask()

IDATECTL_SetDate()

IDATECTL_SetJulianDay()

IDATECTL_SetProperties()

IDATECTL_SetRect()

IDATECTL_SetTitle()

Return to the Contents

IDATECTL_AddRef()

Description:

This function increments the reference count for the IDateCtl Interface object

Prototype:

```
uint32 IDATECTL_AddRef(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to IDateCtl Interface object

Return Value:

Updated reference count

Comments:

None

Side Effects:

None

See Also:

IDATECTL_Release()
Return to the List of functions

IDATECTL_EnableCommand()

Description:

This function is used to enable the date control object to send a user defined command to the active applet. If **bEnable** is TRUE, upon receiving the event generated by press of select key, the date control object sends **nCmdId** as EVT_COMMAND to the active applet. Handling of this event is applet-specific and is user-defined.

Prototype:

```
void IDATECTL_EnableCommand(IDateCtl * pIDateCtl, boolean bEnable, uint16
nCmdId)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
bEnable	Boolean value for enable flag
nCmdId	Command ID

Return Value:

None

Comments:

This function can be used to send a user defined command from the date control object to the active applet.

Side Effects:

None

See Also:

None
Return to the List of functions

IDATECTL_GetDate()

Description:

This function gets the date from the date control object.

Prototype:

```
boolean IDATECTL_GetDate (IDateCtl * pIDateCtl, unsigned int * pnYear,
unsigned int * pnMonth, unsigned int * pnDay )
```

Parameters:

pIDateCtl	Pointer to IDateCtl Interface object
pnYear	Placeholder for year [YYYY] for example, 2000
pnMonth	Placeholder for month [MM] for example, 12
pnDay	Placeholder for day [DD] for example, 31

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IDATECTL_SetDate()
Return to the List of functions

IDATECTL_GetDateString()

Description:

This function gets the date string in specified format.

Prototype:

```
boolean IDATECTL_GetDateString(IDateCtl * pIDateCtl, AECHAR * pBuffer,
unsigned int nMaxSize, unsigned int * pnChars, uint32 dwDateFormat)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object	
pBuffer	Placeholder for date string	
nMaxSize	Maximum number of characters to be read in the buffer	
pnChars	Placeholder for number of characters written in **pBuffer**	
dwDateFormat	Format of the date string. Use one of the date string formats given:	
	DFMT_DD_MONTH_YYYY	"18 July 2000"
	DFMT_DD_MON_YYYY	"18 Jul 2000"
	DFMT_DD_MON_YY	"18 Jul '00"
	DFMT_MONTH_DD_YYYY	"July 18, 2000"
	DFMT_MON_DD_YYYY	"Jul 18, 2000"
	DFMT_MON_DD_YY	"Jul 18, '00"

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

The date string formats specified for parameter **dwDateFormat** are mutually exclusive.

Side Effects:

None

See Also:

IDATECTL_GetDayString()
IDATECTL_GetMonthString()
Return to the List of functions

IDATECTL_GetDayOfWeek()

Description:

This function gets the day of week from the date control object.

Prototype:

```
uint16 IDATECTL_GetDayOfWeek(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

One of the following

DOW_SUNDAY

DOW_MONDAY

DOW_TUESDAY

DOW_WEDNESDAY

DOW_THURSDAY

DOW_FRIDAY

DOW_SATURDAY

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDATECTL_GetDayString()

Description:

This function gets the name of the day corresponding to the date control object's current date.

Prototype:

```
AECHAR * IDATECTL_GetDayString(IDateCtl * pIDateCtl, AECHAR * pBuffer,
unsigned int nMaxSize, unsigned int * pnChars)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
pBuffer	Placeholder for day name
nMaxSize	Buffer size
pnChars	Placeholder for number of characters written in **pBuffer**

Return Value:

pointer	Pointer to the end of day string in **pBuffer**, if successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IDATECTL_GetMonthString()
IDATECTL_GetDateString()
Return to the List of functions

IDATECTL_GetJulianDay()

Description:

This function gets the Julian day value of the specified date control object.

Prototype:

```
int32 IDATECTL_GetJulianDay(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

Julian day value of date control object

Comments:

None

Side Effects:

None

See Also:

IDATECTL_GetJulianDay()
Return to the List of functions

IDATECTL_GetMonthString()

Description:

This function gets the name of the month of the date control object's current date.

Prototype:

```
AECHAR * IDATECTL_GetMonthString(IDateCtl * pIDateCtl, AECHAR * pBuffer,
unsigned int nMaxSize, unsigned int * pnChars)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
pBuffer	Placeholder for month name
nMaxSize	Buffer size
pnChars	Placeholder for number of characters written in **pBuffer**

Return Value:

Pointer	Pointer to the end of month name string in **pBuffer**, if successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IDATECTL_GetDayString()
IDATECTL_GetDateString()
Return to the List of functions

IDATECTL_GetProperties()

Description:

This function returns the date control-specific properties or flags. Presently there are no date control-specific properties and this function always returns 0 (zero).

Prototype:

```
uint32 IDATECTL_GetProperties(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

0 (zero)

Comments:

None

Side Effects:

None

See Also:

IDATECTL_SetProperties()
Return to the List of functions

IDATECTL_GetRect()

Description:

This function fills the given pointer to an AEERect structure with the coordinates of the current bounding rectangle of the date control object. This is particularly useful after a control is created to determine its optimal/default size and position.

Prototype:

```
void IDATECTL_GetRect(IDateCtl * pIDateCtl, AEERect * prc)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
prc	Rectangle to be filled with the coordinates of the date control object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IDATECTL_SetRect()
Return to the List of functions

IDATECTL_HandleEvent()

Description:

This function is used to handle the events received by date control object. A date control object handles events received by it only if it is active. The events processed by the date control object are the press of UP, DOWN, LEFT and RIGHT keys. If command sending is enabled for the date control object, upon receiving event generated by the press of the SELECT key, it sends the command specified by IDATECTL_EnableCommand function as command event to the active applet.

Prototype:

```
boolean IDATECTL_HandleEvent(IDateCtl * pIDateCtl, AEEEvent evt, uint16 wp,
uint32 dwp)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
evt	Event code
wp	[AVK_UP|AVK_DOWN|AVK_LEFT|AVK_RIGHT|AVK_SELECT]
dwp	Key symbol string corresponding to the key type specified by wp

Return Value:

TRUE	If the event was processed by the date control
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

IDATECTL_EnableCommand()
Return to the List of functions

IDATECTL_IsActive()

Description:

This function returns whether the date control object is active or not. The active state is indicated by a return value of TRUE whereas the inactive state is indicated by a return value of FALSE.

Prototype:

```
boolean IDATECTL_IsActive(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

TRUE	If the date control is active
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

IDATECTL_SetActive()
Return to the List of functions

IDATECTL_Redraw()

Description:

This function instructs the date control object to redraw its contents. The Date control object does not redraw its contents every time the underlying data of the date control changes. This allows several data updates to occur while minimizing screen flashes.

Prototype:

```
boolean IDATECTL_Redraw(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

TRUE	If the date control was redrawn
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDATECTL_Release()

Description:

This function decrements the reference count for the date control object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IDATECTL_Release(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

Updated reference count of the object

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDATECTL_Reset()

Description:

This function instructs the date control to reset (free/delete) its contents as well as to immediately leave active/focus mode.

Prototype:

```
void IDATECTL_Reset(IDateCtl * pIDateCtl)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object

Return Value:

None

Comments:

This function makes the control inactive. An inactive control doesn't handle the events sent to it.

Side Effects:

None

See Also:

IDATECTL_SetActive()
Return to the List of functions

IDATECTL_SetActive()

Description:

This function is used to make a date control object active. Only an active date control object handles the events sent to it. An inactive date control object just ignores the events.

Prototype:

```
void IDATECTL_SetActive(IDateCtl * pIDateCtl,boolean bActive)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
bActive	Boolean flag that specifies whether to activate (TRUE) or deactivate (FALSE) the date control

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IDATECTL_IsActive()
Return to the List of functions

IDATECTL_SetActiveDayMask()

Description:

This function sets a new active day mask. The active day mask is a 32-bit integer in which each of 0 (zero) to 30 bits specifies active status of a day. An active day, having bit corresponding to it set, is drawn with a dot (.) at the upper left hand corner.

Prototype:

```
void IDATECTL_SetActiveDayMask(IDateCtl * pIDateCtl, uint32 dwMask)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
dwMask	New active day mask

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDATECTL_SetDate()

Description:

This function assigns the specified date to the date control object.

Prototype:

```
boolean IDATECTL_SetDate(IDateCtl * pIDateCtl, unsigned int nYear, unsigned
int nMonth, unsigned int nDay)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
nYear	Year[YYYY] for example, 2000
nMonth	Month[MM] for example, 12
nDay	Day[DD] for example, 31

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

Minimal error checking: any day that is between 1 and 31 is valid and any month that is between 1 and 12 is valid. An input of 4/31/98 becomes 5/1/98.

Side Effects:

None

See Also:

IDATECTL_GetDate()
Return to the List of functions

IDATECTL_SetJulianDay()

Description:

This function assigns the specified Julian day to the date control object.

Prototype:

```
boolean IDATECTL_SetJulianDay(IDateCtl * pIDateCtl, int32 lJulDate)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
lJulDate	Julian day to be assigned

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IDATECTL_GetJulianDay()
Return to the List of functions

IDATECTL_SetProperties()

Description:

This function sets date control-specific properties or flags. Presently there are no date control-specific properties or flags to be set.

Prototype:

```
void IDATECTL_SetProperties(IDateCtl * pIDateCtl, uint32 dwProps)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
dwProps	32-bit set of flags/properties

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IDATECTL_GetProperties()
Return to the List of functions

IDATECTL_SetRect()

Description:

This function can be used to set the specified rectangle as the bounding rectangle of the date control object. If the date control object is in month view, the specified rectangle is also used to determine the grid used for drawing days.

Prototype:

```
void IDATECTL_SetRect(IDateCtl * pIDateCtl, const AEERect * prc)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
prc	Bounding rectangle for the date control object

Return Value:

None

Comments:

By default, entire device screen is set as the bounding rectangle of the date control object.

Side Effects:

None

See Also:

IDATECTL_GetRect()
Return to the List of functions

IDATECTL_SetTitle()

Description:

This function is used to set title of a date control object. If pText is not NULL, it sets the string specified by pText as the title of the date control object. If pText is NULL, it reads the title string corresponding to the given resource identifier from the resource file and sets it as the title of the date control object.

Prototype:

```
boolean IDATECTL_SetTitle(IDateCtl * pIDateCtl, const char * pszResFile,
uint16 wResID, AECHAR * pText)
```

Parameters:

pIDateCtl	Pointer to the IDateCtl Interface object
pszResFile	Null terminated string containing resource file name
wResID	String resource identifier
pText	Null terminated title string

Return Value:

TRUE	If success
FALSE	If unsuccessful

Comments:

None

Side Effects:

If **pText** is NULL and **pszResFile**, **wResID** are valid, this function assigns the date control object title string to **pText**.

See Also:

None
Return to the List of functions

IDBMgr Interface

The IDBMgr Interface functions are used to create, open and remove databases, which are collections of multifield records. Once a database has been opened, you use functions in the IDatabase Interface to create and retrieve database records and close the database, and you use functions in the IDBRecord Interface to access and update the fields of individual records.

> **CAUTION:** Your application must have a privilege level of File or All to be able to invoke the functions in this interface that create or delete the database.

The function IDBMGR_OpenDatabase() opens an existing database given its name, which corresponds to the name of a BREW file that holds the database's contents. When calling this function, you can request that the database be created if it does not already exist. IDBMGR_OpenDatabaseEx() is similar, but it also allows you to specify a minimum record size and minimum number of records when creating a new database; the function reserves an amount of memory sufficient to hold the specified number of records. Both of these functions return a pointer to an instance of the IDatabase Interface, which can be used to access the opened database.

The function IDBMGR_Remove() removes a database by deleting the file that holds its contents. If the database is open, you must first call IDBMGR_Release() to close it prior to removal.

To use the functions in the IDBMgr Interface

1. Call ISHELL_CreateInstance() to create an instance of the IDBMgr Interface.
2. Call IDBMGR_OpenDatabase() to open an existing database or to create a new one. If you need to specify a minimum size for a newly created database, use IDBMGR_OpenDatabaseEx().
3. Using the IDatabase Interface pointer obtained in step 2, call functions in the IDatabase Interface to create and retrieve records in the database. You can also use IDBRecord Interface functions to access and modify the fields of database records.
4. Call IDATABASE_Release() to close the database when you have completed accessing it.
5. Call IDBMGR_Remove() to remove the database when necessary.
6. Call IDBMGR_Release() when you no longer need the IDBMgr Interface instance.

List of functions

Functions in this interface include:

IDBMGR_AddRef()

IDBMGR_OpenDatabase()

IDBMGR_OpenDatabaseEx()

IDBMGR_Remove()

IDBMGR_Release()

Return to the Contents

IDBMGR_AddRef()

Description:

This function increments the reference count of the IDBMgr Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IDBMGR_Release().

Prototype:

```
uint32 IDBMGR_AddRef(IDBMgr * pIDBMgr)
```

Parameters:

pIDBMgr Pointer to the IDBMgr Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IDBMGR_Release()
Return to the List of functions

IDBMGR_OpenDatabase()

Description:

This function opens the specified database. If database is already open, it returns NULL. Boolean flag bCreate specifies action to be taken if database does not exist. If the specified database does not exist, and bCreate is TRUE, this function creates a new database with minimum records and minimum record size parameters both set to 0 (zero).

Prototype:

```
IDatabase * IDBMgr_OpenDatabase(IDBMgr * pif, const char * pszFile, boolean bCreate)
```

Parameters:

pif	Pointer to the IDBMgr Interface object
pszFile	Null terminated string denoting the Database file name
bCreate	Specifies if the database must be created if the database is not found. The database is created only if this flag is set to true.

Return Value:

pointer	To the IDatabase object, if successful
NULL	If unsuccessful

Comments

Use IDATABASE_Release() for closing a database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDBMGR_OpenDatabaseEx()

Description:

This function opens the specified database. If database is already open, it returns NULL. Boolean flag bCreate specifies action to be taken if database does not exist. If the specified database does not exist, and bCreate is TRUE, this function creates a new database. While creating the database, the minimum record size can be specified by dwMinSize, and the minimum number of records can be specified by wMinRecs.

Prototype:

```
IDatabase * IDBMgr_OpenDatabaseEx(IDBMgr * pif, const char * pszFile, boolean bCreate, uint32 dwMinSize, uint16 wMinRecs)
```

Parameters:

pif	Pointer to the IDBMgr Interface
pszFile	Null terminated string denoting the Database file name
bCreate	Specifies if the database must be created if the database is not found. The database is created only if this flag is set to true.
dwMinSize	Minimum size of the records in the database
wMinRecs	Minimum number of records that the database can hold

Return Value:

pointer	To the IDatabase object, if successful
NULL	If unsuccessful

Comments:

Use IDATABASE_Release() for closing a database.

Side Effects:

None

See Also:

None
Return to the List of functions

IDBMGR_Remove()

Description:

This function removes the specified database.

Prototype:

```
int IDBMgr_Remove(IDBMgr * pif, const char * pszFile)
```

Parameters:

pif	IDBMgr * pointer to IDBMgr object
pszFile	Null terminated string specifying the name of the database to be removed

Return Value:

SUCCESS	If database was successfully removed
EBADFILENAME	If database file can not be found

Comments:

Once the database is removed, the corresponding database file is deleted. Subsequent opens or accesses to a removed database fail.

Side Effects:

None

See Also:

None
Return to the List of functions

IDBMGR_Release()

Description:

This function decrements the reference count of the IDBMgr Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IDBMGR_Release(IDBMgr * pIDBMgr)
```

Parameters:

pIDBMgr	Pointer to the IDBMgr Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IDBMGR_AddRef()
Return to the List of functions

IDBRecord Interface

The IDBRecord Interface contains a set of functions that are used to access and update the fields of database records. You use functions in the IDatabase Interface to obtain an instance of the IDBRecord Interface for a particular record. An IDBRecord Interface pointer is returned by IDATABASE_CreateRecord() when a new record is created, and by IDATABASE_GetNextRecord() and IDATABASE_GetRecordByID() when existing records are retrieved from a database. Once you have obtained an IDBRecord Interface pointer for a record, you can use it to access its fields with the operations described later in this section. When you have completed access to the database record, you call IDBRECORD_Release() to close it.

CAUTION: Your application must have a privilege level of File or All to be able to invoke the functions in this interface that modify the contents of the database.

Each field of a record contains its name, data type, a pointer to its contents, and the length of the contents. The record access functions in the IDBRecord Interface operate on the current field of the record. The function IDBRECORD_Reset() makes the first field of the record the current one, and IDBRECORD_NextField() advances the current field to next field in the record. IDBRECORD_NextField() also returns the name, data type and length (but not the contents) of the new current field, or an end-of-record indication when all the fields of the record have been enumerated. The function IDBRECORD_GetField() returns a pointer to the contents of the current field, and also returns the field's name, data type and length. The IDBRecord Interface provides some simpler functions that can be used to access a field's contents when the data type of the contents is already known: IDBRECORD_GetFieldWord(), IDBRECORD_GetFieldWord(), and IDBRECORD_GetFieldString() retrieve the contents of the current field when its type is word, double-word and character-string, respectively. Each of these functions returns a failure indication if the current field's contents are not of the appropriate type.

To modify the fields of a record, you use the function IDBRECORD_Update(). This function accepts the same input as IDATABASE_CreateRecord(): you supply a pointer to an array of AEEDBField structures that contains new values for the name, type, contents and length of each field in the record, along with the number of fields in the array. To update a single field in a record, you must specify values for all the fields of the record.

The function IDBRECORD_GetID() returns the unique identifier of a record. The record ID can be used when calling IDATABASE_GetRecordByID() to retrieve the record. IDBRECORD_Remove() removes the record from the database.

To use functions in the IDBRecord Interface

1 Call ISHELL_CreateInstance() if necessary to obtain an instance of the IDBMgr Interface.

2 Call IDBMGR_OpenDatabase() or IDBMGR_OpenDatabaseEx() to obtain an IDatabase Interface pointer to a new or existing database.

3 Call IDATABASE_CreateRecord(), IDATABASE_GetRecordByID() or IDATABASE_GetNextRecord() to obtain an IDBRecord Interface pointer for the record you wish to access.

To access the record as needed:

1 Call IDBRECORD_Reset() and IDBRECORD_NextField() to iterate through the fields of the record and obtain the name, data type and length of each field. To access the contents of the current field, use one of the IDBRECORD_GetField() functions described above.

2 Call IDBRECORD_GetID() to obtain the record's unique ID.

3 Call IDBRECORD_Update() to supply new values for all the fields of the record.

4 Call IDBRECORD_Remove() to remove the record from the database.

5 Call IDBRECORD_Release() to close the record when you have completed accessing it (if you removed the record in step 4, it is not necessary to release it here).

List of functions

Functions in this interface include:

IDBRECORD_AddRef()

IDBRECORD_GetField()

IDBRECORD_GetFieldDWord()

IDBRECORD_GetFieldString()

IDBRECORD_GetFieldWord()

IDBRECORD_GetID()

IDBRECORD_NextField()

IDBRECORD_Release()

IDBRECORD_Remove()

IDBRECORD_Reset()

IDBRECORD_Update()

Return to the Contents

IDBRECORD_AddRef()

Description:

This function increments the reference count of the IDBRecord Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IDBRECORD_Release().

Prototype:

```
uint32 IDBRECORD_AddRef(IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IDBRECORD_Release()
Return to the List of functions

IDBRECORD_GetField()

Description:

This function returns the raw data, type, name and length of the current field. The data returned does NOT include the header, and the length returned is the data length.

Prototype:

```
byte * IDBRECORD_GetField( IDBRecord * pIDBRecord,AEEDBFieldName * pName,
AEEDBFieldType * pDBFieldType, uint16 * pnLen)
```

Parameters:

pIDBRecord	[in]	Pointer to the IDBRecord Interface object whose field is requested
pName	[out]	Pointer to the field name
pDBFieldType	[out]	Pointer to the field type
pnLen	[out]	Pointer to length of the field (in bytes)

Return Value:

Pointer	To the buffer containing the requested field, if successful
NULL	If unsuccessful

Comments:

To illustrate the usage of this function, consider the following example:

If the field in the database record corresponds to AEEDBFIELD_LASTNAME, and contains an AECHAR string wStr of last name "Smith", then this function would return a pointer to a byte buffer containing the AECHAR string "Smith", and the parameters would be populated as follows:

* `pDBFieldName = AEEDBFIELD_LASTNAME`
* `pDBFieldType = AEEDB_FT_STRING`
* `pDBFieldLen = WStrlen(wStr) * sizeof(AECHAR) = 6 * 2 = 12 (0 terminated).`

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_GetFieldDWord()

Description:

This function does an IDBRECORD_GetField() operation on the specified database record, and checks the returned field type. If the field type is AEE_FT_DWORD, this function sets pdwRet to point to the buffer containing the dword.

Prototype:

```
boolean IDBRECORD_GetFieldDWord( IDBRecord * pIDBRecord, dword * pdwRet)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object whose field is requested
pdwRet	Pointer to dword returned by this function

Return Value:

TRUE	If the field is of type AEE_FT_DWORD
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_GetFieldString()

Description:

This function returns a AECHAR * string for a field of that type.

Prototype:

```
AECHAR * IDBRECORD_GetFieldString( IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object whose field is requested

Return Value:

pointer	To a string containing the requested field, If successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_GetFieldWord()

Description:

This function does an IDBRECORD_GetField() operation on the specified database record, and checks the returned field type. If the field type is AEE_FT_WORD, this function sets pwRet to point to the buffer containing the word.

Prototype:

```
boolean IDBRECORD_GetFieldWord( IDBRecord * pIDBRecord, word * pwRet)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object whose field is requested
pwRet	Pointer to word returned by this function

Return Value:

TRUE	If the field is of type AEE_FT_WORD
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_GetID()

Description:

This function gets the ID of the specified record.

Prototype:

```
uint16 IDBRECORD_GetID( IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object whose ID is requested

Return Value:

ID	Of the record specified, if successful
AEE_DB_ENULLREC	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_NextField()

Description:

This function sets the field pointer to the next (or first if current position is -1) field in the record. Returns the type, name and the data length of the field.

Prototype:

```
AEEDBFieldType IDBRECORD_NextField( IDBRecord * pIDBRecord, AEEDBFieldName
* pName int16 * pnLen)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object whose next field is requested
pName	Pointer to the next field
pnLen	Pointer to length of the next field

Return Value:

AEEDBFieldType structure containing field type	If successful
AEEDB_FT_NONE	If unsuccessful

Comments:

If the (specified or default) field pointer points to the last field, an invalid field type is returned.

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_Release()

Description:

This function decrements the reference count for the database record object. If the reference count reaches 0 (zero), the database record is freed from the memory.

Prototype:

```
uint32 IDBRECORD_Release(IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object object whose reference count needs to be decremented

Return Value:

Updated reference count of the object

Comments:

The object is freed and is no longer valid if 0 (zero) is returned.

Side Effects:

None

See Also:

IDBRECORD_AddRef()
Return to the List of functions

IDBRECORD_Remove()

Description:

This function removes a record from the database and frees the IDBRecord object.

Prototype:

```
int IDBRECORD_Remove( IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord which needs to be removed

Return Value:

SUCCESS	If database record was successfully removed
AEE_DB_EBADREC	If the record to be removed was in a bad state
AEE_DB_EBADSTATE	If the database is in a bad state

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_Reset()

Description:

This function makes the first field of pIDBRecord the current field.

Prototype:

```
void IDBRECORD_Reset(IDBRecord * pIDBRecord)
```

Parameters:

pIDBRecord Pointer to the IDBRecord Interface object which needs to be reset

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDBRECORD_Update()

Description:

This function updates a record given a new set of field values.

Prototype:

```
int IDBRECORD_Update( IDBRecord * pIDBRecord, AEEDBField * pDBFields, int iNumFields)
```

Parameters:

pIDBRecord	Pointer to the IDBRecord Interface object which needs to be updated
pDBFields	Pointer to the new set of field values
iNumFields	Number of fields in the new set

Return Value:

SUCCESS	If database record was successfully updated
ENOMEMORY	If there was not enough memory for this operation
AEE_DB_ENULLFIELD	If pDBFields is NULL
AEE_DB_EBADFIELDNUM	If iNumFields < 0

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDialog Interface

The IDialog Interface consists of two functions that operate on dialogs created using the functions in the IShell Interface (refer to the description of the IShell interface for more details about its dialog-related functions). IDIALOG_GetControl() is used to obtain interface pointers to the date, menu, text and time controls that make up the dialog. You can use these pointers to modify the properties and rectangles of the controls, or to retrieve the data the user has entered or selected in each control. IDIALOG_SetFocus() specifies which control in a multicontrol currently has the focus (this control receives keypad input from the user). Many controls generate control tabbing events when the user presses the LEFT and RIGHT arrow keys. Dialogs use these control tabbing events to enable the user to navigate between controls. IDIALOG_SetFocus() can be used with controls that do not support control tabbing (for example, SoftKey menus), or to designate the control that has focus initially.

To use IDIALOG_GetControl() and IDIALOG_SetFocus()

1. Create the controls in your dialog using the BREW Resource Editor or by populating the required dialog data structures in your code.

2. Call ISHELL_CreateDialog() to create the dialog and display it on the screen.

3. Call ISHELL_GetActiveDialog() to obtain an IDialog Interface pointer to the dialog created in step 2 (The ISHELL_CreateDialog() does not return such a pointer).

4. Call IDIALOG_GetControl() to access the controls in the dialog, supplying as input the interface pointer obtained in step 3 and the control IDs you specified when you created the controls in step 1. You can use this function immediately after the dialog has been created to customize the appearance and properties of its controls.

5. Call IDIALOG_SetFocus() as needed while the dialog is active to allow the user to select the control in which data is to be selected or entered. For example, if your dialog uses a SoftKey menu, you can provide a menu item that allows the user to return to a previous control to change the data entered.

6. Call IDIALOG_GetControl() when data entry is complete to obtain the values the user has entered or selected in each control.

7. Call ISHELL_EndDialog() to terminate the dialog.

List of functions

Functions in this interface include:

IDIALOG_AddRef()

IDIALOG_GetControl()

IDIALOG_Release()

IDIALOG_SetEventHandler()

IDIALOG_SetFocus()

Return to the Contents

IDIALOG_AddRef()

Description:

This function increments the reference count of the IDialog Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IDIALOG_Release().

Prototype:

```
uint32 IDIALOG_AddRef(IDialog * pIDialog)
```

Parameters:

pIDialog	Pointer to the IDialog Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IDIALOG_Release()
Return to the List of functions

IDIALOG_GetControl()

Description:

This function retrieves the IControl pointer for the control associated with the specified identifier.

Prototype:

```
IControl * IDIALOG_GetControl(IDialog * pIDialog, int16 wID)
```

Parameters:

pIDialog	Pointer to the IDialog Interface object
wID	ID of the control

Return Value:

IControl *	If successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateDialog()
IDIALOG_SetFocus()
Return to the List of functions

IDIALOG_Release()

Description:

This function decrements the reference count of the IDialog Interface object. The object is freed from memory and is no longer valid once it's reference count reaches 0 (zero).

Prototype:

```
uint32 IDIALOG_Release(IDialog * pIDialog)
```

Parameters:

pIDialog	Pointer to the IDialog Interface object

Return Value:

Decremented reference count for the object

Comments:

The object is freed and is no longer valid if 0 (zero) is returned.

Side Effects:

None

See Also:

IDIALOG_AddRef()
Return to the List of functions

IDIALOG_SetEventHandler()

Description:

Sets or resets the event handler for a dialog. This function can be used to select an alternate event callback for application events sent by a dialog to an application.

Prototype:

```
void IDIALOG_SetEventHandler (IDialog * pIDialog, PFNAEEEVENT pfn, void *
pUser);
```

Parameters:

pIDialog	Pointer to the IDialog Interface object
pfn	Pointer to the event callback function
pUser	User data pointer sent as first argument to event handler (the other three parameters are the event code and the single-word and double-word data associated with the event).

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateDialog()
IDIALOG_SetFocus()
Return to the List of functions

IDIALOG_SetFocus()

Description:

This function sets the active control focus to the control associated with the identifier specified.

Prototype:

```
int16 IDIALOG_SetFocus(IDialog * pIDialog, int16 wID)
```

Parameters:

pIDialog	Pointer to the IDialog Interface object
wID	ID of the control

Return Value:

Identifier of the control that had focus

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateDialog()
Return to the List of functions

IDisplay Interface

The IDisplay Interface functions draw text, bitmaps, and simple geometric shapes such as straight lines and rectangles on the device display (you use the IGraphics Interface functions to draw more complex lines and shapes). Because the IDisplay Interface is used by all applications, an instance of IDisplay is created automatically when your application is created.

Several IDisplay functions are concerned with drawing text on the screen and obtaining information about its size so that it can be optimally positioned. IDISPLAY_DrawText() draws a text string at specified x and y coordinates on the screen; you also can specify the text's font and its vertical and horizontal alignment, as well as a clipping rectangle that limits the portion of the screen in which the text is to be drawn. IDISPLAY_MeasureText() measures the width of a text string in a particular font. This function can be used to determine if a string will fit in a given region of the screen. IDISPLAY_MeasureTextEx() is an extended version of this function that allows you to specify the horizontal space available to display the text string (this function returns the number of characters of your string that will fit in this space). IDISPLAY_GetFontMetrics() is used to return the height in pixels of text in a given font, which can be used to calculate the number of lines of text that will fit in a vertical region of the screen. IDISPLAY_GetSymbol() returns the character value (AECHAR) associated with the given symbol (many devices have special characters that are used to denote the keys on the device or an AM/PM indicator).

The geometric operations in the IDisplay Interface operate on rectangular regions of the screen. IDISPLAY_EraseRect() draws a rectangle whose coordinates are specified in an AEERect structure; you can also specify the colors of the rectangle's frame and interior.

The following functions are special cases of IDISPLAY_DrawRect() that capture some of its most common uses:

IDISPLAY_EraseRect() erases the specified rectangle (for example, fills it with the default screen background color).

IDISPLAY_EraseRgn() erases a region of the screen given by its height and width and the x and y coordinates of its top left corner (instead of the AEERect structure that is the input to IDISPLAY_EraseRect()).

IDISPLAY_FillRect() fills the given rectangle with a specified color.

IDISPLAY_FrameRect() draws a frame in the default frame color around the given rectangle, but does not modify the rectangle's interior.

IDISPLAY_FrameSolidRect() draws a frame in the default frame color around the given rectangle, and fills its interior with the default background color.

IDISPLAY_InvertRect() inverts the colors of the rectangle (this can be used to display the rectangle's current contents in reverse video).

IDISPLAY_DrawHLine() and IDISPLAY_DrawVLine() are used to draw horizontal and vertical lines on the screen. Both functions take the line's length in pixels and its starting x and y coordinates as input; the line is drawn one pixel wide and in black.

IDISPLAY_SetColor() specifies the color that is used to display a given item on the screen. Items whose colors can be specified include text, lines and backgrounds. Colors are specified as RGB values.

IDISPLAY_BitBlt() is used to draw a bitmap image to a region of the screen. As input to this function, you specify the size and location of the rectangle in which the bitmap is displayed, a pointer to the source bitmap, the x and y coordinates of the portion of the source bitmap that is to be drawn, and the raster operation that is used to combine the pixels of the source bitmap and the destination region.

IDISPLAY_Backlight() turns the device's backlight (if any) on and off.

IDISPLAY_SetAnnunciators() turns the device's annunciators on and off. Annunciators are small images that appear on the device screen to signal certain conditions such as the presence of voice mail or a newly arrived SMS message (the set of supported annunciators is different for each device).

The screen-drawing operations mentioned above do not take effect until the screen has been updated. IDISPLAY_Update() causes the updating of the screen to take place. To minimize unnecessary screen re-drawing, you typically call IDISPLAY_Update() once after multiple drawing operations. This function places the update in a queue of tasks that is to be performed at a later time; the function IDISPLAY_UpdateEx() lets you force the screen update to occur immediately.

The use of the functions in the IDisplay Interface typically consists of the following steps:

1 Call ISHELL_CreateInstance() to obtain an instance of the IDisplay Interface (you can also use the instance pointed to by the **m_pIDisplay** variable in your applet data structure).

2 Call IDISPLAY_EraseRect(), IDISPLAY_EraseRgn(), or one of the of the other rectangle-filling functions if necessary to clear the region of the screen in which your drawing operations occurs.

3 Use the IDisplay drawing operations to draw the text, lines, and rectangles and bitmaps you need on the screen.

4 Call IDISPLAY_Update() or IDISPLAY_UpdateEx() to cause the drawing operations to take effect.

5 Call IDISPLAY_Release() to free the IDisplay Interface when you no longer need it (the instance pointed to in your applet data structure is freed automatically when your applet terminates).

List of functions

Functions in this interface include:

IDISPLAY_AddRef()

IDISPLAY_Backlight()

IDISPLAY_BitBlt()

IDISPLAY_ClearScreen()

IDISPLAY_DrawFrame()

IDISPLAY_DrawHLine()

IDISPLAY_DrawRect()

IDISPLAY_DrawText()

IDISPLAY_DrawVLine()

IDISPLAY_EraseRect()

IDISPLAY_EraseRgn()

IDISPLAY_FillRect()

IDISPLAY_FrameButton()

IDISPLAY_FrameRect()

IDISPLAY_FrameSolidRect()

IDISPLAY_GetFontMetrics()

IDISPLAY_GetSymbol()

IDISPLAY_InvertRect()

IDISPLAY_MeasureText()

IDISPLAY_MeasureTextEx()

IDISPLAY_Release()

IDISPLAY_SetAnnunciators()

IDISPLAY_SetColor()

IDISPLAY_Update()

IDISPLAY_UpdateEx()

Return to the Contents

IDISPLAY_AddRef()

Description:

This function increments the reference count of the IDisplay Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 IDISPLAY_AddRef(IDisplay * pIDisplay)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IDISPLAY_Release()
Return to the List of functions

IDISPLAY_Backlight()

Description:

This function turns the backlight of the device on or off depending on the parameter bOn. The behavior of the backlight is completely dependent on the specific device. When this function is used to turn the backlight on, the device manufacturer decides how long it remains on before it is turned off again. Similarly, when this function is used to turn off the backlight, the device manufacturer of the device decides how long it remains off before it is turned on again.

Prototype:

```
void IDISPLAY_Backlight(IDisplay * pIDisplay,boolean bOn)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for controlling the backlight of the device
bOn	If TRUE, the backlight is turned on If FALSE, the back light is turned off

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_BitBlt()

Description:

This function performs a bit-block transfer of the data corresponding to a rectangle of pixels from the specified source bitmap into the given Display area. After calling this function, IDISPLAY_Update() must be called to update the screen. If there is a sequence of drawing operations being performed, it is sufficient to call IDISPLAY_Update() once after all the drawing is done.

Prototype:

```
void IDISPLAY_BitBlt(IDisplay * pIDisplay, int xDest, int yDest, int cxDest,
int cyDest, const void * pbmSource, int xSrc, int ySrc, AEE_RasterOp
dwRopCode)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object into which the bit-block transfer needs to be done
xDest	Specifies the x-coordinates of the upper left corner of the destination rectangular area
yDest	Specifies the y-coordinates of the upper left corner of the destination rectangular area
cxDest	Specifies the width of the destination rectangle. If this is less than zero(0) or is greater than the width of the source bitmap (pmSource) , this parameter is taken to be equal to the width of the source bitmap.
cyDest	Specifies the height of the destination rectangle. If this is less than zero(0) or is greater than the height of the source bitmap (pmSource) , this parameter is taken to be equal to the height of the source bitmap.
pbmSource	Pointer to a structure containing the source bitmap. The data being pointed is without the AEE header. In the case of .BMP format, the data can start from BITMAPFILEHEADER.
xSrc	Specifies the x-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin
ySrc	Specifies the y-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin
dwRopCode	Specifies the Raster operation that must be used while doing the bit-block transfer

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IDISPLAY_Update()
AEERasterOp
Return to the List of functions

IDISPLAY_ClearScreen()

Description:

This function clears the whole device screen.

Prototype:

```
void IDISPLAY_ClearScreen(IDisplay * pIDisplay)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also

None
Return to the List of functions

IDISPLAY_DrawFrame()

Description:

This function draws complex frames based upon the color resolution of the system. It allows single and 3D frames to be drawn. Passing a valid clrFill to the routine fills the inside of the specified rectangle with the specified color. The specified rectangle is adjusted by the size of the resulting operation. This allows the routine to be called and have it automatically adjust the rectangle so that subsequent operations (such as text drawing) are offset by the proper amount.

Prototype:

```
int IDISPLAY_DrawFrame(IDisplay * pIDisplay, AEERect * prc, AEEFrameType ft,
RGBVAL clrFill)
```

Parameters:

pIDisplay	[in]	Pointer to the IDisplay Interface object
prc	[in/out]	Pointer to the source rectangle. If this is NULL, no frame is drawn
ft	[in]	Frame type
clrFill	[in]	Fill type for the inside of the frame

Return Value:

None

Comments:

None

Side Effects:

None

See Also:
AEEFrameType
Return to the List of functions

IDISPLAY_DrawHLine()

Description:

This function draws a horizontal line of the given length, starting from the given point.

Prototype:

```
void IDISPLAY_DrawHLine(IDisplay * pIDisplay,int16 x,int16 y,int16 len)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used to draw the horizontal line
x	X coordinate of the starting point of the line
y	Y coordinate of the starting point of the line
len	Length of the line

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_DrawRect()

Description:

This function draws the given rectangle using the specified color and flags. After calling this function, IDISPLAY_Update() must be called to update the screen. If there is a sequence of drawing operations being performed, it is sufficient to call IDISPLAY_Update() once after all the drawing is done.

Prototype:

```
void IDISPLAY_DrawRect(IDisplay * pIDisplay,const AEERect * pRect, RGBVAL
clrFrame, RGBVAL clrFill, uint32 dwFlags)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for drawing the rectangle
pRect	Pointer to a AEERect structure that defines the coordinates of the rectangle to be drawn. These coordinates are in terms of screen coordinates with the left, top of the screen being 0,0.
clrFrame	This specifies the color to be used for drawing the frame (outer borders) of the rectangle. This parameter is used only if **dwFlags** contains the flag IDF_RECT_FRAME.
clrFill	This specifies the color to be used for filling the rectangle. This parameter is used only if **dwFlags** contains the flag IDF_RECT_FILL.
dwFlags	Specifies the flags to be used for drawing the rectangle. This can be a logical OR of one or more of the following flags: IDF_RECT_FRAME : Draw the outer borders of the rectangle only IDF_RECT_FILL : Fill the rectangle with **clrFill** color IDF_RECT_INVERT: Invert the contents of the specified rectangle. When this flag is set, **clrFrame** and **clrFill** parameters won't matter.

Return Value:

None

Comments:

If **pRect** is NULL and **dwFlags** contains IDF_RECT_FILL, this function clears the entire screen using **clrFill**. If **pRect** is NULL without IDF_RECT_FILL flag, this function treats **pRect** as an empty rectangle.

Side Effects:

None

See Also:

IDISPLAY_Update()
AEERect
Return to the List of functions

IDISPLAY_DrawText()

Description:

This function draws the given text at the given location, using the given font and bounds the text to the clipping rectangle. After calling this function, IDISPLAY_Update() must be called to update the screen. If there is a sequence of drawing operations being performed, it is sufficient to call IDISPLAY_Update() once after all the drawing is done.

Prototype:

```
int IDISPLAY_DrawText(IDisplay * pIDisplay,AEEFont Font, const AECHAR *
pcText, int nChars, int x, int y, const AEERect * prcBackground, uint32
dwFlags)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for drawing the text
Font	Specifies the font that needs to be used for drawing the text
pcText	Contains the String that needs to be drawn
nChars	Specifies the number of characters in pcText. If this is -1, the length is automatically computed by this function.
x	Specifies the x coordinate of the location on the screen where the text needs to be drawn. The upper, left corner of the screen is treated as [0,0]. This parameter doesn't matter if horizontal alignment flag is set.
y	Specifies the x coordinate of the location on the screen where the text needs to be drawn. The upper, left corner of the screen is treated as [0,0]. This parameter doesn't matter if vertical alignment flag is set.

prcBackground	Specifies the coordinates of the clipping rectangle. If this is NULL, the whole screen is taken as the clipping rectangle. No text is drawn outside this clipping rectangle. Clipping of characters (if necessary) is done starting from the right of the string. If any of the Flags for the Rectangle is specified, then this rect is also used as filling.
dwFlags	Specifies the flags that can be used for drawing the screen. This can be a logical OR of one of the items selected from each of the following entries: a. One of the horizontal alignment flags (IDF_ALIGN_LEFT, IDF_ALIGN_CENTER, IDF_ALIGN_RIGHT) b. One of the vertical alignment flags (IDF_ALIGN_TOP,IDF_ALIGN_MIDDLE, IDF_ALIGN_BOTTOM) c. One of the text format flags (IDF_TEXT_UNDERLINE, IDF_TEXT_INVERTED) d. One of the rect format flags (IDF_RECT_FRAME, IDF_RECT_FILL, IDF_RECT_INVERT) --- these flags works on the prcBackground rectangle, using the CLR_USER_BACKGROUND as the fill color and CLR_USER_FRAME as the frame color. If any of the Flags for the Rectangle is specified, then this rect is also used as filling.

If no alignment flags are specified, the position of the text is determined by parameter x and y. The text blocks everything behind it by default. To avoid this effect, use IDF_TEXT_TRANSPARENT flag. Currently in BREW Emulator IDF_ALIGN_SPREAD and IDF_ALIGN_FILL are not supported.

Example:

The combination IDF_ALIGN_CENTER | IDF_ALIGN_TOP | IDF_TEXT_UNDERLINE draws underlined text at the top (vertical) center (horizontal) of the clipping rectangle.

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IDISPLAY_Update()
AEEFont
AEERect
Return to the List of functions

IDISPLAY_DrawVLine()

Description:

This function draws a vertical line of the given length, starting from the given point.

Prototype:

```
void IDISPLAY_DrawVLine(IDisplay * pIDisplay,int16 x,int16 y,int16 len)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used to draw the vertical line
x	X coordinate of the starting point of the line
y	Y coordinate of the starting point of the line
len	Length of the line

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_EraseRect()

Description:

This function fills the given rectangle with the default background color (for example, the color associated with the item CLR_USER_BACKGROUND).

Prototype:

```
void IDISPLAY_EraseRect(IDisplay * pIDisplay, AEERect * pRect)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to erase the rectangle
pRect	Valid pointer to a rectangle whose color needs to be inverted

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IDISPLAY_EraseRgn()

Description:

This function fills the region enclosed by the given coordinates with the default background color (for example, the color associated for the item CLR_USER_BACKGROUND).

Prototype:

```
void IDISPLAY_EraseRgn(IDisplay * pIDisplay, int16 x, int16 y, uint16 cx,
uint16 cy)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to fill the region
x	X coordinate of the top, left corner of the region
y	Y coordinate of the top, left corner of the region
cx	Width of the region
cy	Height of the region

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_FillRect()

Description:

This function draws a rectangle and fills it with a specified color.

Prototype:

```
void IDISPLAY_FillRect(IDisplay * pIDisplay,AEERect * pRect,RGBVAL clrFill)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to fill the rectangle
pRect	Valid pointer to a rectangle that needs to be filled with the specified color
clrFill	Specifies the color to be used to fill the rectangle

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_FrameButton()

Description:

This function draws a 3D framed button based upon the color resolution of the system. The function draws a lowered button when **bPressed** is TRUE, and a raised button when the **bPressed** is FALSE. This function draws complex frames based upon the color resolution of the system. It allows single and 3D frames to be drawn. Passing a valid **clrFill** to the routine fills the inside of the specified rectangle with the specified color. The specified rectangle is adjusted by the size of the resulting operation. This allows the routine to be called and have it automatically adjust the rectangle so that subsequent operations (such as text drawing) are offset by the proper amount.

Prototype:

```
void IDISPLAY_FrameButton(IDisplay * pIDisplay, AEERect * prc, boolean
bPressed, RGBVAL clrFill)
```

Parameters:

pIDisplay	[in]	Pointer to the IDisplay Interface object
prc	[in/out]	Pointer to the source rectangle
bPressed	[in]	Button pressed/raised indicator
clrFill	[in]	Fill type for the inside of the frame

Return Value:

The input rectangle (prc) is adjusted by this call.

Comments:

None

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IDISPLAY_FrameRect()

Description:

This function draws the borders of a rectangle. The color used for drawing the borders is the current color assigned to the item CLR_USER_FRAME.

Prototype:

```
void IDISPLAY_FrameRect(IDisplay * pIDisplay, AEERect * pRect)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to draw the rectangle
pRect	Valid pointer to a rectangle whose borders need to be drawn

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IDISPLAY_FrameSolidRect()

Description:

This function draws the borders of a rectangle and fills it with a color. The color used for drawing the borders is the current color assigned to the item CLR_USER_FRAME. The color used for filling the rectangle is the current color assigned to the item CLR_USER_BACKGROUND.

Prototype:

```
void IDISPLAY_FrameSolidRect(IDisplay * pIDisplay, AEERect * pRect)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to draw the rectangle
pRect	Valid pointer to a rectangle that needs to be drawn and filled

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IDISPLAY_GetFontMetrics()

Description:

This function retrieves information about the specified font. It retrieves information about the ascent and descent values for the specified font.

NOTE: `ascent + descent = total character height`, but does not include any leading spaces.

Prototype:

```
int IDISPLAY_GetFontMetrics(IDisplay * pIDisplay, AEEFont Font,
int * pnAscent, int * pnDescent)
```

Parameters:

pIDisplay	[in]	Pointer to the IDisplay Interface object whose font metrics is to be retrieved
Font	[in]	Specifies the font type for which the Ascent and Descent information is to be retrieved
pnAscent	[in/out]	On input, this must be a valid pointer to int. On function return, it points to an integer denoting the Ascent value for the specified font. NULL pointer is OK and in this case it remains NULL on return.
pnDescent	[in/out]	On input, this must be a valid pointer to int. On function return, it points to an integer denoting the Descent value for the specified font. NULL pointer is OK and in this case it remains NULL on return.

Return Value:

the character height for the specified font	If successful. This is the sum of the Ascent and Descent values for the specified font.
EFAILED	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

AEEFont
Return to the List of functions

IDISPLAY_GetSymbol()

Description:

This function returns the AECHAR value corresponding to the specified symbol value.

Prototype:

```
AECHAR IDISPLAY_GetSymbol(IDisplay * pIDisplay, AEESymbol sym, AEEFont fnt)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for changing the color of an user item
sym	Requested symbol
fnt	Requested font

Return Value:

The AECHAR associated with the specified symbol.

Comments:

None

Side Effects:

None

See Also:

AEEFont
AEESymbol
Return to the List of functions

IDISPLAY_InvertRect()

Description:

This function inverts the color in the given Rectangle. It inverts the color and re-fills the rectangle with that inverted color pattern.

Prototype:

```
void IDISPLAY_InvertRect(IDisplay * pIDisplay, AEERect * pRect)
```

Parameters:

pIDisplay	Pointer to IDisplay Interface object to be used to invert the rectangle
pRect	Valid pointer to a rectangle whose color needs to be inverted

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IDISPLAY_MeasureText()

Description:

This function measures the width of a given text string if drawn using the specified font.

Prototype:

```
int IDISPLAY_MeasureText(IDisplay * pIDisplay, AEEFont Font, const AECHAR *
pcText)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for measuring the width of the given text
Font	Specifies the font that needs to be used for measuring the width of the given text. This function measures the width of a given text string if drawn using this font.
pcText	Pointer to a string for which the measurement needs to be done

Return Value:

Number of pixels required to draw the characters in the string pcText	If successful,
0 (zero)	If otherwise

Comments:

None

Side Effects:

None

See Also:

AEEFont
Return to the List of functions

IDISPLAY_MeasureTextEx()

Description:

This function measures the width of a given text string if drawn using the specified font. The return is the actual pixel width of the string.

Prototype:

```
int IDISPLAY_MeasureText(IDisplay * pIDisplay, AEEFont Font, const AECHAR *
pcText, int nChars, int nMaxWidth, int * pnFits)
```

Parameters:

pIDisplay	[in]	Pointer to the IDisplay Interface object to be used for measuring the width of the given text
Font	[in]	Specifies the font that needs to be used for measuring the width of the given text. This function measures the width of a given text string if drawn using this font.
pcText	[in]	Pointer to a string for which the measurement needs to be done.
nChars	[in]	Specifies the number of characters in **pcText**. If this is -1, the length is automatically computed by this function.
nMaxWidth	[in]	Specifies the maximum available pixel width that can be used for drawing the text. If **nMaxWidth** is set to -1, then the **nChars** part of text string is measured, and * **pnFits** returned always is the entire text length. If **nMaxWidth > 0**, then it represents the maximum available pixel width for the text to be rendered within; this function computes the maximum number of characters which actually fit within this constraint and return this number of characters in * **pnFits** and the actual pixel width of these characters in * **pnWidth**.
pnFits	[in/out]	On input, this must be a valid pointer to an integer. On return, this pointer points to an integer that denotes the number of characters that fit the given pixel width specified by **nMaxWidth**. If **nMaxWidth** is 0 (zero), then this parameter points to an integer denoting the entire length of the given string **pcText**.

Return Value:

Pixels	If successful, returns the number of pixels required to draw the characters in the string **pcText**. The number of characters that can be drawn in this width is contained in * **pnFits**.
0 (zero)	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

AEEFont
Return to the List of functions

IDISPLAY_Release()

Description:

This function decrements the reference count for the IDisplay object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IDisplay_Release(IDisplay * pIDisplay)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object whose reference count needs to be decremented

Return Value:

The updated reference count.

Comments:

None

Side Effects:

None

See Also:

IDISPLAY_AddRef()
Return to the List of functions

IDISPLAY_SetAnnunciators()

Description:

This function turns the specified annunciators on (or off). The support and behavior of this function is totally dependent on the specific device.

The complete list of annunciators is:

ANNUN_MSG //Voice mail
ANNUN_NET_MSG //Net Message
ANNUN_ALARMCLOCK //Alarm Clock
ANNUN_NET_LOCK //Device Locked
ANNUN_STOPWATCH //Stop Watch
ANNUN_COUNTDOWN //Count Down clock
ANNUN_SILENCEALL //Ringer Off

A device may support few or all of the above listed Annunciators.

Prototype:

```
void IDISPLAY_SetAnnunciators(IDisplay * pIDisplay, uint16 wVal, uint16
wMask)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for enabling/disabling annunciators
wVal	Specifies that set of annunciators that this function is trying to enable or disable. This is a logical OR of one or more of the annunciator flags defined above.
wMask	For each annunciator flag contained in **wVal**, the corresponding bit here specifies if that annunciator needs to be turned on or off.

Return Value:

None

Comments:

To turn the ANNUN_MSG and ANUN_NET_MSG annunciators ON and to turn the ANNUN_COUNTDOWN annunciator off, use:

```
uint16 wVal = 0;
uint16 wMask = 0;
wVal |= ANNUN_MSG | ANNUN_NET_MSG | ANNUN_COUNTDOWN;
wMask |= ANNUN_MSG | ANNUN_NET_MSG;
IDISPLAY_SetAnnunciators(pIDisplay,wVal,wMask) ;
```

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_SetColor()

Description:

This function sets the color of the specified user Item. User items are those listed in the top of the AEEClrItem enumeration. These items have a prefix CLR_USER. The color of the system-items (for example, the items in the bottom of the AEEClrItem enumeration which have a prefix CLR_SYS) cannot be changed. The color of an item can either be set to a specific RGB value or it can be set to the color of another item in the AEEClrItem enumeration. For example:

To set the color of the item CLR_USER_TEXT to be the same as the color of the item CLR_SYS_TITLE_TEXT, use:

```
IDISPLAY_SetColor(pIDisplay,CLR_USER_TEXT,CLR_SYS_TITLE_TEXT)
```

To set it to white, use

```
IDISPLAY_SetColor(pIDisplay,CLR_USER_TEXT, RGB_WHITE)
```

To set it to any specific RGB value use

```
IDISPLAY_SetColor(pIDisplay,CLR_USER_TEXT,
MAKE_RGB(0x40,0x30,0x50) )
```

Prototype:

```
RGBVAL IDISPLAY_SetColor(IDisplay * pIDisplay, AEEClrItem item, RGBVAL rgb)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object to be used for changing the color of an user item
item	Specifies the user item whose color needs to be changed. This has to be one of the items in the AEEClrItem enumeration which have a prefix CLR_USER.
rgb	Specifies the new color to be associated with the item mentioned in “item” parameter. This can be either be an index of an item in the AEEClrItem enumeration or it can be a RGB value.

Return Value:

the previous color associated with "item"	If successful
RGB_NONE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

AEEClrItem
Return to the List of functions

IDISPLAY_Update()

Description:

This function updates the screen. The update message is posted in the queue of the user interface task, thereby allowing all the drawings to be done before updating the screen.

Prototype:

```
void IDISPLAY_Update(IDisplay * pIDisplay)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object that needs to be updated

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IDISPLAY_UpdateEx()

Description:

This function updates the screen. If the bDefer flag is set to TRUE, the screen is refreshed (updated) immediately. If the bDefer flag is set to FALSE, the update message is posted in the queue of the user interface task, thereby allowing all the drawings to be done before updating the screen.

Prototype:

```
void IDISPLAY_UpdateEx(IDisplay * pIDisplay, boolean bDefer)
```

Parameters:

pIDisplay	Pointer to the IDisplay Interface object that needs to be updated
bDefer	Boolean flag to indicate whether the screen needs to be updated immediately or deferred

Return Value:

None

Comments:

On BREW Emulator the **bDefer** parameter makes no difference. The screen is always updated immediately.

Side Effects:

None

See Also:

None
Return to the List of functions

IFile Interface

The IFile Interface functions allow you to read and modify the contents of files created with the IFileMgr Interface. To obtain an instance of the IFile Interface for a file, you call IFILEMGR_OpenFile() for that file. You then use the IFile Interface pointer returned by IFILEMGR_OpenFile() to access that file with the operations described later in this section. When you have completed access to the file, you call IFILE_Release() to close it.

> **CAUTION:** Your application must have a privilege level of **File** or **All** to be able to modify files with IFILE_Write() or IFILE_Truncate(). Your application must have a privilege level of **Shared** or **All** to invoke these functions on files in the shared application directory.

The function IFILE_GetInfo() returns information about an open file, including its name, size and creation timestamp (if the file is not open, you can obtain the same information by calling IFILEMGR_GetInfo() with the name of the file as input).

IFILE_Read() reads a specified number of bytes from the file and copies them into a character buffer in your application. IFILE_Write() writes a number of bytes to the file from a character buffer that contains the data to be written. All read and write operations are with respect to the current file cursor, which determines which byte of the file is to be read or written next. Initially, the cursor points to the first byte of the file (unless it was opened for appending, in which case the cursor points to a position just past the last byte of the file). Each file read and write sets the cursor to the position just past the last byte read or written. You can use IFILE_Seek() to set the cursor to a particular position in the file and to obtain the current value of the cursor, which allows you to read from and write to any position in the file.

The function IFILE_Truncate() allows you to reduce the size of a file, which discards the contents of the truncated portion at the end of the file.

To use functions in the IFile Interface,

1. Call ISHELL_CreateInstance() if necessary to obtain an instance of the IFileMgr Interface.
2. If the file you wish to access does not exist yet,
   - Call IFILEMGR_OpenFile() to create the file, supplying the file's name as input. Upon creation, the file is open for reading and writing.
   - Otherwise, call IFILEMGR_OpenFile() to open the file for reading, reading and writing, or appending, based on the type of access you need.

3 Using the IFile pointer obtained in step 2, call IFILE_Seek() if necessary to position the file cursor, and then call IFILE_Read() and IFILE_Write() to read from and write to the file as needed.

4 Call IFILE_GetInfo() if you need to obtain information about the file, and call IFILE_Truncate() to truncate it.

5 Call IFILE_Release() to close the file when you have completed accessing it.

List of functions

Functions in this interface include:

IFILE_AddRef()

IFILE_Cancel()

IFILE_GetInfo()

IFILE_Read()

IFILE_Readable()

IFILE_Release()

IFILE_Seek()

IFILE_Truncate()

IFILE_Write()

Return to the Contents

IFILE_AddRef()

Description:

This function increments the reference count of the IFile Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IFILE_Release().

Prototype:

```
uint32 IFILE_AddRef(IFile * pIFile)
```

Parameters:

pIFile	Pointer to the IFile Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IFILE_Release()
Return to the List of functions

IFILE_Cancel()

Description:

This function cancels the callback registered with this IFile interface object with IFILE_Readable().

Prototype:

```
void IFILE_Cancel(IFile * pIFile, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIFile	Pointer to IFile interface object for which the registered callback function needs to be cancelled.
pfn	Address of the callback function.
pUser	User defiend data that will be passed to the callback function when it is invoked.

Return Value:

None

Comments:

This function doesn't care for the values of **pfn** and **pUser** parameters.

Side Effects:

None

See Also:

IFILE_Readable()
Return to the List of functions

IFILE_GetInfo()

Description:

This function gets the file creation date, file size, file name and file attributes of the file pointed to by the IFile Interface object.

Prototype:

```
int IFILE_GetInfo(IFile * pIFile, FileInfo * pInfo)
```

Parameters:

pIFile	[in]	Pointer to IFile Interface object
pInfo	[out]	placeholder for file information

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

The file information is returned in FileInfo structure provided by **pInfo**.

Side Effects:

None

See Also:

IFILEMGR_GetInfo()
FileInfo
Return to the List of functions

IFILE_Read()

Description:

This function reads a specified number of bytes from an open file. The read operation is non-blocking.

Prototype:

```
uint32 IFILE_Read(IFile * pIFile, void * pBuffer, uint32 dwCount)
```

Parameters:

pIFile	[in]	Pointer to IFile Interface object
pBuffer	[out]	Buffer from which the file data is read
dwCount	[in]	Number of bytes to be read

Return Value:

Number of bytes read	If successful
0 (zero)	If unsuccessful

Comments:

To read data from a file, the file needs to be open. See IFILEMGR_OpenFile() function of the IFileMgr Interface for more details on opening a file. The bytes are read, starting from the location of the file pointer in the IFile Interface object. The file pointer within a file can be relocated using the IFILE_Seek() function.

Side Effects:

When the read operation is over, the file pointer of the IFile Interface object points to the end of the block of bytes that were read.

See Also:

IFILEMGR_OpenFile(),
IFILE_Write(),
IFILE_Seek()
Return to the List of functions

IFILE_Readable()

Description:

This function is used for registering a callback function which tries to read from the file at a later time.

Prototype:

```
void IFILE_Readable(IFile * pIFile, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIFile	Pointer to IFile interface object for which the callback function needs to be registered.
pfn	Address of the callback function.
pUser	User defiend data that will be passed to the callback function when it is invoked.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IFILE_Cancel()
Return to the List of functions

IFILE_Release()

Description:

This function decrements the reference count of the IFile Interface object. If the reference count reaches 0 (zero), the file associated with the IFile Interface object closes.

Prototype:

```
uint32 IFILE_Release(IFile * pIFile)
```

Parameters:

pIFile	Pointer to the IFile Interface object

Return Value:

The updated reference count.

Comments:

None

Side Effects:

None

See Also:

IFILE_AddRef()
IFILEMGR_OpenFile()
Return to the List of functions

IFILE_Seek()

Description:

This function moves the file pointer of the IFile Interface object a given distance from a specified origin.

Prototype:

```
uint32 IFILE_Seek(IFile * pIFile, FileSeekType seekType, int32 moveDistance)
```

Parameters:

pIFile	Pointer to IFile Interface object
seekType	[_SEEK_CURRENT|_SEEK_START|_SEEK_END]
moveDistance	Distance to move

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

In the special case when the seek type is _SEEK_CURRENT and the move distance is 0 (zero), the position of the current file pointer is returned.

Comments:

The specified origin (FileSeekType) can take three values:

_SEEK_CURRENT
_SEEK_START
_SEEK_END

The move distance can be specified in positive or negative directions (values). For example, negative direction is used when seeking from the end of the file (_SEEK_END). If the seek type is _SEEK_CURRENT and move distance is 0 (zero), this acts as tell operation and returns the current position of the file pointer. Otherwise file pointer is moved by given distance from specified origin. If the file was opened in the _OFM_READ mode, this operation succeeds if the move distance from the specified origin is within the boundaries of the file. See IFILEMGR_OpenFile() function description to find out more on the file open modes. If the file was opened in the _OFM_READWRITE mode and the

move distance points to location before the beginning of the file, this operation fails. If the move distance from the specified origin is past the end of the file, the file size is extended to the point of the move distance (from the seek origin).

For example:

A file is created in READWRITE mode, and the file size if 40 bytes. The IFILE_Seek() function is used with seek type of _SEEK_START, and a move distance of 120. This causes the file size to increase to 120 bytes, and the file pointer points to the end of the file.

Side Effects:

None

See Also:

IFILEMGR_OpenFile()
Return to the List of functions

IFILE_Truncate()

Description:

This function truncates the file specified by the IFile Interface object to position specified by the **truncate_pos** parameter.

Prototype:

```
int IFILE_Truncate(IFile * pIFile, uint32 truncate_pos)
```

Parameters:

pIFile	Pointer to IFile Interface object
truncate_pos	Truncate position

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

To truncate a file, the file needs to be open in the read/write mode.

Side Effects:

After the file is truncated the file pointer is moved to the start position of the file.

See Also:

None
Return to the List of functions

IFILE_Write()

Description:

This function writes the specified number of bytes to an open file. The file must have been open by the IFileMgr when creating the file object. The operation is non-blocking.

Prototype:

```
uint32 IFILE_Write(IFile * pIFile, PACKED const void * pBuffer, uint32
dwCount)
```

Parameters:

pIFile	[in]	Pointer to IFile Interface object
pBuffer	[out]	Buffer into which the data is written
dwCount	[in]	Number of bytes to be written

Return Value:

Number of bytes written	If successful
0 (zero)	If unsuccessful

Comments:

To write data to a file, the file needs to be open in the read/write mode. See IFILEMGR_OpenFile() function of the IFileMgr Interface for more details on opening a file. The bytes are written, starting from the location of the file pointer in the IFile Interface object. The file pointer within a file can be relocated using the IFILE_Seek() function.

Side Effects:

When the write operation is over, the file pointer of the IFile Interface object points to the end of the block of bytes that were written.

See Also:

IFILEMGR_OpenFile(),
IFILE_Read()
Return to the List of functions

IFileMgr Interface

The IFileMgr Interface functions are used to create, remove and rename files/directories. It also provides the tools to obtain information about them. To create an instance of the IFileMgr Interface, you call ISHELL_CreateInstance() with AEECLSID_FILEMGR ClassID. IFileMgr operations can be used to access files in your application's directory. BREW also provides a shared application directory that allows files to be shared among applications. This is done by starting the target path with the name of the shared directory, which is defined by the constant AEE_SHARED_DIR in the header file AEE.h (the actual name of the shared directory is selected by the device manufacturer).

File and directory names in BREW are case-insensitive. This means that if you specify a file or directory name to IFileMgr or any other BREW file-related API, it applies the filename as lower-case. For example, if you invoke IFILEMGR_OpenFile with a file name of "Foo.bar", the file is opened as "foo.bar".

> **CAUTION:** Your application must have a privilege level of **File** or **All** to be able to create files and directories. Your application must have a privilege level of **Shared** or **All** to create files and directories in the shared application directory.

The functions IFILEMGR_EnumInit() and IFILEMGR_EnumNext() are used to enumerate all the directories or files in a given directory (you can also enumerate all the files with a given file extension). You first call IFILEMGR_EnumInit() to specify the directory in which you want to enumerate files and directories. Each subsequent call to IFILEMGR_EnumNext() provides information about one of the requested directories or files, such as its name, size, creation date and attributes. IFILEMGR_EnumNext() returns FALSE when all the specified files or directories have been enumerated. IFILEMGR_GetInfo() returns the same information for a particular file or directory specified by name (you can obtain this information for an open file by calling IFILE_GetInfo() and providing its IFile instance pointer as input). IFILEMGR_GetFreeSpace() returns the number of free bytes available in the device's file system. IFILEMGR_GetLastError() returns the error code of the error (if any) that was most recently detected by an IFileMgr function. This error code can be used to obtain more specific information about why a function failed to perform a requested task. IFILEMGR_Test() checks whether a specified file or directory exists.

IFILEMGR_MkDir() creates a new directory, specified by its name and path relative to the directory of the applet that is making the call. IFILEMGR_RmDir() removes a directory (you must first remove all the files and directories in the directory to be removed).

IFILEMGR_OpenFile() is used to create a new file, or to open an existing file for reading and/or writing. The file is specified by its name and its path relative to the applet's directory. IFILEMGR_OpenFile() returns an IFile instance pointer for the opened file. This pointer is provided as input to functions in the

IFile Interface that are used to read and write the file's contents. You close an open file by calling IFILEMGR_Release() with this pointer as its parameter. IFILEMGR_Rename() is used to rename a file. and IFILEMGR_Remove() removes a file. A file must be closed before it can be renamed or removed; If the file has been opened more than once, each open must be matched by a close before the rename or remove can succeed.

To use the functions in the IFileMgr Interface

1 Call ISHELL_CreateInstance() to create an instance of the IFileMgr Interface.

2 Call the functions listed above to obtain information about the files and directories that are present in your application's directory, or to determine the amount of free space available in the file system.

3 Call IFILEMGR_MkDir() and IFILEMGR_RmDir() to create and remove directories, and call IFILEMGR_OpenFile(), IFILEMGR_Rename(), and IFILEMGR_Remove() to create, rename and remove individual files.

To read and/or modify the contents of a file

1 Call IFILEMGR_OpenFile() to open the file for reading or writing.

2 Call functions in the IFile Interface to access the file's contents.

3 Call IFILE_Release() to close the file when you have completed accessing it.

4 Call IFILEMGR_Release() when you no longer need the IFileMgr Interface instance.

List of functions

Functions in this interface include:

IFILEMGR_AddRef()

IFILEMGR_EnumInit()

IFILEMGR_EnumNext()

IFILEMGR_GetFreeSpace()

IFILEMGR_GetInfo()

IFILEMGR_GetLastError()

IFILEMGR_MkDir()

IFILEMGR_OpenFile()

IFILEMGR_Release()

IFILEMGR_Remove()

IFILEMGR_Rename()

IFILEMGR_RmDir()

IFILEMGR_Test()

Return to the Contents

IFILEMGR_AddRef()

Description:

This function increments the reference count of the IFileMgr Interface object

Prototype:

```
uint32 IFILEMGR_AddRef(IFileMgr * pIFileMgr)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object

Return Value:

Returns the updated reference count.

Comments:

None

Side Effects:

None

See Also:

IFILEMGR_Release()
Return to the List of functions

IFILEMGR_EnumInit()

Description:

This function initializes the IFileMgr Interface object for interactively using the IFILEMGR_EnumNext() operation on the files or directories in the specified directory.

Prototype:

```
int IFILEMGR_EnumInit(IFileMgr * pIFileMgr, const char * pszDir, boolean bDirs)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszDir	NULL terminated string containing the root directory name. "" is a valid file name and refers to the application's ROOT directory.
bDirs	Enumeration type that takes the following values: TRUE: enumerate directories only FALSE: enumerate files only

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IFILEMGR_EnumNext()
Return to the List of functions

IFILEMGR_EnumNext()

Description:

This function returns file information for the next file/directory within the specified root directory of the IFileMgr Interface object.

Prototype:

```
boolean IFILEMGR_EnumNext(IFileMgr * pIFileMgr, FileInfo * pInfo)
```

Parameters:

pIFileMgr	[in]	Pointer to the IFileMgr Interface object
pInfo	[out]	placeholder for file information

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

The file information this function retrieves contains the file attributes, file creation date and file size.

Side Effects:

None

See Also:

IFILEMGR_EnumInit()
FileInfo
Return to the List of functions

IFILEMGR_GetFreeSpace()

Description:

This function gets the number of free bytes currently available on file system. If the passed parameter **pdwTotal** is non-NULL, this function initializes **pdwTotal** with the total room in the file system.

Prototype:

```
uint32 IFileMgr_GetFreeSpace(IFileMgr * pIFileMgr, uint32 * pdwTotal)
```

Parameters:

pIFileMgr	[in]	Pointer to the IFileMgr Interface object
pdwTotal	[in/out]	Placeholder for the total room in the file system.

Return Value:

Number of bytes of file system space currently available	If successful
0 (zero)	If fails

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IFILEMGR_GetInfo()

Description:

This function gets the information on a file identified by a file name. The file information this function retrieves contains the file attributes, file creation date, and file size.

Prototype:

```
int IFILEMGR_GetInfo(IFileMgr * pIFileMgr, const char * pszName, FileInfo * pInfo)
```

Parameters:

pIFileMgr	[in]	Pointer to the IFileMgr Interface object
pszName	[in]	Null terminated string containing the file name. "" is a valid file name and refers to the application's ROOT directory.
pInfo	[out]	placeholder for file information

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IFILEMGR_GetInfo(), FileInfo
Return to the List of functions

IFILEMGR_GetLastError()

Description:

This function would typically be called when a file operation performed by the applet has failed and the applet needs to know about the reason for the failure.

Prototype:

```
int IFILEMGR_GetLastError(IFileMgr * pIFileMgr)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object

Return Value:

SUCCESS	If last file operation was successful

If unsuccessful one of the following errors return:

EFILEEXISTS	File exists
EFILENOEXISTS	File does not exist
EDIRNOTEMPTY	Directory not empty
EBADFILENAME	Bad file name
EBADSEEKPOS	Bad seek position
EFILEEOF	End of file
EFSFULL	File system full
EFILEOPEN	File already open
EBADPARM	Invalid parameter

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IFILEMGR_MkDir()

Description:

This function is used to create a directory, specified by a directory name.

Prototype:

```
int IFILEMGR_MkDir(IFileMgr * pIFileMgr, const char * pszDir)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszDir	NULL terminated string containing a directory name

Return Value:

SUCCESS	If file is successfully created or if file already exists
EFAILED	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

IFILEMGR_RmDir()
Return to the List of functions

IFILEMGR_OpenFile()

Description:

This function is used to open a file in specified mode.

Prototype:

```
IFile * IFILEMGR_OpenFile(IFileMgr * pIFileMgr, const char * pszFile,
OpenFile mode)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszFile	Null terminated string containing the file name
mode	File open mode

The file open mode takes the following values:

_OFM_READ	Open in read-only mode
_OFM_READWRITE	Open in read/write mode
_OFM_CREATE	Create
_OFM_APPEND	Open for appending

Return Value:

Pointer	To the IFile Interface object of the file opened If successful,
NULL	If the function call fails

Comments:

If NULL pointer is returned, use IFILEMGR_GetLastError() to get error details. To close a file that has been opened using the IFILEMGR_OpenFile(), release the IFile Interface pointer using the IFILEMGR_Release(). If pszFile contains the path where the file exists (or needs to be created), the file is opened (or created) if the caller has the privileges to open/create files in that directory, and the directory exists. Directories can be created using the IFILEMGR_MkDir(). Files are created with read/write access.

Side Effects:

If the file is opened in the _OFM_APPEND mode the file pointer is

moved to the end of the file.

See Also:

IFILEMGR_MkDir()
Return to the List of functions

IFILEMGR_Release()

Description:

This function decrements the reference count for the IFileMgr Interface object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IFILEMGR_Release(IFileMgr * pIFileMgr)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object

Return Value:

Returns the updated reference count.

Comments:

None

Side Effects:

None

See Also:

IFILEMGR_AddRef()
Return to the List of functions

IFILEMGR_Remove()

Description:

This function is used to remove a file identified by a given file name.

Prototype:

```
int IFILEMGR_Remove(IFileMgr * pIFileMgr, const char * pszName)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszName	Null terminated string containing the name of the file to to be removed

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

A file needs to be closed prior to it being removed. Use IFILE_Release() to close a file.

Side Effects:

None

See Also:

None
Return to the List of functions

IFILEMGR_Rename()

Description:

This function renames the file from the source name to the destination name.

Prototype:

```
int IFILEMGR_Rename(IFileMgr * pIFileMgr, const char * pszSrc, const char *
pszDest)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszSrc	Source file to be renamed
pszDest	Destination file

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

If EFAILED is returned, use IFILEMGR_GetLastError() to get the error details.

Side Effects:

None

See Also:

IFILEMGR_EnumNext()
Return to the List of functions

IFILEMGR_RmDir()

Description:

This function is used to remove a directory identified by a given directory name.

Prototype:

```
int IFILEMGR_RmDir(IFileMgr * pIFileMgr, const char * pszDir)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszDir	NULL terminated string containing a directory name

Return Value:

SUCCESS	If successful
EFAILED	If unsuccessful

Comments:

If there are files or directories elements beneath the directory to be removed, they must be removed prior to calling this function, or this function call fails and the directory is not removed.

Side Effects:

None

See Also:

IFILEMGR_MkDir()
Return to the List of functions

IFILEMGR_Test()

Description:

This function tests if the specified file/directory exists.

Prototype:

```
int IFILEMGR_Test(IFileMgr * pIFileMgr, const char * pszName)
```

Parameters:

pIFileMgr	Pointer to the IFileMgr Interface object
pszName	NULL terminated string containing file/directory name. "" is a valid file name and refers to the application's ROOT directory.

Return Value:

SUCCESS	If specified file/directory exists
EFAILED	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

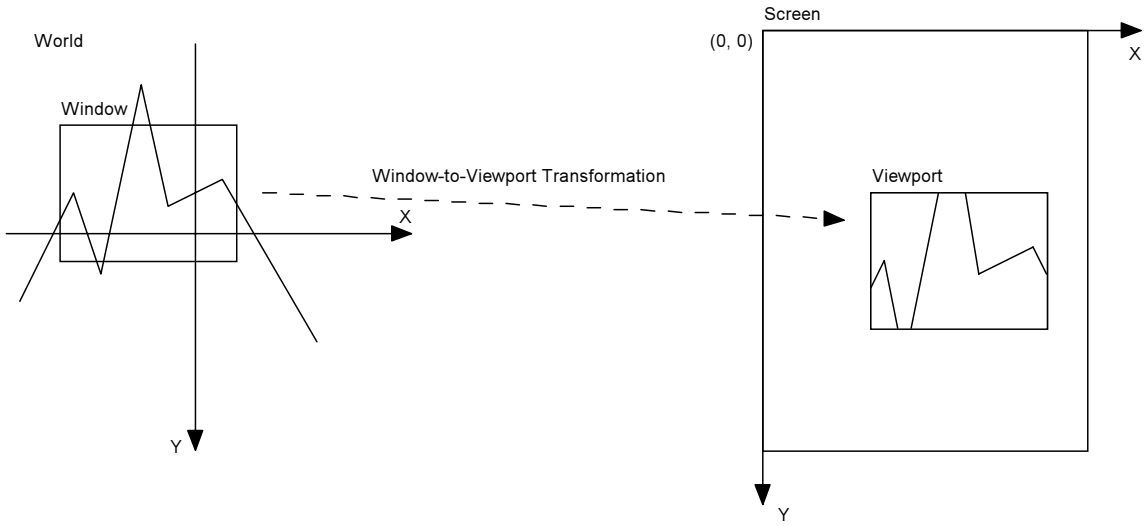
IGraphics Interface

The IGraphics Interface functions extend the drawing capability of the IDisplay Interface to more complex 2D geometric primitives, such as circles and arbitrary polygons. The IGraphics Interface also supports clipping and viewing transformation. IGraphics drawing operations get updated on the screen only after calling either IGRAPHICS_Update() function or IDISPLAY_Update() function. BREW recommends IGRAPHICS_Update() over IDISPLAY_Update(). Future releases of this interface will support double buffering, which will allow the program to direct drawing operations to an off-screen buffer that can be swapped with the current screen contents.

IGraphics coordinate systems and viewing transformation management

In BREW, all the coordinate systems are expressed in terms of pixels. In IGraphics interfaces, BREW makes a distinction between the world-coordinate system and the screen-coordinate system. In both systems, the positive x-axis extends from left to right, and the positive y-axis extends downward.

The world-coordinate system is the logical coordinate system that a programmer uses to describe the scene. The origin of this system can be anywhere for the programmer's convenience. The screen coordinate system always has its origin at the upper left corner of the device screen. A world-coordinate system area selected for display is called a window (notice the difference from what a window means for Microsoft Windows). An area on the screen to which a window is mapped is called a viewport. The window defines what is to be viewed; the viewport defines where it is to be displayed. Since IGraphics does not support scaling in this version, the window and viewport have‘ identical size. IGraphics Interface supports the viewing transformation (also referred to as the window-to-viewport transformation) with the following three functions: IGRAPHICS_SetViewport(), IGRAPHICS_Translate(), and IGRAPHICS_Pan().



IGRAPHICS_SetViewport() provides the ability to set a rectangular area (viewport) within the screen coordinate system. This viewport is the displayable area for all IGraphics operations. IGRAPHICS_GetViewport() returns the current viewport in screen coordinates. By default, the entire screen is the IGraphics viewport .

IGRAPHICS_Translate() provides the ability to move the window in the world-coordinate system. By default the upper left corner of the window is at the origin (0, 0) of the world-coordinate system.

Developers who are not familiar with coordinate systems and other graphics-related terminology, BREW recommends additional reading of *Computer Graphics: C Version* by Donald Hearn and M. Pauline Baker.

IGraphics Set and Get functions

The IGraphics functions use a number of parameters to determine how objects are to be drawn on the screen. Once a parameter's value is set, that value is used in all subsequent drawing operations until the parameter is changed. The IGraphics Interface includes several Set functions that let you set the values of these parameters and corresponding Get functions that let you retrieve the parameter values. The following Set functions are provided by IGraphics Interface:

- IGRAPHICS_SetBackground() sets the background color to be used when IGRAPHICS_ClearRect() and IGRAPHICS_ClearViewport() are called. This background color is also used when setting a clipping region with the AEE_GRAPHICS_CLEAR flag set.
- IGRAPHICS_SetClip() specifies the clipping region to be used for drawing. Drawing operations is to take place only inside the clipping region, which is defined by an AEEClip structure.
- IGRAPHICS_SetColor() sets the value of the foreground color. All lines and shape borders are to be drawn in this color.
- IGRAPHICS_SetFillColor() sets the color to be used to draw the interiors of closed geometric shapes.
- IGRAPHICS_SetFillMode() sets the fill mode. If the fill mode is TRUE, the interiors of closed geometric primitives are filled with the current fill color. If the fill mode is FALSE, only the border of the geometric primitive is drawn.
- IGRAPHICS_SetPaintMode() sets the paint mode, which determines how the color of a pixel to be drawn is combined with the color already on the screen. If the paint mode is AEE_PAINT_COPY, the new pixel color overwrites the old color. If the paint mode is AEE_PAINT_XOR, the color displayed to the screen is the exclusive-OR of the new and old pixel color .
- IGRAPHICS_SetPointSize() sets the width in pixels of points drawn to the screen.

IGraphics drawing functions

The IGraphics functions used to draw various 2D geometric primitives are as follows:

- IGRAPHICS_DrawPoint() draws a point at a specified x and y coordinate. The point's height and width in pixels are equal to the current point size.
- IGRAPHICS_DrawLine() draws a line, given its starting and ending points.
- IGRAPHICS_DrawRect() draws a rectangle. The rectangle is defined by its upper left corner and the pixel counts in positive x and y directions.
- IGRAPHICS_DrawCircle() draws a circle, which is defined by its center point and radius.
- IGRAPHICS_DrawArc() draws an arc, which is a portion of the border of a circle. The function takes as input the center point and radius of the circle and the starting and ending angles defining the portion of the circle to be displayed.
- IGRAPHICS_DrawPie() draws a pie, which is a wedge-shaped portion of a circle. It is defined as the region included between two lines drawn from the center of the circle to its circumference. This function takes as input the center point and radius of the circle and the angles of the two lines.
- IGRAPHICS_DrawEllipse() draws an ellipse, which is defined by its center point and the half-length of its major and minor axis. The major axis has to be aligned with either x-axis or y-axis.
- IGRAPHICS_DrawPolygon() draws a polygon, which is defined by a set of points specifying its vertices.
- IGRAPHICS_DrawPolyline() draws a polyline, which is defined by a list of points. This function draws line segments that connect each adjacent pair of points in the list.
- IGRAPHICS_DrawTriangle() draws a triangle, which is defined by the coordinates of its three vertices.

All input to the IGraphics drawing functions are in the world-coordinate system.

IGraphics miscellaneous functions

- IGRAPHICS_ClearRect() clears a rectangular region in the world-coordinate system by filling the region with the current background color.
- IGRAPHICS_ClearViewport() clears the viewport on the screen. The input rectangle is in the screen coordinate system.
- IGRAPHICS_Update() updates the screen with the latest drawing operations.

IGraphics interface usage

To use the IGraphics Interface functions

1. Call ISHELL_CreateInstance() to create an instance of the IGraphics Interface.
2. Call IGRAPHICS_SetViewport() to specify a rectangular area of the screen for display.
3. Call IGRAPHICS_SetBackground() to set the background color.
4. Call IGRAPHICS_ClearViewport() to clear the viewport by filling it with the background color.
5. Call IGRAPHICS_Translate() or IGRAPHICS_Pan() to specify the viewing window in the world-coordinate system.
6. Call the parameter setting functions to set the foreground color, clipping region, fill color and mode, paint mode, and point size that is to be used to draw the geometric primitives.
7. Call one of the drawing functions to draw an IGraphics object on the screen. All objects are in the world-coordinate system.
8. Repeat step 6 and 7 for as many geometric primitives as needed.
9. Call IGRAPHICS_Update() to display the drawing on the screen.
10. Release the IGraphics Interface object by calling IGRAPHICS_Release().

List of functions

Functions in this interface include:

IGRAPHICS_AddRef()

IGRAPHICS_ClearRect()

IGRAPHICS_ClearViewport()

IGRAPHICS_DrawArc()

IGRAPHICS_DrawCircle()

IGRAPHICS_DrawEllipse()

IGRAPHICS_DrawLine()

IGRAPHICS_DrawPie()

IGRAPHICS_DrawPoint()

IGRAPHICS_DrawPolygon()

IGRAPHICS_DrawPolyline()

IGRAPHICS_DrawRect()

IGRAPHICS_DrawTriangle()

IGRAPHICS_EnableDoubleBuffer()

IGRAPHICS_GetBackground()

IGRAPHICS_GetClip()

IGRAPHICS_GetColor()

IGRAPHICS_GetColorDepth()

IGRAPHICS_GetFillColor()

IGRAPHICS_GetFillMode()

IGRAPHICS_GetPaintMode()

IGRAPHICS_GetPointSize()

IGRAPHICS_GetViewport()

IGRAPHICS_Pan()

IGRAPHICS_Release()

IGRAPHICS_SetBackground()

IGRAPHICS_SetClip()

IGRAPHICS_SetColor()

IGRAPHICS_SetFillColor()

IGRAPHICS_SetFillMode()

IGRAPHICS_SetPaintMode()

IGRAPHICS_SetPointSize()

IGRAPHICS_SetViewport()

IGRAPHICS_Translate()

IGRAPHICS_Update()

Return to the Contents

IGRAPHICS_AddRef()

Description:

This function increments the reference count for the IGraphics Interface object.

Prototype:

```
uint32 IGRAPHICS_AddRef(IGraphics * pIGraphics)
```

Parameters:

pIGraphics Pointer to the IGraphics Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IGRAPHICS_Release()
Return to the List of functions

IGRAPHICS_ClearRect()

Description:

This function clears a rectangular area of the screen by filling it with the background color. The input rectangle is in the world-coordinate system.

Prototype:

```
int IGRAPHICS_ClearRect(IGraphics * pIGraphics, AEERect * pRect)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pRect	Pointer to the rectangle that specifies the area to clear

Return Value:

SUCCESS	If unsuccessful
EBADPARM	If the **pRect** parameter is invalid

Comments:

IGRAPHICS_SetBackground() has no effect until IGRAPHICS_ClearRect() or IGRAPHICS_ClearViewport() is called.

Side Effects:

None

See Also:

AEERect
Return to the List of functions

IGRAPHICS_ClearViewport()

Description:

This function clears the current viewport to the background color.

Prototype:

```
void IGRAPHICS_ClearViewport(IGraphics * pIGraphics)
```

Parameters:

pIGraphics Pointer to the IGraphics Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also

None
Return to the List of functions

IGRAPHICS_DrawArc()

Description:

This function draws a circular arc. The center, radius, starting angle, and ending angle are specified by **pArc.** The color of the arc is the current foreground color.

Prototype:

```
int IGRAPHICS_DrawArc(IGraphics * pIGraphics, AEEArc * pArc)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pArc	Pointer to the arc to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If unsuccessful
EBADPARM	If the **pArc** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEEArc
Return to the List of functions

IGRAPHICS_DrawCircle()

Description:

This function draws a circle. The center, and radius of the circle are specified by **pCircle.** The color of the outline is the current foreground color. If the fill-mode is turned on, the interior of the circle is filled with the current fill color.

Prototype:

```
int IGRAPHICS_DrawCircle(IGraphics * pIGraphics, AEECircle * pCircle)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pCircle	Pointer to the circle to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pCircle** parameter is invalid

Comments:

The input circle is in the world-coordinate system.

Side Effects:

None

See Also:

AEECircle
Return to the List of functions

IGRAPHICS_DrawEllipse()

Description:

This function draws an ellipse specified by **pEllipse**. The color of the outline is the current foreground color. If the fill-mode is turned on, its interior is filled with the current fill-color.

Prototype:

```
int IGRAPHICS_DrawEllipse(IGraphics * pIGraphics, AEEEllipse * pEllipse)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pEllipse	Pointer to the ellipse to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pEllipse** parameter is invalid

Comments:

The input is in the world-coordinate system .

Side Effects:

None

See Also:

AEEEllipse
Return to the List of functions

IGRAPHICS_DrawLine()

Description:

This function draws a line segment, given the starting point and ending point. The color of the line is the current foreground color.

Prototype:

```
int IGRAPHICS_DrawLine(IGraphics * pGraphics, AEELine * pLine)
```

Parameters:

pGraphics	Pointer to the IGraphics Interface object
pLine	Pointer to the line to be drawn

Return Value:

An error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pLine** parameter is invalid

Comments:

The input is in the world-coordinate system

Side Effects:

None

See Also:

AEELine
Return to the List of functions

IGRAPHICS_DrawPie()

Description:

This function draws a circular pie specified by **pPie**. The color of the outline is the current foreground color. If fill-mode is turned on, the interior of the pie is filled with the current fill color.

Prototype:

```
int IGRAPHICS_DrawPie(IGraphics * pIGraphics, AEEPie * pPie)
```

Parameters:

pIGraphics	Pointer tothe IGraphics Interface object
pPie	Pointer to the pie to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pPie** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEEPie
Return to the List of functions

IGRAPHICS_DrawPoint()

Description:

This function draws a point with the current point size and foreground color.

Prototype:

```
int IGRAPHICS_DrawPoint(IGraphics * pGraphics, AEEPoint * pPoint)
```

Parameters:

pGraphics	Pointer to the IGraphics Interface object
pPoint	Pointer to the point to be drawn

Return Value:

An error code is returned to indicate the status of this transaction:

SUCCESS	If the pointer is successfully drawn
EBADPARM	If the **pPoint** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEEPoint
Return to the List of functions

IGRAPHICS_DrawPolygon()

Description:

This function draws a polygon specified by **pPolygon**. The color of the outline is the current foreground color. If the fill-mode is turned on, its interior is filled with the current fill-color.

Prototype:

```
int IGRAPHICS_DrawPolygon(IGraphics * pIGraphics, AEEPolygon * pPolygon)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pPolygon	Pointer to the polygon to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pPolygon** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEEPolygon
Return to the List of functions

IGRAPHICS_DrawPolyline()

Description:

This function draws a polyline specified by **pPolyline**. The color of the outline is the current foreground color.

Prototype:

```
int IGRAPHICS_DrawPolyline(IGraphics * pIGraphics, AEEPolyline * pPolyline)
```

Parameters

pIGraphics	Pointer to the IGraphics Interface object
pPolyline	Pointer to the polyline to be drawn

Return Value:

SUCCESS	If function successfully draws a polyline
EBADPARM	If the **pPolyline** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEEPolyline
Return to the List of functions

IGRAPHICS_DrawRect()

Description:

This function draws an axis-aligned rectangle specified by **pRect**. The color of the outline is the current foreground color. If the fill-mode is turned on, the interior is filled with the current fill color.

Prototype:

```
int IGRAPHICS_DrawRect(IGraphics * pIGraphics, AEERect * pRect)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pRect	Pointer to the rectangle to be drawn

Return Value:

The error code indicating the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pRect** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_DrawTriangle()

Description:

This function draws a triangle specified by **pTriangle**. The color of the outline is the current foreground color. If the fill-mode is turned on, its interior is filled with the current fill-color.

Prototype:

```
int IGRAPHICS_DrawTriangle(IGraphics * pIGraphics, AEETriangle * pTriangle)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pTriangle	Pointer to the triangle to be drawn

Return Value:

The error code that indicates the status of this transaction:

SUCCESS	If successful
EBADPARM	If the **pTriangle** parameter is invalid

Comments:

The input is in the world-coordinate system.

Side Effects:

None

See Also:

AEETriangle
Return to the List of functions

IGRAPHICS_EnableDoubleBuffer()

Description:

This function turns on double buffering by passing in TRUE and turns it off by passing in FALSE.

Prototype:

```
boolean IGRAPHICS_EnableDoubleBuffer(IGraphics * pIGraphics, boolean flag)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
flag	A Boolean value: Set flag to TRUE to enable double buffering. Set flag to FALSE to disable double buffering.

Return Value:

TRUE	If the transaction is successful
FALSE	If otherwise

If double buffering is not supported by the system, it returns FALSE when the client tries to turn on the double buffering.

Comments:

Double buffering allows you to maintain two separate buffers. One is displayed while the other is being drawn. A call to IGRAPHICS_Update() swaps the buffers.

Double buffering is not supported in this version.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetBackground()

Description:

This function queries the RGB values of the current background color.

Prototype:

```
void IGRAPHICS_GetBackground(IGraphics * pIGraphics, uint8 * r, uint8 * g,
uint8 * b)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
r	Pointer to the red component of the RGB color
g	Pointer to the green component of the RGB color
b	Pointer to the blue component of the RGB color

Return Value:

None

Comments:

The default background color is white.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetClip()

Description:

This function gets the current clipping area, specified by a closed geometric primitive.

Prototype:

```
boolean IGRAPHICS_GetClip(IGraphics * pIGraphics, AEEClip * pShape)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pShape	Pointer the current clipping shape

Return Value:

TRUE	If the query is successful,
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

AEEClip
Return to the List of functions

IGRAPHICS_GetColor()

Description:

This function gets the RGBA value for the current foreground color.

Prototype:

```
void IGRAPHICS_GetColor(IGraphics * pIGraphics, uint8 * r, uint8 * g, uint8 * b, uint8 * alpha)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
r	Pointer to the red component of the RGBA value
g	Pointer to the green component of the RGBA value
b	Pointer to the blue component of the RGBA value
alpha	Alpha value

Return Value:

None

Comments:

The **alpha** parameter is just a placeholder in the current version. It has no effect on the foreground color.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetColorDepth()

Description:

This function gets the color depth of the device.

Prototype:

```
uint8 IGRAPHICS_GetColorDepth(IGraphics * pIGraphics)
```

Parameters:

pIGraphics Pointer to the IGraphics Interface object

Return Value:

The color depth.

Comments:

None

Side Effects:

None

See Also:

ISHELL_GetDeviceInfo()
Return to the List of functions

IGRAPHICS_GetFillColor()

Description:

This function gets the RGBA value for the current fill color.

Prototype:

```
void IGRAPHICS_GetFillColor(IGraphics * pIGraphics, uint8 * pRed, uint8 *
pGreen, uint8 * pBlue, uint8 * pAlpha)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pRed	Red component of the RGBA value
pGreen	Green component of the RGBA value
pBlue	Blue component of the RGBA value
pAlpha	Alpha value

Return Value:

None

Comments:

pAlpha is just a placeholder in the current version. The value referred to by * **pAlpha** has no effect on the fill color.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetFillMode()

Description:

This function queries the current fill-mode. TRUE for fill and FALSE for not-fill.

Prototype:

```
boolean IGRAPHICS_GetFillMode(IGraphics * pIGraphics)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object

Return Value:

TRUE	For fill
FALSE	For not-fill

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetPaintMode()

Description:

This function gets the current paint mode.

Prototype:

```
AEEPaintMode IGRAPHICS_GetPaintMode(IGraphics * pIGraphics)
```

Parameters:

pIGraphics Pointer to the IGraphics Interface object

Return Value:

The enum value of the current paint mode

Comments:

None

Side Effects:

None

See Also:

AEEPaintMode
Return to the List of functions

IGRAPHICS_GetPointSize()

Description:

This function gets the current point size in terms of number of pixels.

Prototype:

```
uint8 IGRAPHICS_GetPointSize(IGraphics * pIGraphics)
```

Parameters:

pIGraphics Pointer to the IGraphics Interface object

Return Value:

The current point size

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_GetViewport()

Description

This function gets the current viewport (a rectangular area in the screen coordinate frame) for drawing.

Prototype:

```
boolean IGRAPHICS_GetViewport(IGraphics * pIGraphics, AEERect * pRect,
boolean * framed)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pRect	Pointer to the rectangular shape that specifies the current viewport area for drawing
framed	TRUE if the viewport is framed, FALSE otherwise

Return Value:

TRUE	If successful
FALSE	If otherwise

Comments:

The output is in screen coordinate system.

Side Effects:

None.

See Also:

IGRAPHICS_SetViewport()
AEERect
Return to the List of functions

IGRAPHICS_Pan()

Description:

This function re-centers the window in the world-coordinate system. In some sense, (cx, cy) is the world coordinates at which the camera is pointing.

Prototype:

```
void IGRAPHICS_Pan(IGraphics * pIGraphics, int16 cx, int16 cy)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
cx	X coordinate of the new center in world-coordinate frame
cy	Y coordinate of the new center in world-coordinate frame

Return Value:

None.

Comments:

None.

Side Effects

None

See Also:

IGRAPHICS_Translate()
Return to the List of functions

IGRAPHICS_Release()

Description:

This function decrements the reference count for the graphics object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IGRAPHICS_Release(IGraphics * pIGraphics)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object whose reference count needs to be decremented

Return Value:

Updated reference count for the object

Comments:

None

Side Effects:

None

See Also:

IGRAPHICS_AddRef()
Return to the List of functions

IGRAPHICS_SetBackground()

Description:

This function sets the RGB values of the current background color. This color remains in effect until it is set to different values.

Prototype:

```
RGBVAL IGRAPHICS_SetBackground(IGraphics * pIGraphics, uint8 r, uint8 g,
uint8 b)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
r	Red component of the RGB value
g	Green component of the RGB value
b	Blue component of the RGB value

Return Value:

The updated RGB value for the current background color

Comments:

The program does not show the new background color until IGRAPHICS_ClearRect() or IGRAPHICS_ClearViewport() is called.

The value of each color component (red, green and blue) must be a positive value between 0 (zero) and 255. Any integer values beyond this range are converted to a value within the range from 0 (zero) to 255. However the conversion itself is machine dependent. The user is encouraged to use values within the range to have predictable color effect.

Side Effects:

None

See Also:

IGRAPHICS_GetBackground()
Return to the List of functions

IGRAPHICS_SetClip()

Description:

This function sets the clipping area. Graphics content is displayed on the screen only if it is within the clipping area. If the pointer pShape is NULL, it resets the clipping region to the display window. The clipping is in the world-coordinate system.

Prototype:

```
boolean IGRAPHICS_SetClip(IGraphics * pIGraphics, AEEClip * pShape), uint8
nFlag)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pShape	Pointer to the clipping shape
pFlag	Bitmap flags

Return Value:

TRUE	If the clipping area is set successfully,
FALSE	If otherwise

Comments:

If **AEE_GRAPHICS_FRAME** is set for **nFlag**, it draws the frame using the current foreground color and the paint mode, while setting the clipping region.

If **AEE_GRAPHICS_CLEAR** is set for **nFlag**, it clears the interior of of the clipping region while setting it.

If **AEE_GRAPHICS_FILL** is set for **nFlag**, it fills the interior using the current fill color and the current paint mode, while setting the clipping region..

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_SetColor()

Description:

This function sets the current RGBA value. All subsequent graphics objects is drawn in this color until this function is called again for different RGBA values.

Prototype:

```
RGBVAL IGRAPHICS_SetColor(IGraphics * pIGraphics, uint8 r, uint8 g, uint8 b,
uint8 alpha)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
r	Red component of the RGBA value
g	Green component of the RGBA value
b	Blue component of the RGBA value
alpha	Alpha value

Return Value:

Updated RGBA value for foreground

Comments:

In the current version, **alpha** value is just a placeholder, which has no effect to the foreground color. Eventually this value will be used to support transparent color and color blending. The default foreground color is black.

Side Effects:

None

See Also:

IGRAPHICS_GetColor()
Return to the List of functions

IGRAPHICS_SetFillColor()

Description:

This function sets the fill color for all subsequent drawings of closed geometric primitives.

Prototype:

```
RGBVAL IGRAPHICS_SetFillColor(IGraphics * pIGraphics, uint8 r, uint8 g,
uint8 b, uint8 alpha)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
r	Red component of the RGBA value
g	Green component of the RGBA value
b	Blue component of the RGBA value
alpha	Alpha value

Return Value:

The updated RGBA value of the current fill color

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_SetFillMode()

Description:

This function turns on the fill-mode for all subsequent closed geometric primitives by passing in TRUE. It turns off the fill-mode by passing in FALSE.

Prototype:

```
boolean IGRAPHICS_SetFillMode(IGraphics * pIGraphics, boolean fFill)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
fFill	Boolean value: Set **fFill** to TRUE to turn on the fill-mode. Set **fFill** to FALSE to turn off the fill-mode.

Return Value:

The updated value for the current fill-mode.

TRUE	For on
FALSE	For off

Comments:

If an invalid value (other than TRUE or FALSE) is passed as input parameter, the fill-mode is set to FALSE by default.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_SetPaintMode()

Description:

This function sets the paint mode. In AEE_PAINT_COPY mode, the new graphics content overwrites the previous one. In AEE_PAINT_XOR mode, the color is determined by the XOR binary operation of the old and new colors.

Given an arbitrary background color and an arbitrary new foreground color, XORing the foreground color against the background color twice, recovers the background color.

Prototype:

```
AEEPaintMode IGRAPHICS_SetPaintMode(IGraphics * pIGraphics, AEEPaintMode mode)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
mode	enum value for paint mode

Return Value:

Updated enum value for paint mode.

Comments:

If an invalid integer (enum) value is passed as the input "mode" parameter, the paint mode is set to AEE_PAINT_COPY by default.

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_SetPointSize()

Description:

This function sets the size (width) of a point in terms of number of pixels.

Prototype:

```
uint8 IGRAPHICS_SetPointSize(IGraphics * pIGraphics, uint8 u8Size)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
u8Size	Size (width) of a point in terms of number of pixels

Return Value:

The updated value of the point size.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IGRAPHICS_SetViewport()

Description:

A world-coordinate area selected for display is called a window. An area on a display device to which a window is mapped is called a viewport.

This function sets the viewport (a rectangular area) for drawing. The default viewport is the screen without a frame If pRect is NULL, this function resets the viewport to the default.

Prototype:

```
boolean IGRAPHICS_SetViewport(IGraphics * pIGraphics, AEERect * pRect,uint8
nFlag)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
pRect	Pointer to the rectangular shape, which specifies the area for display
nFlag	Bitmap flags for different options • If flag is set as AEE_GRAPHICS_FRAME then the minimum value for dx, dy is 3 pixel. • If flag is set as AEE_GRAPHICS_NONE then the minimum value for dx, dy is 1 pixel. • If IGRAPHICS_SetViewport returns FALSE for any reason, then previous viewport settings are retained.

Return Value:

TRUE	If successful
FALSE	If otherwise

Comments:

It returns TRUE only if the displayable area is non-empty and the viewport is completely contained in the physical screen. Otherwise, it return FALSE.

If AEE_GRAPHICS_FRAME is set, it draws the frame.

If AEE_GRAPHICS_CLEAR is set, it clears the viewport to the background color.

While the inputs of other IGraphics functions are in the world-coordinate system, the input of this function is in the screen-coordinate system.

Side Effects:

None

See Also:

IGRAPHICS_GetViewport()
Return to the List of functions

IGRAPHICS_Translate()

Description:

This function translates the window's origin in the world-coordinate system. Namely, different areas of the world are thereby displayed. The units are in number of pixels. The window's origin is always its upper left corner, which is the default origin of the world-coordinate system. As a result of the translation, all geometric primitives are translated by the same amount, but in the opposite direction in the viewport. Please refer to IGRAPHICS_SetViewport() for the definition of **window** and **viewport**.

Prototype:

```
void IGRAPHICS_Translate(IGraphics * pIGraphics, int16 x, int16 y)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object
x	The window's origin to be translated by x pixels along X-axis
y	The window's origin to be translated by y pixels along Y-axis

Return Value:

None.

Comments:

Conceptually, it helps to imagine a **window** as the camera. This camera can be translated such that it points to different area of the world. The world is the scene that the programmer has designed.

Side Effects:

None.

See Also:

IGRAPHICS_Pan()
Return to the List of functions

IGRAPHICS_Update()

Description:

If double buffering is supported and enabled, it swaps the two buffers and pushes the offscreen buffer to the screen. If the double buffering is not supported or it is not enabled, this function updates the device screen.

Prototype:

```
void IGRAPHICS_Update(IGraphics * pIGraphics)
```

Parameters:

pIGraphics	Pointer to the IGraphics Interface object

Return Value:

None

Comments:

Double buffering is not supported in this release.

Side Effects:

None

See Also:

None
Return to the List of functions

IHeap Interface

The IHeap Interface includes functions for allocating and freeing memory, and for obtaining information about the amount of device memory that is available and in use. For simple memory allocation tasks, you can use the MALLOC() and FREE() macros that are defined in the file AEEStdLib.h.

IHEAP_CheckAvail() checks whether there is enough memory available to allocate a block of a specified size. IHEAP_GetMemStats() returns the amount (number of bytes) of memory in the system that is currently in use (you can use the function ISHELL_GetDeviceInfo() to obtain the total amount of memory on the device).

The function IHEAP_Malloc() allocates a block of memory of a specified size and returns a pointer to it. IHEAP_MallocRec() takes as input a data type; it allocates enough memory for a single instance of this data type and returns a pointer to the memory allocated (the pointer is cast to the specified type). IHEAP_Realloc() is used to change the size of an allocated memory block. IHEAP_StrDup() makes a copy of a character string and returns a pointer to the memory allocated to hold the copy. You call IHEAP_Free() to free the memory blocks allocated by these functions.

List of functions

Functions in this interface include:

IHEAP_AddRef()

IHEAP_CheckAvail()

IHEAP_Free()

IHEAP_GetMemStats()

IHEAP_Malloc()

IHEAP_MallocRec()

IHEAP_Realloc()

IHEAP_Release()

IHEAP_StrDup()

Return to the Contents

IHEAP_AddRef()

Description:

This function increments the reference count for the heap object

Prototype:

```
uint32 IHEAP_AddRef(IHeap * pIHeap)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object

Return Value:

The updated reference count

Comments:

None

Side Effects:

None

See Also:

IHEAP_Release()
Return to the List of functions

IHEAP_CheckAvail()

Description:

This function checks whether a memory block of the given size can be allocated. This function does not do any actual allocation of memory. It just returns TRUE or FALSE indicating whether or not it is possible to allocate a block of the given size.

Prototype:

```
boolean IHEAP_CheckAvail(IHeap * pIHeap, uint32 dwSize)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
dwSize	Size of the block whose allocation needs to be verified

Return Value:

TRUE	If a block of the given size can be allocated
FALSE	If unsuccessful or if a block of the given size cannot be allocated

Comments:

None

Side Effects:

This function may walk-through the heap (and collapse adjacent free blocks if any)

See Also:

IHEAP_GetMemStats()
Return to the List of functions

IHEAP_Free()

Description:

This function frees an allocated block of memory. It releases the memory back to the memory pool.

Prototype:

```
void IHEAP_Free(IHeap * pIHeap, void * pMemFree)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
pMemFree	Pointer to the memory block that needs to be freed

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IHEAP_GetMemStats()

Description:

This function returns statistics about the current memory use. It returns the total memory currently in use. To check if a block of a specific size can be allocated, the function IHEAP_CheckAvail() must be called. To get the total memory in the system, the function ISHELL_GetDeviceInfo() must be called.

Prototype:

```
uint32 GetMemStats(IHeap * pIHeap)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object

Return Value:

Total used memory in the system

Comments:

None

Side Effects:

None

See Also:

IHEAP_CheckAvail()
ISHELL_GetDeviceInfo()
Return to the List of functions

IHEAP_Malloc()

Description:

This function allocates a block of memory of the requested size and returns the pointer to that memory block.

Prototype:

```
void * IHEAP_Malloc(IHeap * pIHeap, uint32 dwSize)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
dwSize	Specifies the size of the memory block to be allocated

Return Value:

If successful, returns pointer to the allocated memory block

If failed, returns NULL

Comments:

None

Side Effects:

None

See Also:

IHEAP_Realloc()
IHEAP_MallocRec()
Return to the List of functions

IHEAP_MallocRec()

Description:

This function allocates the memory required for a specified standard data type. It then casts the allocated pointer to that data type before returning to the caller.

Prototype:

```
(type * ) IHEAP_MallocRec(IHeap * pIHeap,type dataType)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
dataType	Specifies the standard data type for which memory needs to be allocated

Return Value:

Pointer	To the requested data type, if successful,
NULL	If fails,

Comments:

None

Side Effects:

None

See Also:

IHEAP_Malloc()
IHEAP_Realloc()
Return to the List of functions

IHEAP_Realloc()

Description:

This function re-allocates a memory block and changes its size.

Prototype:

```
void * IHEAP_Realloc(IHeap * pIHeap, void * pMemBlock, uint32 dwNewSize)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
pMemBlock	Pointer to the memory block that needs to be re-allocated. If pMemblock is NULL, this function behaves the same way as malloc and allocates a new block of dwNewSize bytes.
dwNewSize	Specifies the new size of the memory block

Return Value:

Pointer	To the re-allocated block, if successful
NULL	If unsuccessful

Comments:

The return value is NULL if the size is 0 (zero) and **pMemBlock** is valid, or if there is not enough available memory to expand **pMemBlock** to the given size. In the first case, the original block is freed. In the second, the original block is unchanged.

Side Effects:

None

See Also:

None
Return to the List of functions

IHEAP_Release()

Description:

This function decrements the reference count for the heap object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IHEAP_Release(IHeap * pIHeap)
```

Parameters:

pIHeap Pointer to IHeap Interface to the heap object

Return Value:

The updated reference count

Comments:

None

Side Effects:

None

See Also:

IHEAP_AddRef()
Return to the List of functions

IHEAP_StrDup()

Description:

This function duplicates a given string. It allocates memory for the new string and then copies the contents of the incoming string into this new string. It then returns the new string.

Prototype:

```
AECHAR * IHEAP_StrDup(IHeap * pIHeap, AECHAR * pszIn)
```

Parameters:

pIHeap	Pointer to IHeap Interface to the heap object
pszIn	Pointer to the string that needs to be duplicated

Return Value:

Pointer	To the duplicated string, If successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IImage Interface

The IImage Interface is used for drawing bitmap images and displaying animated bitmaps to the screen. This interface supports bitmaps in the Windows Bitmap (.BMP) format and may support a native bitmap formats specific to each device. Each instance of the IImage Interface is associated with a single bitmap image, which can be read in either from a resource file created with the BREW Resource Editor or directly from a file containing the bitmap. A call to IDISPLAY_Update() or IDISPLAY_UpdateEx() is needed to update the screen.

The IImage Interface supports the display of animated bitmaps. An animated bitmap consists of multiple frames that are side by side along the width of the bitmap (for example, an animated bitmap with four 20x20 frames would be 80 pixels wide and 20 pixels high). When you start animation, IImage displays the frames of the image one after the other at a rate you can specify. When the last frame has been displayed, the animation starts again with the first frame, and IImage repeatedly cycles through the frames until you explicitly stop the animation or release the IImage Interface instance.

The IIMAGE_Draw() function draws a bitmap image on the screen at a specified position. IIMAGE_DrawFrame() draws a single frame of an animated bitmap. IIMAGE_GetInfo() retrieves information about a bitmap, including its height and width in pixels, the number of colors, and the number of frames it contains if the bitmap is animated.

Return to the List of functions is used when you are reading the bitmap contents directly from a file instead of from a BREW resource file. In this case, you use the IFileMgr Interface and IFile Interface to read the bitmap file into a buffer in memory, create an instance of the IImage Interface, and then use Return to the List of functions to associate the address of the memory buffer with the IImage instance.

IIMAGE_Start() starts the animation of an animated bitmap, displaying its frames at the specified screen coordinates (before you call this function, you must use IIMAGE_SetParm() to define the number of frames in the image, as described later in this section). IIMAGE_Stop() stops the animation, with the last displayed frame of the image remaining on the screen until it is overwritten.

IIMAGE_SetParm() is used to set the values of the following image parameters:

- IPARM_SIZE determines the height and width of the bitmap. By reducing these values from the bitmap's original size, you can display a smaller portion of the bitmap. For example, setting the size of a 50x50 bitmap to 40x40 effectively removes a 10-pixel-wide strip from the right side and bottom of the image when it is displayed.

- IPARM_OFFSET determines the x and y coordinates of the bitmap's origin, which are both 0 (zero) by default. By setting the origin to a larger value, you can display a smaller portion of the bitmap. For example, changing the origin of a 50x50 bitmap from (0,0) to (10, 10) effectively removes a 10-pixel-wide strip from the left side and top of the image when it is displayed.

- IPARM_CXFRAME sets the width in pixels of each frame in an animated bitmap. This parameter or IPARM_NFRAMES must be set before starting animation of the bitmap. The frame width can evenly divide the total width of the bitmap.

- IPARM_NFRAMES sets the total number of frames in an animated bitmap. This parameter or IPARM_CXFRAME must be set before starting animation of the bitmap. The number of

frames can evenly divide the total width of the bitmap. If this value is specified to be less than or equal to 0 (zero), the number of frames is automatically calculated using the formula:

- nFrames = WidthOfImage / HeightOfImage.
- IPARM_RATE sets the animation rate, which is the number of milliseconds between the display of each frame of an animated bitmap. The default value is 150 milliseconds.
- IPARM_ROP sets the raster operation, which defines how the pixels currently on the screen (the destination bitmap) are to be logically combined with those of the bitmap image that is to be drawn on the screen (the source bitmap). The supported raster operations and the Boolean pixel operations they represent are as follows:

AEE_RO_OR	src | dest
AEE_RO_NOT	!src
AEE_RO_XOR	src ^ dest
AEE_RO_COPY	src
AEE_RO_TRANSPARENT	src
AEE_RO_MASK	src & dest
AEE_RO_MASKNOT	!src & dest
AEE_RO_MERGENOT	!src | dest

To use the IImage Interface
Obtain an instance of the IImage Interface that contains the bitmap you choose to display.

1 Next do one of the following:

If your bitmap is contained in a BREW resource file, call ISHELL_LoadResImage(), which returns a pointer to an IImage instance that contains the bitmap you specified.
You can also use ISHELL_LoadImage() to load the bitmap format image directly from the file. If your bitmap is contained in a file, call ISHELL_LoadImage() and pass the file name of the bitmap file. If the function is successful, then it returns a valid pointer to the IImage Interface object .
If your image is not animated, call IIMAGE_Draw() to draw the image at the desired position on the screen. If your image is animated, call IIMAGE_Start() to begin animation at a specified screen position and use IIMAGE_Stop() to stop the animation.
Call IIMAGE_Release() to free the IImage Interface instance (this also stops animation if necessary).

Example 1:

```
IImage * pImage; // Place holder for IImage interface pointer
// Use ISHELL to load the image from resource file
pImage = ISHELL_LoadResImage(.....);
IIMAGE_Draw(pIMage,x,y); // Draw image
```

In addition to the simple drawing ability, the IImage Interface also supports animated images. This includes image formats that do not normally support animation (such as Windows .BMP).

Example 2:

```
IImage * pImage; // Place holder for IImage interface pointer
// Use ISHELL to load the image from resource file

pImage = ISHELL_LoadResImage(.....);
// If the image is a BMP, we can divide into "frames" each of
// which is "cxFrames" wide as follows:

IIMAGE_SetParm(IImage,IPARM_CXFRAME,cxFrame,0);

IIMAGE_Start(IImage,10,10); // Start animating the image
// Later we can stop the animation by either calling IIMAGE_Stop
// or IIMAGE_Release.

IIMAGE_Stop(IImage); // Stop the animation
```

List of functions

Functions in this interface include:

IIMAGE_AddRef()

IIMAGE_Draw()

IIMAGE_DrawFrame()

IIMAGE_GetInfo()

IIMAGE_HandleEvent()

IIMAGE_Notify()

IIMAGE_Release()

IIMAGE_SetParm()

IIMAGE_SetStream()

IIMAGE_Start()

IIMAGE_Stop()

Return to the Contents

IIMAGE_AddRef()

Description:

This function increments the reference count of the IImage Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IIMAGE_Release().

Prototype:

```
uint32 IIMAGE_AddRef(IImage * pIImage)
```

Parameters:

pIImage	Pointer to the IImage Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IIMAGE_Release()
Return to the List of functions

IIMAGE_Draw()

Description:

This function draws the image on the screen at the specified position.

Prototype:

```
void IIMAGE_Draw(IImage * pIImage, int x, int y)
```

Parameters:

pIImage	A valid pointer to the IImage Interface object
x	Specifies the x-coordinates of the upper left corner of the destination rectangular area where the image needs to be drawn
y	Specifies the y-coordinates of the upper left corner of the destination rectangular area where the image needs to be drawn

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IIMAGE_DrawFrame()
Return to the List of functions

IIMAGE_DrawFrame()

Description:

This function draws a specific frame (contained within the image) on the screen at the specified position.

Prototype:

```
void IIMAGE_DrawFrame(IImage * pIImage, int nFrame, int x, int y)
```

Parameters:

pIImage	Pointer to the IImage Interface object whose frame needs to be drawn on the screen
nFrame	Specifies the frame that needs to be drawn. Frame Numbers start from 0 (zero).
x	Specifies the x-coordinates of the upper left corner of the destination rectangular area where the frame needs to be drawn
y	Specifies the y-coordinates of the upper left corner of the destination rectangular area where the frame needs to be drawn

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IIMAGE_Draw()
Return to the List of functions

IIMAGE_GetInfo()

Description:

This function retrieves information about the specific image. The information is returned in the AEEImageInfo data structure.

Prototype:

```
void IIMAGE_GetInfo(IImage * pIImage, AEEImageInfo * pi)
```

Parameters:

pIImage	[in]	Pointer to the IImage Interface object whose information needs to be retrieved
pi	[in/out]	On input, this must be a valid pointer to the AEEImageInfo data structure. On output, the data structure is filled with valid information about the image.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEEImageInfo
Return to the List of functions

IIMAGE_HandleEvent()

Description:

This function is used to pass events to an IImage instance.

Prototype:

```
boolean IIMAGE_HandleEvent(IImage * pIImage, AEEEvent evt, uint16 wParam,
uint32 dwParam)
```

Parameters:

pIImage	Pointer to an IImage interface instance that will receive the event
evt	event code
wParam	16-bit event data
dwParam	32-bit event data

Return Value:

TRUE	If the event was processed by the IImage instance
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None

Return to the List of functions

IIMAGE_Notify()

Description:

This function registers a callback function that is invoked when a streaming I/O operation initiated by IIMAGE_SetStream() completes retrieving the image data.

Prototype:

```
void IIMAGE_Notify(IImage * pIImage , PFNIMAGEINFO pfn, void * pUser)
```

Parameters:

pIImage	Pointer to an IImage interface instance whose image data is being retrieved from an asynchronous stream
pfn	pointer to the callback function invoked when the image data is completely available
pUser	pointer to user-specific data that is passed to the callback function

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IIMAGE_SetStream()
Return to the List of functions

IIMAGE_Release()

Description:

This function decrements the reference count of the IImage Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IIMAGE_Release(IImage * pIImage)
```

Parameters:

pIImage	Pointer to the IImage Interface object

Return Value:

reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IIMAGE_AddRef()
Return to the List of functions

IIMAGE_SetParm()

Description:

This function can be used for setting various image related parameters of an IImage Interface object.

The parameter to be set is specified by nParm and can be one of the following: IPARM_SIZE , IPARM_OFFSET, IPARM_CXFRAME, IPARM_NFRAMES, IPARM_ROP, IPARM_RATE, IPARM_OFFSCREEN. The new parameter values are specified using **p1** and **p2**.

If **nParm** is **IPARM_SIZE** , this function is used to specify the size of the image that needs to be used for display or animation purposes. The Image is loaded using the ISHELL_LoadResImage(). If the image to be considered for display must be less than the actual image that has been loaded from the resource file, the size can be changed using this function. In this case, the parameters **p1** and **p2** specify the new width and height of the image, respectively. If either p1 or p2 is negative, the width value or height value reverts to full screen width or full screen height respectively. If both are negative, the full screen is used.

- If **nParm** is **IPARM_OFFSET**, this function is used to specify the offset within the entire image that must be used for displaying the image. In this case, the parameters **p1** and **p2** specify the x and y coordinates of the image offset, respectively. The default offset is (0, 0).

- If **nParm** is **IPARM_CXFRAME**, this function is used to specify the parameter **p1** as the width of each frame. If **p1** is less than or equal to 0 (zero), the frame width is set to be equal to the height of the image. The parameter **p2** is not used.

  NOTE: This image parameter is used only for the formats that normally don't support animation (such as Windows .BMP). This is not used for formats that support animation natively (such as GIF).

- If **nParm** is **IPARM_NFRAMES**, this function is used to specify the parameter **p1** as the number of frames. In this case, the width of each frame is determined by the total width / the number of frames. If **p1** is less than or equal to 0 (zero), then the number of frames is automatically calculated using the formula:

  ```
  nFrames = WidthOfImage / HeightOfImage.
  ```

  The parameter **p2** is not used.

NOTE: that this image parameter is used only for the formats that normally don't support animation (such as Windows .BMP). This is not used for formats that support animation natively (such as GIF).

- If **nParm** is **IPARM_RATE**, this function is used to specify the parameter **p1** as the animation rate in milliseconds. The default animation rate is 150 milliseconds. The parameter **p2** is not used.

  NOTE: that this image parameter is used only for the formats that normally don't support animation (such as Windows .BMP). This is not used for formats that support animation natively (such as GIF).

- If **nParm** is **IPARM_ROP**, this function is used to specify the parameter **p1** as the Raster operation to be used while drawing the image. The default Raster operation is AEE_RO_COPY.The parameter **p2** is not used.

  If **nParm** is **IPARM_OFFSCREEN** , this function is used to specify whether to draw the image to the off-screen buffer.

Prototype:

```
void IIMAGE_SetParm(IImage * pIImage, int nParm, int p1, int p2)
```

Parameters:

pIImage	Pointer to the IImage Interface object
nParm	[IPARM_SIZE | IPARM_OFFSET | IPARM_CXFRAME | IPARM_NFRAMES | IPARM_ROP | IPARM_RATE | IPARM_OFFSCREEN]
p1	**nParm** specific parameter value
p2	**nParm** specific parameter value

Return Value:

None

Comments:

Some of the usage examples of this function are:

If an image of size 50x60 has been loaded, for example, using the ISHELL_LoadResImage(), and you are interested only in a size of 30x40 for display or animation purpose, you need to call:

```
IIMAGE_SetParm(pIImage,IPARM_SIZE,30,40);
```

2 In the above example, if you are interested only in that portion of the image which starts at offset (10,10),ou need to call:

```
IIMAGE_SetParm(pIImage,IPARM_OFFSET,10,10);
```

3 Now, if you want to split the bitmap into three frames, you need to call:

```
IIMAGE_SetParm(pIImage,IPARM_NFRAMES,3,0);
```

Side Effects:

None

See Also:

AEE IImage Parameters
Return to the List of functions

IIMAGE_SetStream()

Description:

This function allows an IAStream interface to be associated with an IImage Interface object to allow image data to be streamed in from a file or socket.

Prototype:

```
void IIMAGE_SetStream (IImage * pIImage, IAStream * pIAStream)
```

Parameters:

pIImage	Pointer to an IImage Interface object that receives the streaming image data input
ps	Pointer to an instance of a class that implements the IAStream interface (e.g., IFile or ISocket)

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IASTREAM_Read()
IASTREAM_Readable()
Return to the List of functions

IIMAGE_Start()

Description:

This function animates the given image. It cycles through the individual frames of the image. Each frame is displayed at the coordinates specified by the parameter x and y. The animation timer (that is, time-interval between displaying of two successive frames) is 1500 milliseconds. The animation continues until IIMAGE_Stop() is called.

Prototype:

```
void IIMAGE_Start(IImage * pIImage, int x, int y)
```

Parameters:

pIImage	Pointer to the IImage Interface object that needs to be animated
x	Specifies the x-coordinates of the upper left corner of the destination rectangular area where the frame needs to be drawn
y	Specifies the y-coordinates of the upper left corner of the destination rectangular area where the frame needs to be drawn

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IIMAGE_Stop()
Return to the List of functions

IIMAGE_Stop()

Description:

This function stops the animation of the image that has been started using the IIMAGE_Start().

Prototype:

```
void IIMAGE_Stop(IImage * pIImage)
```

Parameters:

pIImage	Pointer to the IImage Interface object whose animation needs to be stopped

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IIMAGE_Start()
Return to the List of functions

IMemAStream Interface

The IMemAStream interface extends the IAStream Interface to allow a specified memory chunk to be read as a stream. An instance of the IMemAStream can be created using the ISHELL_CreateInstance() Function (with class ID AEECLSID_MEMASTREAM). The specified memory chunk is freed when the IMemAStream Interface object is released.

In addition to the standard IBase Interface functions, AddRef() and Release(), and the standard IAStream Interface functions, Readable(), Read(), and Cancel(), the IMemAStream Interface includes the Set function.

List of functions

Functions in this interface include:

IMEMASTREAM_AddRef()

IMEMASTREAM_Cancel()

IMEMASTREAM_Read()

IMEMASTREAM_Readable()

IMEMASTREAM_Release()

IMEMASTREAM_Set()

Return to the Contents

IMEMASTREAM_AddRef()

Description:

This function increments the reference count of IMemAStream Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IMEMASTREAM_Release().

Prototype:

```
uint32 IMEMASTREAM_AddRef(IMemAstream * po)
```

Parameters:

po	Pointer to the IMemAStream Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IMEMASTREAM_Release()
Return to the List of functions

IMEMASTREAM_Cancel()

Description:

This function allows you to cancel a notification function previously registered using IMEMASTREAM_Readable().

Prototype:

```
void IMEMASTREAM_Cancel(IMemAStream *pIStream, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIStream	Pointer to the IMemAStream Interface object
pfn	Pointer to the user function that was set as the notification function in IMEMASTREAM_Readable().
pUser	Pointer to user data that was provided as a parameter when calling pfn

Return Value:

None.

Comments:

None

Side Effects:

None

See Also:

IMEMASTREAM_Readable()
Return to the List of functions

IMEMASTREAM_Read()

Description:

This function is used to read the number of bytes, nBytes, from the stream, pIMemAStream, into the destination buffer pDest. If fewer than nBytes are available in the stream, the function returns the actual number of bytes read.

This function attempts to read data from the buffer that has been set into the IMemAStream Interface object using IMEMASTREAM_Set(). Hence, prior to calling this function, the function IMEMASTREAM_Set() must be invoked to set a valid memory buffer into the IMemAStream Interface object.

Prototype:

```
int32 IMEMASTREAM_Read(IMemAStream *pIStream, void * pDest, uint32 nBytes)
```

Parameters:

pIStream	Pointer to the IMemAStream Interface object
pDest	Pointer to the buffer into which the data will be read
NBytes	Number of bytes to read from the stream

Return Value:

Number of bytes read

Comments:

If nBytes is greater than the size of the data available in the buffer, which is set using IMEMASTREAM_Set(), then only the available data is read, and this size is returned.

Side Effects:

None

See Also:

IMEMASTREAM_Set()
Return to the List of functions

IMEMASTREAM_Readable()

Description:

This function allows you to specify a function pfn used by the IMemAStream Interface object to notify you when the stream has available data to be read. For the IMemAStream Interface object, the callback is attempted immediately since the data is read from a memory buffer invoked using the Set() function.

Prototype:

```
void IMEMASTREAM_Readable(IMemAStream *pIStream, PFNNOTIFY pfn, void *
pUser)
```

Parameters:

pIStream	Pointer to the IMemAStream Interface object
pfn	Pointer to the user function that must be called by the IMemAStream Interface object when data is available for reading
pUser	Pointer to user data that must be used as a parameter when calling pfn

Return Value:

None.

Comments:

None

Side Effects:

None

See Also:

Return to the List of functions

IMEMASTREAM_Release()

Description:

This function decrements the reference count of IMemAStream Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero). When the reference count reaches 0 (zero), the memory buffer, previously set using the call to IMEMASTREAM_Set(), is also freed.

Prototype:

```
uint32 IMEMASTREAM_Release(IMemAstream * po)
```

Parameters:

po	Pointer to the IMemAStream Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object is freed and no longer valid

Comments:

None

Side Effects:

None

See Also:

IMEMASTREAM_Set()
Return to the List of functions

IMEMASTREAM_Set()

Description:

This function allows you to set the memory chunk that needs to be read as a stream. An instance of the IMemAStream Interface object must already exist.

The responsibility of freeing the buffer pBuff lies with the IMemAStream Interface. You must not free this buffer. The buffer is freed when either of the following two actions occur:

- The IMemAStream Interface object is released using IMEMASTREAM_Release()
- A subsequent call to IMEMASTREAM_Set() is attempted with another buffer

If a buffer has already been set into the stream using a previous call to this function, that buffer is freed before setting the new buffer.

> **CAUTION:** It is dangerous to attempt two consecutive calls to IMEMASTREAM_Set() with the same buffer.

Prototype:

```
void IMEMASTREAM_Set(
          IMemAStream * pIMemAStream,
          byte * pBuff,
          uint32 dwSize,
          uint32 dwOffset,
          boolean bSysMem
         )
```

Parameters:

pIMemAStream	Pointer to the IMemAStream Interface
pBuff	Pointer to the memory chunk that needs to be read as a stream
dwSize	Size of the memory chunk

dwOffset	Offset from the beginning of the memory chunk to be set as the start data for the memory stream
bSysMem	Flag to specify if the memory for the buffer pBuff belongs to the user area or the system memory. This flag is used to decide whether the routines FREE() or SYSFREE() must be used by the IMemAStream Interface object to free the buffer when this object is released or when another Set() is made. For example, if you perform a MALLOC() to allocate the buffer, then bSysMem must be set to FALSE.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IMEMASTREAM_Release()
Return to the List of functions

Menu controls allow the device user to make a selection from a list of items. The UP, DOWN, LEFT and RIGHT arrow keys are used to identify the currently selected menu item, which appears highlighted on the screen. When the user presses the SELECT key and command sending is enabled (see later in this section), an **EVT_COMMAND** is sent to the application or dialog that created the menu, which includes the identifier of the currently selected item. There are four types of menu controls (you select the type you want by specifying its ClassID when you create an instance of the menu control):

- A standard menu control (ClassID **AEECLSID_MENUCTL**) displays one menu item per row on the screen, with each row containing the item's bitmap icon and/or text string. If all the items do not fit on the screen, you can use the UP and DOWN arrow keys to scroll the menu up or down.
- A list control (ClassID **AEECLSID_LISTCTL**) displays only the currently selected menu item on the screen. This type of menu is useful in applications where the available screen real estate is limited. Items in a list control menu contain only text (there are no bitmap icons). You use the UP and DOWN arrow keys to navigate to the desired menu selection.
- A SoftKey menu control (ClassID **AEECLSID_SOFTKEYCTL**) displays the menu items side by side along the bottom portion of the screen. You use the LEFT and RIGHT arrow keys to designate the selected menu item.
- An icon-view menu control (ClassID **AEECLSID_ICONVIEWCTL**) uses a bitmap icon to represent each menu item. The bitmap icons are displayed in one or more rows on the screen, and the arrow keys are used to move between rows and between the icons in each row. The text string corresponding to the currently selected item appears at the bottom of the screen.

As mentioned above, IMENUCTL_HandleEvent() handles the UP, DOWN, LEFT and RIGHT arrow keys and the SELECT key. If a calendar view menu is specified (see later in this section), a standard menu control also handles, the device number keys (**AVK_0** through **AVK_9**), which are used to enter the time of a calendar appointment. Except for SoftKey menus, a menu control sends a control tabbing event (**EVT_CTL_TAB**) when the user presses the LEFT and RIGHT keys. You can use control tabbing to move between controls in a multicontrol screen (if your menu control is part of a dialog, the dialog intercepts the control tabbing events and changes control focus appropriately).

Menu controls support a number of properties that can be set with IMENUCTL_SetProperties() (the property names are the names of the bitmask constants you use to get and set the properties):

- **MP_WRAPSCROLL** causes scrolling to wrap around, for example, the first menu item becomes selected when the DOWN key is pressed while the last menu item is selected. This property is always set for soft key and list control menus.
- **MP_NO_ARROWS** applies only to icon-view menus for which the **MP_ICON_SINGLE_FRAME** property is set. It prevents the drawing of arrows on either side of the item text.
- **MP_NO_REDRAW** suppresses the re-drawing of the menu each time the selected item of a menu changes or the menu is set to active.
- **MP_MAXSOFTKEYITEMS** increases to 12, the number of SoftKey menu items that can be displayed on the screen at once. By default at most three items are displayed.
- **MP_CALENDAR** allows a standard menu to be used as a calendar program. Horizontal lines are drawn between the menu items, and each item represents a calendar appointment at a particular hour of the day. The device's number keys are used to select the hour of an appointment, and the user can enter menu-item text describing the appointment.
- **MP_AUTOSCROLLTIME** automatically scrolls a calendar-view menu so that the entry corresponding to the current time appears on the screen when the calendar is displayed.
- **MP_ICON_TEXT_TOP** causes the text string of the currently selected item in an icon-view menu to appear at the top of the screen instead of the bottom.
- **MP_ICON_SINGLE_FRAME** displays only the icon of the currently selected menu item on the screen (by default, an icon-view menu displays all the icons in rows and columns and highlights the selected one).
- **MP_UNDERLINE_TITLE** causes the menu's title to be underlined.

Menu controls implement several functions in addition to those in the IControl Interface. IMENUCTL_SetTitle() is used to specify a value for the menu's title, which is drawn at the top of its rectangle. IMENUCTL_EnableCommand() enables or disables the sending of **EVT_COMMAND** events to your application when the user presses the SELECT key (command sending is enabled by default). IMENUCTL_SetStyle is used to customize the appearance of the selected item and of the non-selected items in a menu, including special pixel borders, padding space around each item, and other appearance elements.

After creating a menu, you must specify the items that the menu contains. The function IMENUCTL_AddItem() is used to add items to the menu that do not contain bitmap icons. When calling this function, you specify the item's text string (either from a resource file or defined in your code), an integer item ID, and an optional double-word data pointer. When the user selects the item, your application's IAPPLET_HandleEvent() function is called with an **EVT_COMMAND** event; the item ID and double-word pointer are specified as the **wParam** and **dwParam** parameters in this function call. The function IMENUCTL_AddItemEx() is an extended version of IMENUCTL_AddItem() that provides additional information about the added menu item, including its bitmap icon and the font to be used to display its text string. IMENUCTL_DeleteItem() deletes a menu item with a particular ID, and IMENUCTL_DeleteAll() deletes all the items from a menu.

IMENUCTL_GetSel() returns the item ID of the currently selected menu item. You can use this function when the user's selection is not obtained by pressing the SELECT key (for example, you may wish to retrieve the user's selection when the dialog containing the menu control is terminated). IMENUCTL_SetSel() specifies the item ID that to be the currently selected one. The menu control normally determines this based on user presses of the arrow keys, but you can use

IMENUCTL_SetSel() to designate the initially selected item when the menu is first displayed. IMENUCTL_GetItemData() retrieves the double-word data pointer of a menu item. IMENUCTL_SetItemText() is used to change the text string of an existing menu item.

IMENUCTL_SetItemTime() sets the start time and duration associated with an item in a calendar menu (see above), and IMENUCTL_GetItemTime() retrieves the time information of a calendar-menu item.

To create and use a menu control, perform the following steps

1. Call ISHELL_CreateInstance() to create the menu control instance and obtain an interface pointer to it, specifying which of the four types of menu control you would like by its ClassID.

2. Call IMENUCTL_SetTitle() to specify a menu title if desired, and call IMENUCTL_AddItem() or IMENUCTL_AddItemEx() for each item to be added to the menu.

3. Call IMENUCTL_SetRect() to define the screen rectangle in which the menu is to be drawn.

4. Call IMENUCTL_SetProperties() if needed to set any of the menu control properties, and call IMENUCTL_SetStyle if you would like to customize the appearance of your menu items.

5. When you have completely specified the contents and properties of the menu control, call IMENUCTL_SetActive() to draw the control on the screen and enable it to receive key

events from the user. While the menu control is active, your application's IAPPLET_HandleEvent() function must call IMENUCTL_HandleEvent() to pass all handled key events to the menu control for processing.

6 Determine how you are to obtain the user's menu selection. If you process the selection when the user presses the SELECT key, the IAPPLET_HandleEvent() function can contain logic to handle the selection of each menu item when the **EVT_COMMAND** is received. If your application receives **EVT_COMMAND** events from more than one control, be sure that the item IDs passed in the wParam parameter are unique. If you will retrieve the user's selection at some other time, you must call IMENUCTL_GetSel() and/or IMENUCTL_GetItemData() at this time to access the currently selected menu item and its double word data (if any).

7 When you no longer need the menu control, call IMENUCTL_Release() to free it.

List of functions

Functions in this interface include:

IMENUCTL_AddRef()

IMENUCTL_AddItem()

IMENUCTL_AddItemEx()

IMENUCTL_DeleteAll()

IMENUCTL_DeleteItem()

IMENUCTL_EnableCommand()

IMENUCTL_GetItemData()

IMENUCTL_GetItemTime()

IMENUCTL_GetProperties()

IMENUCTL_GetRect()

IMENUCTL_GetSel()

IMENUCTL_HandleEvent()

IMENUCTL_IsActive()

IMENUCTL_Redraw()

IMENUCTL_Release()

IMENUCTL_Reset()

IMENUCTL_SetActive()

IMENUCTL_SetColors()

IMENUCTL_SetItemText()

IMENUCTL_SetItemTime()

IMENUCTL_SetProperties()

IMENUCTL_SetRect()

IMENUCTL_SetSel()

IMENUCTL_SetStyle

IMENUCTL_SetTitle()

Return to the Contents

IMENUCTL_AddRef()

Description:

This function increments the reference count of the IMenuCtl Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IMENUCTL_Release()

Prototype:

```
uint32 IMENUCTL_AddRef(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IMENUCTL_Release()
Return to the List of functions

IMENUCTL_AddItem()

Description:

This function adds a new menu item to a menu. The text string identifying the menu can be specified from a resource file or using the **pText** parameter. If **pText** parameter is a valid string, it uses this string as item name. If **pText** is NULL, it reads the string corresponding to the given resource identifier, **wResID**, from resource file and sets it as item name. If the text string and the resource information are invalid, **IMENUCTL_AddItem()** fails. If the text string in the **pText** field and the resource file information are both valid, the **pText** parameter field take precedence, and is used for the menu item name. If the object identified by IMenuCtl Interface is soft key menu, this function also determines the extent of new item in x-axis and maximum number of items able to be displayed on soft key bar. The **IData** is used to store a double-word data value associated with the menu item to be created. IMENUCTL_GetItemData() can be used to get back the parameter value.

Prototype:

```
boolean IMENUCTL_AddItem(IMenuCtl * pIMenuCtl, const char * pszResFile,
uint16 wResID, uint16 nItemID, AECHAR * pText, uint32 lData)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
pszResFile	NULL terminated string that contains the resource file name
wResID	Resource identifier that identifies a text string resource
nItemID	Item identifier that uniquely identifies a menu item
pText	Null terminated string that contains the menu item name
IData	Item data associated with the menu item

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

This function does not automatically update the screen. For the new menu item to take effect on the device screen, use IMENUCTL_Redraw(). If the menu is to be activated subsequent to adding menu items, IMENUCTL_Redraw() does not need to used since IMENUCTL_SetActive() updates the screen with the new menu items. IMENUCTL_DeleteItem()can be used to remove an item from the menu

Side Effects:

When the newly added menu item is activated by a user, **nItemID** and **lData** values are sent to the applet event handling function in the short-data and long-data fields respectively.

See Also:

IMENUCTL_DeleteItem()
IMENUCTL_Redraw()
IMENUCTL_SetActive()
Return to the List of functions

IMENUCTL_AddItemEx()

Description:

This function adds an item to a menu, list or icon view. The new item's properties are indicated by the CtlAddItem * parameter.

If pai->pText parameter is a valid string, it uses this string as item name.

If pai->pText is NULL, it reads the string corresponding to the given resource identifier associated with pai->wText and pai->pszResText. If this fails, IMENUCTL_AddItemEx() returns FALSE.

if pai->pImage is a valid IImage * , the function calls IIMAGE_AddRef() and uses the image. This allows the caller to release the image object.

If the text string in the pai->pText field and the resource file information are both valid, the pai->pText parameter field take precedence, and is used for the menu item name.

If the pai->pImage is NULL and the pai->wImage is specified, the function attempts to load the image from the specified resource.

If the image fails to load, the function returns FALSE.

Prototype:

```
boolean IMENUCTL_AddItemEx(IMenuCtl * pIMenuCtl, CtlAddItem * pai)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
pai	Pointer to CtlAddItem structure

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

This function does not automatically update the screen. For the new menu item to take effect on the device screen, use IMENUCTL_Redraw(). If the menu is to be activated subsequent to adding menu items, IMENUCTL_Redraw() does not need to used since IMENUCTL_SetActive() updates the screen with the new menu items.

Side Effects:

When the newly added menu item is activated by a user, **nItemID** and **lData** values are sent to the applet event handling function in the long-data and short-data fields, respectively.

See Also:

IMENUCTL_Redraw()
IMENUCTL_AddItem()
IMENUCTL_AddItemEx()
Return to the List of functions

IMENUCTL_DeleteAll()

Description:

This function deletes all the menu items from a menu control object.

Prototype:

```
boolean IMENUCTL_DeleteAll(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface object

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

This function does not update the device screen when invoked. To update the device screen use IMENUCTL_Redraw().

Side Effects:

None

See Also:

None
Return to the List of functions

IMENUCTL_DeleteItem()

Description:

This function deletes a menu item from the menu control object. The **nItemID** field identifies the menu item to be deleted.

Prototype:

```
boolean IMENUCTL_DeleteItem(IMenuCtl * pIMenuCtl,uint16 nItemID)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface object
nItemID	Menu item identifier that is used when the menu item was added to the menu

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

This function does not update the device screen when invoked. To update the device screen use IMENUCTL_Redraw().

Side Effects:

None

See Also:

None
Return to the List of functions

IMENUCTL_EnableCommand()

Description:

This function is used to enable sending of specified command by the menu control object to the AEE Shell upon receiving the event generated by pressing SELECT key.

Prototype:

```
void IMENUCTL_EnableCommand(IMenuCtl * pIMenuCtl, boolean bEnable)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface of the menu control object
bEnable	Enable/Disable flag

Return Value:

None

Comments:

The SELECT key is located on different physical locations of a device depending on the device manufacturer and/or model. When this key is pressed the event received by the applet is of type AVK_SELECT.

Side Effects:

None

See Also:

IMENUCTL_HandleEvent()
Return to the List of functions

IMENUCTL_GetItemData()

Description:

This function gets the data associated with a menu item in the given menu control object. The **nItemID** parameter identifies the menu item, of which the data is requested.

Prototype:

```
boolean IMENUCTL_GetItemData(IMenuCtl * pIMenuCtl, uint16 nItemID, uint32 *
plData)
```

Parameters:

pIMenuCtl	[in]	Pointer to IMenuCtl Interface object
nItemID	[in]	Menu item identifier that is used when this menu item is added to the menu
plData	[out]	Placeholder for item data

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

The data that is retrieved is associated with a menu when the menu item is added. See IMENUCTL_AddItem() for more information on menu item data.

Side Effects:

None

See Also:

IMENUCTL_AddItem()
Return to the List of functions

IMENUCTL_GetItemTime()

Description:

If the menu control is calendar, it retrieves the start time and duration (difference between end time and start time) of the menu item specified by the menu item Id.

Prototype:

```
int IMENUCTL_GetItemTime(IMenuCtl * pIMenuCtl, uint16 nItemID, uint16 *
pwDuration)
```

Parameters:

pIMenuCtl	[in]	Pointer to the IMenuCtl Interface object
nItemID	[in]	Menu item Identifier
pwDuration	[out]	Duration in minutes

Return Value:

Start time	Which is set in IMENUCTL_SetItemTime()
-1	If the menu item cannot be found

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_SetItemTime()
Return to the List of functions

IMENUCTL_GetProperties()

Description:

This function is used to retrieve the menu control-specific properties.

or flags.

Prototype:

```
uint32 IMENUCTL_GetProperties(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

Following are 32-bit properties defined for the menu control object:

MP_WRAPSCROLL	If set, wrap when scrolling off the end of screen (only applicable to SoftKey and List controls)
MP_NO_ARROWS	If set, no arrows even if scroll is possible (Icon View)
MP_MAXSOFTKEYITEMS	Shows maximum number of soft key items per screen
MP_NO_REDRAW	If set, IMENUCTL_Redraw() function is not internally called in IMENUCTL_SetActive() or when changing selection
MP_PAGESCROLL	If set scroll by pages, otherwise smooth

Following are attributes defined only for icon view of the menu control object:

MP_ICON_TEXT_TOP	Icon View: Text at top
MP_ICON_TEXT_NOSEL	Icon View: Non reverse-video text
MP_ICON_SINGLE_FRAME	Icon View: Single Frame
MP_UNDERLINE_TITLE	Menu: Underline title

Following are two attributes defined only for calendar event list view of the menu control object:

MP_CALENDAR	If set, specifies that menu control object is in calendar event list view
MP_AUTOSCROLLTIME	If set, specifies that auto-scroll if in calendar list view

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_SetProperties()
Return to the List of functions

IMENUCTL_GetRect()

Description:

This function returns the control rectangle value of the menu control object. This is particularly useful after a control object is created to determine its optimal/default size and positions.

Prototype:

```
void IMENUCTL_GetRect(IMenuCtl * pIMenuCtl, AEERect * prc)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
prc	Rectangle to be filled with the rectangle coordinates of the menu control

Return Value:

None

Comments:

The menu control object is displayed in a rectangle area of the screen specified by the coordinates stored in the menu control object. These coordinates by default correspond to the coordinates of the whole display screen when the menu control object is instantiated. Subsequently these coordinates can be changed by using the IMENUCTL_SetRect() function to encompass any section of the display screen. The IMENUCTL_GetRect() function is used to retrieve the current specifications of the control rectangle.

Side Effects:

None

See Also:

IMENUCTL_SetRect()
Return to the List of functions

IMENUCTL_GetSel()

Description:

This function gets the menu item identifier of the menu control objects current selection.

Prototype:

```
uint16 IMENUCTL_GetSel(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

Returns the ID of the current menu item selection	**Of the** IMenuCtl Interface object

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_SetSel()
Return to the List of functions

IMENUCTL_HandleEvent()

Description:

This function is used by menu control object to handle events received by it. An active menu control object handles key press events as well as set title and add item events received by it whereas an inactive menu control object handles only set title and add item events. The typical key press events processed by the menu control object are the press of UP, DOWN, LEFT and RIGHT keys. If command sending is enabled for the menu control object, upon receiving event generated by the press of “select” key, it sends a command event to the AEE Shell.

Prototype:

```
boolean IMENUCTL_HandleEvent(IMenuCtl * pIMenuCtl, AEEEvent evt, uint16 wp,
uint32 dwp)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
evt	Event code
wp	16-bit event data
dwp	32-bit event data

Return Value:

TRUE	If the event was processed by the IMenuCtl Interface object
FALSE	If the event was not processed

Comments:

The menu control event handler is used in applets to handle menu related inputs from a user. for example, when a user presses the down arrow key to move from one menu item selection to another, this event is handled by the menu control event handler. In this case the applet event handler function receives the corresponding event. The applet event handler at this time can call the menu control object event handler to process this event. If the applet event handler does not call the menu control event handler, this event remains unprocessed. The SELECT key is located on different physical locations of a device depending on the device manufacturer and/or model. When this key is pressed the event word parameter **wParam** received by the applet is of type AVK_SELECT.

Side Effects:

None

See Also:

IMENUCTL_EnableCommand()
Return to the List of functions

IMENUCTL_IsActive()

Description:

This function returns the active/inactive state of the menu control object.

Prototype:

```
boolean IMENUCTL_IsActive(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

TRUE	If the menu control is active
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_SetActive()
Return to the List of functions

IMENUCTL_Redraw()

Description:

This function instructs the menu control object to redraw its contents. The menu control object does not redraw its contents every time the underlying data behind the menu control changes. This allows several data updates to occur while minimizing screen flashes. For example, several changes can be made to the contents of the menu control object with no visible effect until the Redraw function is called.

Prototype:

```
boolean IMENUCTL_Redraw(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

TRUE	If the menu control was redrawn
FALSE	If unsuccessful

Comments:

The menu control object is displayed in a rectangle area of the screen specified by the coordinates stored in the menu control object. These coordinates by default correspond to the coordinates of the whole display screen when the menu control object is instantiated. Subsequently these coordinates can be changed by using the IMENUCTL_SetRect() to encompass any section of the display screen. The IMENUCTL_Redraw() function only redraws the menu in the area of the screen bounded by the current specifications of the menu control object rectangle coordinates.

Side Effects:

None

See Also:

None
Return to the List of functions

IMENUCTL_Release()

Description:

This function decrements the reference count of the IMenuCtl Interface object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 IMENUCTL_Release(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

Returns the updated reference count.

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_AddRef()
Return to the List of functions

IMENUCTL_Reset()

Description:

This function instructs the menu control to reset (free/delete) its contents as well as to immediately leave active/focus mode. Resetting of the menu control object removes the menu items from the menu and removes the title associated with the menu.

Prototype:

```
void IMENUCTL_Reset(IMenuCtl * pIMenuCtl)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object

Return Value:

None

Comments:

A reset of a menu control object does not update the device screen, hence, all the graphics associated with the menu remains on the device screen. To remove the menu from the device screen use IMENUCTL_Redraw() of the menu control object.

Side Effects:

A Reset menu control object does not handle any events received from a user, such as UP, DOWN, LEFT, and RIGHT arrow key presses.

See Also:

IMENUCTL_SetActive()
IMENUCTL_Redraw()
Return to the List of functions

IMENUCTL_SetActive()

Description:

This function is used to make a menu control object active or inactive. Only an active menu control object handles the event sent to it. Inactive menu control object just ignores the events.

Prototype:

```
void IMENUCTL_SetActive(IMenuCtl * pIMenuCtl,boolean bActive)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
bActive	Boolean flag that specifies whether to activate (TRUE) or deactivate (FALSE) the menu control object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_IsActive()
Return to the List of functions

IMENUCTL_SetColors()

Description:

By default, the color of menu elements are determined by entries in the system color table defined by the handset manufacturer. These entries include:

CLR_SYS_ITEM	Background color of unselected items
CLR_SYS_ITEM_TEXT	Text color for unselected items and Arrows
CLR_SYS_ITEM_SEL	Background color for selected items
CLR_SYS_ITEM_SEL_TEXT	Text color for selected items
CLR_USER_FRAME	Simple frame color
CLR_SYS_SCROLLBAR	Scrollbar frame color
CLR_SYS_SCROLLBAR_FILL	Fill color of scrollbar
CLR_SYS_TITLE	Background of title text
CLR_SYS_TITLE_TEXT	Color of title text

This function allows the caller to override most of these settings for the menu control objects.

Prototype:

```
void IMENUCTL_SetColors(IMenuCtl * pIMenuCtl, AEEMenuColors * pc);
```

Parameters:

pIMenuCtl	Pointer to the menu control interface object.
pc	Pointer to structure containing both: Bitmask to indicate the item whose color needs to be changed, and Color values

Return Value:

None

Comments:

Passing NULL to this function will reset the menu colors. The caller cannot override the system settings for 3-D framed objects.

Side Effects:

None

See Also:

AEEMenuColors
Return to the List of functions

IMENUCTL_SetItemText()

Description:

This function sets a new menu item text name, given the menu item Id. The new text is passed in through a string pointer or through a resource file name and an Id. If both are provided, the string pointer takes precedence.

Prototype:

```
void IMENUCTL_SetItemText(IMenuCtl * pIMenuCtl, uint16 nItemID, const char
* pszResFile, uint16 wResID, TCHAR * pText)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface object
nItemID	Menu item identifier
pszResFile	NULL terminated string containing the resource file name
wResID	Resource Id
pText	New menu item text in string form

Return Value:

None

Comments:

This function can be used to overwrite a menu name of a currently existing menu item. In this case, the `nItemID` parameter contains the currently existing menu item ID.

Side Effects:

None

See Also:

None
Return to the List of functions

IMENUCTL_SetItemTime()

Description:

If the menu control is calendar, it sets a new time of the menu item specified by the menu item Id.

Prototype:

```
void IMENUCTL_SetItemTime(IMenuCtl * pIMenuCtl, uint16 nItemID, uint16
wMinStart, uint16 wDuration)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface object
nItemID	Menu item identifier
wMinStart	Start time in minutes
wDuration	Duration in minutes

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_GetItemTime()
Return to the List of functions

IMENUCTL_SetProperties()

Description:

This function sets menu control-specific properties or flags. These properties define the behavior of the menu control object.

Prototype:

```
void IMENUCTL_SetProperties(IMenuCtl * pIMenuCtl, uint32 dwProps)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
dwProps	32-bit set of flags/properties

Following are 32-bit properties defined for the menu control object:

MP_WRAPSCROLL	If set, wrap when scrolling off the end of screen (only applicable to SoftKey and List controls)
MP_NO_ARROWS	If set, no arrows even if scroll is possible (IconView)
MP_MAXSOFTKEYITEMS	Shows maximum number of soft key items per screen
MP_NO_REDRAW	If set, IMENUCTL_Redraw() function is not internally called in IMENUCTL_SetActive() or when changing selection

Following are attributes defined only for icon view of the menu control object.

MP_ICON_TEXT_TOP	Icon View: Text at top
MP_ICON_SINGLE_FRAME	Icon View: Single Frame
MP_UNDERLINE_TITLE	Menu: Underline title

Following are two attributes defined only for calendar event list view of the menu control object:

MP_CALENDAR	If set, specifies that menu control object is in calendar event list view.
MP_AUTOSCROLLTIME	If set, specifies that auto-scroll if in calendar list view.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_GetProperties()
Return to the List of functions

IMENUCTL_SetRect()

Description:

This function is used to set the coordinates of the control rectangle of the menu control object. The control rectangle of the menu control object represents the area on the device screen where the menu is drawn. The values of the control rectangle are stored within the menu control object. This function is used to change the value of the control rectangle to a new one.

Prototype:

```
void IMENUCTL_SetRect(IMenuCtl * pIMenuCtl, const AEERect * prc)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
prc	Control rectangle for the menu control object

This rectangle specifies the coordinates on the device screen where the menu is to be drawn.

Return Value:

None

Comments:

The coordinates stored in the control rectangle of the menu control object by default corresponds to the whole device display screen. This function can be used to set the control rectangle coordinates to a new value. If no other menu draw operations follow the invocation of this function, the IMENUCTL_Redraw() needs to be used for the new control rectangle coordinates to take effect. In the case of the soft key menu the default control rectangle encompasses the whole device screen, where the soft key menu is draw at the bottom of the device screen. When the IMENUCTL_SetRect() is used to set the control rectangle to a new area, such as the top half of the device screen, the soft key menu is placed at the bottom of new control rectangle area. In this case the soft key menu is in the middle of the screen, since the bottom of the new control rectangle corresponds to the middle of the device screen.

Side Effects:

None

See Also:

IMENUCTL_GetRect()
Return to the List of functions

IMENUCTL_SetSel()

Description:

This function makes a menu item the current selection of the menu control object and requests a redraw if the MP_NO_REDRAW property is not set.

Prototype:

```
void IMENUCTL_SetSel(IMenuCtl * pIMenuCtl, uint16 nItemID)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
nItemID	Menu item identifier that is used when this menu item is added to the menu

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IMENUCTL_GetSel()
Return to the List of functions

IMENUCTL_SetStyle

Description:

This function sets the display style for the menu items. Different styles can be set for normal (not selected)/selected menu items. This can also be used to change the menu cursor size.

Prototype:

```
void IMENUCTL_SetStyle(IMenuCtl * pIMenuCtl, AEEItemStyle * pNormal,
AEEItemStyle * pSel)
```

Parameters:

pIMenuCtl	Pointer to the IMenuCtl Interface object
pNormal	Style for normal menu items
pSel	Style for selected menu items

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None.
Return to the List of functions

IMENUCTL_SetTitle()

Description:

This function is used to set the title of a menu control object. The text string identifying the title can be specified from a resource file or using the **pText** parameter. If **pText** parameter is a valid string, it uses this string as title name. If **pText** is NULL, it reads the string corresponding to the given resource identifier, **wResID**, from resource file and sets it as title name. If the text string and the resource information are invalid IMENUCTL_SetTitle() fails. If the title string in the **pText** field and the resource file information are both valid, the **pText** parameter field take precedence, and is used for the menu title.

Prototype:

```
boolean IMENUCTL_SetTitle(IMenuCtl * pIMenuCtl, const char * pszResFile,
uint16 wResID, TCHAR * pText)
```

Parameters:

pIMenuCtl	Pointer to IMenuCtl Interface object
pszResFile	NULL terminated string containing resource file name
wResID	Resource identifier
pText	NULL terminated character string to be used for the menu title

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

This function does not update the device screen when invoked. For the new title to take effect on the device screen, use IMENUCTL_Redraw.

The title of a soft key control menu is not displayed.

Side Effects:

None

See Also:

IMENUCTL_Redraw()
Return to the List of functions

IModule Interface

The IModule Interface provides a mechanism for controlling access to a grouping of associated applets or components. The module provides a single point of entry for the AEE Shell to request classes owned by the module. In most cases, a module exposes a single applet or shared-component class. However, the use of the IModule Interface allows for modules to expose a wide variety of classes without fixed entry-points to expose each such class. Further, the module can serve as a base object for any associated objects. This may allow them to share memory, and so forth via the private implementation of the module class.

List of functions

Functions in this interface include:

IMODULE_AddRef()

IMODULE_CreateInstance()

IMODULE_FreeResources()

IMODULE_Release()

Return to the Contents

IMODULE_AddRef()

Description:

This function increments the reference count of the IModule Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See IMODULE_Release().

Prototype:

```
uint32 IMODULE_AddRef(IModule * pIModule)
```

Parameters:

pIModule	Pointer to the IModule Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

IMODULE_Release()
Return to the List of functions

IMODULE_CreateInstance()

Description:

IMODULE_CreateInstance() provides the mechanism for the AEE to request classes on an as-needed basis from the module. Upon successful creation of the requested object class, the module can return the class object with a positive reference count.

NOTE: The requested class must be implemented in accordance with the class definition specified for the class.

Prototype:

```
int IMODULE_CreateInstance(IModule * pIModule,IShell * pIShell,AEECLSID
ClsId,void * * ppObj)
```

Parameters:

pIModule	[in]	Pointer to theIModule Interface object
pIShell	[in]	Pointer to the IShell Interface object
ClsId	[in]	Requested ClassID exposed by the module
ppObj	[out]	Returned object. Filled by the IMODULE_CreateInstance() function

Return Value:

SUCCESS	Object class was created
ENOMEMORY	Insufficient memory
EBADCLASS	Requested class is unsupported

Comments:

Object must be returned with a positive reference count

Side Effects:

None

See Also:

None
Return to the List of functions

IMODULE_FreeResources()

Description:

This function is called by the AEE Shell when the shell or device detects a low memory or storage condition. This can include low RAM or Flash/File storage. The module is passed a pointer to IHeap Interface and/or IFileMgr Interface depending upon the specific condition involved. The module can release any unused RAM/File storage under this condition.

Prototype:

```
void IMODULE_FreeResources(IModule * pIModule, IHeap * ph, IFileMgr * pfm)
```

Parameters:

pIModule	Pointer to the IModule Interface object
ph	Pointer to IHeap Interface object
pfm	Pointer to IFileMgr Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

IMODULE_Release()

Description:

This function decrements the reference count of the IModule Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 IMODULE_Release(IModule * pIModule)
```

Parameters:

pIModule	Pointer to the IModule Interface object

Return Value:

reference count	The decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

IMODULE_AddRef()
Return to the List of functions

INetMgr Interface

The INetMgr Interface functions provide mechanisms to get and set the parameters associated with the network subsystem of the device. It also provides a means to create multiple I socket interfaces which use TCP or UDP to transmit and receive data over the network The ISocket Interface contains functions that connect the sockets, read and write data over the connections, and close the sockets.

> **CAUTION:** Your application must have a privilege level of **Network** or **All** to be able to invoke the functions in this interface.

The function INETMGR_OpenSocket() is used to open a TCP or UDP socket. If successful, this function returns a pointer to an instance of the ISocket Interface. You can use this pointer to call the ISOCKET_Connect() function to specify the destination IP address and port number to which the socket is to be connected. When the first socket is connected on the device, BREW brings up the network subsystem, including the CDMA physical layer, Radio Link Protocol (RLP) and Point-to-Point Protocol (PPP) connections. These connections are shared by all of the device's connected sockets. You close a socket by calling the ISOCKET_Release(). When the last connected socket on the device is closed, BREW terminates the network subsystem connections after a specified linger time.

The function INETMGR_GetHostByName() takes as input the name of a host (for example, brew.qualcomm.com) and returns a list of IP addresses that can be used to access that host. Because retrieving this list requires network communication, INETMGR_GetHostByName() is asynchronous: you specify the name of a callback function as an input parameter, and this function is called when the IP address list is available. INETMGR_GetMyIPAddr() returns the device's own IP address. INETMGR_NetStatus() returns the current status (opening, open, closing, or closed) of the device's PPP connection as well as some statistics on the current performance of the network subsystem. INETMGR_SetLinger() sets the linger time of the PPP connection. This is the amount of time that BREW is to wait to terminate the PPP connection after the device's last connected socket is closed (the default value is 30 seconds).

The function INETMGR_GetLastError() returns information about the last error detected by a function in the INetMgr Interface. In cases where such a function returns something other than success, INETMGR_GetLastError() provides more detailed information about why the function failed to perform its task.

To use the functions in the INetMgr Interface

1 Call ISHELL_CreateInstance() to create an instance of the INetMgr Interface.

2 Use functions in the INetMgr Interface as needed:

- Call INETMGR_GetHostByName() or INETMGR_GetMyIPAddr() to obtain remote or local IP address information.
- Call INETMGR_NetStatus() to obtain PPP connection status.
- Call INETMGR_SetLinger() to set the PPP linger time.

To set up a TCP or UDP socket and transfer data over it:

1 Call INETMGR_OpenSocket() to open the socket.

2 Using the pointer returned by INETMGR_OpenSocket(), call ISOCKET_Connect() to connect the socket to the desired destination address.

3 Use functions in the ISocket to read and write data on the connection.

4 Call ISOCKET_Release() to close the connection when you have completed the necessary exchange of data.

5 Call INETMGR_Release() when you no longer need the INetMgr Interface instance.
CAUTION: This brings down the network subsystem, so you must not release the interface while there are any connected sockets on the device.

List of functions

Functions in this interface include:

INETMGR_AddRef()

INETMGR_GetHostByName()

INETMGR_GetLastError()

INETMGR_GetMyIPAddr()

INETMGR_NetStatus()

INETMGR_OnEvent()

INETMGR_OpenSocket()

INETMGR_Release()

INETMGR_SetLinger()

Return to the Contents

INETMGR_AddRef()

Description:

This function increments the reference count of the INetMgr Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See INETMGR_Release().

Prototype:

```
uint32 INETMGR_AddRef(INetMgr * pINetMgr)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

INETMGR_Release()
Return to the List of functions

INETMGR_GetHostByName()

Description:

This function initiates retrieval of IP addresses associated with the specified host name. Results are placed in the result structure and a callback is called to notify the caller of completion. An AEECallback record is used to specify and cancel callbacks, a la ISHELL_Resume (see comments on AEECallback, later in this section).

The memory 'pres' and 'pcb' point to must remain valid for the entire duration of the operation (i.e until the completion callback is called, or until the operation is canceled.) The result structure need not be initialized before the operation; GetHostByName will assign its values.

The AEECallback must be properly initialized.

The text string at 'psz' can be discarded after the call to INETMGR_GetHostByName().

The call to INETMGR_GetHostByName() always "succeeds" in that it guarantees the callback will be called. Any errors related to handling the request are delivered to the callback, so all error checking can be done there.

Prototype:

```
void INETMGR_GetHostByName(INetMgr *pINetMgr,
         AEEDNSResult *pres,
         const char * psz,
         AEECallback *pcb);
```

Parameters:

pINetMgr	Pointer to INetMgr Interface object
pres	Pointer to result structure
psz	Domain name to be resolved, terminated by a colon (":"), or slash ("/") character. nill ('\0'),
pData	User data pointer sent as first argument to callback

Return Value:

None

On completion:

- pres->addrs[] contains the result IP addresses.
- pres->nResult contains the number of addresses obtained (1...AEEDNSMAXADDRS), or an error code if the host has no addresses or if an error was encountered while requesting the information. The following error code definitions are specific to INETMGR_GetHostByName():

AEE_NET_BADDOMAIN	Host name is mal-formed; not a valid host name.
AEE_NET_UNKDOMAIN	Unknown host, or one without IP addresses.
AEE_NET_ETIMEDOUT	No response was seen within the maximum time limit.
EUNSUPPORTED	No DNS servers are configured.
ENOMEMORY	Can not perform query due to allocation failure.

Other error codes related to network connection failure, and so forth, are also possible. See ISOCKET_SendTo() for a complete list.

Comments:

AEECallback conventions and hints:

- To specify the callback function to be called and its argument, one can use the CALLBACK_Init macro:
  `CALLBACK_Init(&callback, CallbackFunction, (void *)ptrArg)`
  This must be done before the GetHostByName() call. For clarity, CALLBACK_Init() must immediately preceed the scheduling call.
- Before the AEECallback record is first used -- typically, in the constructor of the object that contains the AEECallback -- its pfnCancel member must be intialized to NULL. (For heap-allocated objects in BREW, which are initialized to zero, this step is unnecessary.)
- To cancel a request in progress and prevent the callback from being called, use CALLBACK_Cancel(&callback).
- Any object containing an AEECallback must call CALLBACK_Cancel () from its destructor to guarantee that no dangling callback pointers are left.
- An AEECallback can be re-used multiple times, either before of after completion of the previous asynchronous operation. An AEECallback can only keep track of one pending callback at a time. Using the AEECallback to initiate a new asynchronous operation before

the prior one completes will stop the still-in-progress operation and cancel the pending callback.

Performance issues:

Results will be cached for the duration specified by the DNS TTL value. An internal limit on the number of cache entries will be applied, and when exceeded the oldest entries will be expired to make room for new ones. Multiple requests for the same domain will be coalesced into a single transaction with the DNS server(s). If there is no response from the network, GetHostByName() will stop trying after about 42 seconds. Applications are free to impose their own time limits and cancel the operation. Specifically, interactive applications can allow the user to terminte the operation.

Side Effects:

None

See Also:

None
Return to the List of functions

INETMGR_GetLastError()

Description:

This function returns the last error that occurred at the INetMgr Interface. The value returned depends on the most recently called function. The documentation for each function describes what INETMGR_GetLastError() can return when called right after that function.

Prototype:

```
int INETMGR_GetLastError(INetMgr * pINetMgr)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object to be used to retrieve the last error

Return Value:

The most recently occurred error

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

INETMGR_GetMyIPAddr()

Description:

This function returns the IP address of the local host (i.e device).

Prototype:

```
INAddr INETMGR_GetMyIPAddr(INetMgr * pINetMgr)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object to be used to retrieve the IP address

Return Value:

IP address of the device

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

INETMGR_NetStatus()

Description:

This function returns the current network status. It returns a NetState enum of the type NetState. If the AEENetStats pointer is valid, the buffer is filled with the current network connection information such as data rate, active time, bytes sent, and other items. This allows the caller to view the performance of the current session as well as all sessions since the last time the device was reset.

Prototype:

```
NetState INETMGR_NetStatus(INetMgr * pINetMgr, AEENetStats * pNetStats)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object to be used to get the current network status
pNetStats	Pointer to block to be filled with statistical data

Return Value:

A NetState enum indicating the current network status

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

INETMGR_OnEvent()

Description:

This function allows the caller to register to receive notification of in progress INetMgr operations. These include both network and socket functions.

Prototype:

```
void INETMGR_OnEvent(INetMgr * po, PFNNETMGREVENT pfn, void * pUser, boolean bRegister)
```

Parameters:

po	Pointer to the INetMgr Interface
pfn	User-specified callback to call when event occurs
pUser	User-specified context data passed as first argument to callback
bRegister	TRUE, if registering, FALSE if deregistering

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to List of functions

INETMGR_OpenSocket()

Description:

This function creates a socket and returns a pointer to the ISocket Interface.

Prototype:

```
ISocket * INETMGR_OpenSocket(INetMgr * pINetMgr, NetSocket type)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object to be used to create the socket
type	Specifies the socket type: AEE_SOCK_STREAM for TCP AEE_SOCK_DGRAM for UDP

Return Value:

Pointer	To the ISocket Interface, if successful,
NULL	In this case, specific error code can be retrieved by calling INETMGR_GetLastError()

Error Codes:

AEE_NET_EMFILE	No more sockets available for opening
AEE_NET_ESOCKNOSUPPORT	The specified socket type is not supported in this address family
AEE_NET_GENERAL_FAILURE	General Failure

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

INETMGR_Release()

Description:

This function decrements the reference count for the INetMgr Interface object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 INETMGR_Release(INetMgr * pINetMgr)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface object whose reference count needs to be decremented

Return Value:

The updated reference count

Comments:

If linger time has not been set and there are no more open sockets for this INetMgr Interface object , this function closes the associated PPP connection.

Side Effects:

None

See Also:

INETMGR_SetLinger()
Return to the List of functions

INETMGR_SetLinger()

Description:

This function sets the linger time for the network connection specified by **pINetMgr**. The function sets the linger time to the value specified by **wSecs** and returns the previous set value. When the connection is created, the default linger time is set to 30 seconds. If the linger time is set to 0 (zero) and the linger time is running, the network connection is closed.

Prototype:

```
uint16 INETMGR_SetLinger(INetMgr * pINetMgr, uint16 wSecs)
```

Parameters:

pINetMgr	Pointer to the INetMgr Interface
wSecs	Linger time in seconds

Return Value:

Previous linger time value

Comments:

None

Side Effects:

None

See Also:

INETMGR_Release()
Return to the List of functions

INotifier Interface

Description:

Notifications is a mechanism by which classes can be notified when certain events occur in other classes. For example, an applet can register to be notified when there is an incoming UDP Packet. Whenever there is an incoming UDP Packet, the applet that has registered to be notified receives an EVT_NOTIFY event.

Notifiers are classes that generate and dispatch notifications when certain events occur. For example, the INetMgr Interface acts as a notifier and it sends out notifications when a network connection is opened or closed. Whenever a notifier needs to send out notifications, it uses the ISHELL_Notify() function.

The applets can register for notifications by using the ISHELL_RegisterNotify() function. For example, an applet can use this mechanism to register for notification from the INetMgr Interface for events specific to that class.

The INotifier Interface specifies the functions that must be implemented by any class that needs to be a notifier. Any class that wants to send out notifications so that other applets can receive it must implement the functions in the INotifier Interface.

To have a class be a notifier:

1. The class must implement the INotifier Interface.

2. Define the set of notifications (or masks) that the class can issue. For example, some of the notifications that the INetMgr Interface class can issue include:

   NMASK_OPENED

   NMASK_CLOSED

   NMASK_UDP_LISTEN

   These masks must be made available to other applets so the applets that are interested in these notifications can register for them.

3 Whenever the notifier class wants to issue a notification, it must invoke the ISHELL_Notify() function. The AEE Shell then takes care of informing all the applets that have registered for this notification.

List of functions

Functions in this interface include:

INOTIFIER_AddRef()

INOTIFIER_Release()

INOTIFIER_SetMask()

Return to the Contents

INOTIFIER_AddRef()

Description:

This function increments the reference count of the INotifier Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 INOTIFIER_AddRef(INotifier * pINotifier)
```

Parameters:

pINotifier Pointer to the INotifier Interface object

Return Value:

Incremented reference count for the object. .

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

INOTIFIER_Release()
Return to the List of functions

INOTIFIER_Release()

Description:

This function decrements the reference count of the INotifier Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 INOTIFIER_Release(INotifier * pINotifier)
```

Parameters:

pINotifier	Pointer to the INotifier Interface object

Return Value:

Referenceecount	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

INOTIFIER_AddRef()
Return to the List of functions

INOTIFIER_SetMask()

Description:

This function is invoked by the AEE Shell to inform the notifier class of all the notifications (issued by that class) that other applets are interested in. This function must be implemented by all notifiers (i.e. by all classes that implement the INotifier Interface). This function is strictly meant to be invoked internally by the AEE Shell. It must never be directly invoked by other applets or classes.

Prototype:

```
void INOTIFIER_SetMask(INotifier * pINotifier, const uint32 * dwMasks)
```

Parameters:

pINotifier	Pointer to the INotifier Interface object
dwMasks	Specifies the array of masks representing the notifications that other applets are interested in

Return Value:

None

Comments:

A class may be capable of emitting multiple notifications (for example, NMASK_OPENED, NMASK_CLOSED, and NMASK_UDP_LISTEN). However, the applets that have registered for notifications may only be interested in the NMASK_OPENED and NMASK_CLOSED notifications. Whenever an application registers for a notification using the ISHELL_RegisterNotify(), it must specify the mask for the notification that it is interested in. The AEE Shell then invokes this function INOTIFIER_SetMask() on the notifier to inform the notifier of all the notifications that the applets are interested in.

Side Effects:

None

See Also:

ISHELL_RegisterNotify()
ISHELL_Notify()
Return to the List of functions

The functions in the IShell Interface provide a variety of services to BREW applications. Later in this section is a high-level overview of the IShell Interface functions, with each subsection describing a group of related functions.

Alarms

The AEE Shell's alarm functions enable an application to be notified when the current time reaches a specified value. Unlike timers, which can only be active while your application is running, you can receive notification that an alarm has expired even when your application is not running. Alarms are typically used when the time of notification is in the fairly distant future. For example, a calendar application can use an alarm to alert the user when a time of a calendar appointment is about to be reached.

To set an alarm, you call the ISHELL_SetAlarm() function, specifying the number of minutes from the current time at which the alarm notification is to occur, a 16-bit alarm code, and the BREW ClassID of the application (yours or another) that receives notification when the alarm time is reached. At the notification time, the IAPPLET_HandleEvent() function of the notified application is called with an **EVT_ALARM** event and the 16-bit alarm code as parameters (the latter parameter allows an application to distinguish among more than one simultaneously active alarm). If the notified application is not currently running, the AEE Shell creates an instance of it to process the notification event, after which it isterminated (the application may choose to activate itself if necessary). The AEE Shell stores alarms in a BREW database and continuously checks for alarm expirations while the BREW-enabled device is turned on. If an alarm's expiration time passes while the device is turned off, the AEE Shell generates the notification the next time the device is turned on.

The ISHELL_CancelAlarm() function is used to cancel a currently active alarm. ISHELL_AlarmsActive() checks whether any of BREW's built-in annunciators (alarm clock, countdown timer, or stopwatch) are currently active.

Application management

The AEE Shell's application management functions have a number of purposes, including

- Creating, starting, and stopping BREW classes and applications
- Obtaining information about the modules and classes present on the device
- Allowing applications to send events to each other
- Allowing BREW applications to execute without interfering with other activities that the device must perform

The ISHELL_CreateInstance() function is used to create instances of both BREW classes and of user-defined classes supported by the modules present on the device. ISHELL_StartApplet() allows a specified applet to start execution. It creates an instance of the applet if necessary, suspends the currently running applet (if any), and then invokes the specified applet's IAPPLET_HandleEvent() function with the EVT_APP_START event, which allows it to begin execution. ISHELL_CloseApplet() sends the currently executing applet the EVT_APP_STOP event and calls its Release function. ISHELL_CloseApplet() is used primarily by the AEE Shell itself, since it is not possible for one applet to stop another.

ISHELL_ActiveApplet() is used to obtain the ClassID of the applet that is currently running. The functions ISHELL_EnumAppletInit() and ISHELL_EnumNextApplet() can be used to enumerate the applets in the modules that are present on the device. As each applet is enumerated, ISHELL_EnumNextApplet() returns a pointer to an AEEAppInfo data structure for the applet, which identifies the applet's MIF file, titles, icons, and type (game, tool, PIM, and other types). ISHELL_QueryClass() determines if a particular class is available on the device (if the class is an applet, you can supply a pointer to an AEEAppInfo data structure, which is populated with the relevant information if the applet is available).

You use the ISHELL_SendEvent() function to send an event to a specified class. If an instance of that class is not currently present, the AEE Shell creates one, and then invokes its IAPPLET_HandleEvent() function with the specified event code and data parameters. Unless it chooses to start itself, the application terminates after it completes processing of the event. The function ISHELL_PostEvent() is similar, except that the destination class's IAPPLET_HandleEvent() function is not called immediately. The event is placed in a queue and is sent at a later time, which allows the calling application to continue its processing without interruption. ISHELL_HandleEvent() is used when the identity of the class that is to receive the event is not known; in this case, the AEE Shell sends the event to the currently running applet or its active dialog (if any).

The AEE Shell includes several functions that enable BREW applications to coexist with other activities on the device. The BREW application model is based on cooperative multitasking, which means that each application must be designed to execute for as short a time as possible when processing an event and then exit to give other device activities a chance to execute. The function ISHELL_Busy() lets a BREW application determine if any activities are in progress on the device that require it to exit immediately (ISHELL_ForceExit() is identical to ISHELL_Busy()). ISHELL_Resume() allows an application to break up time-consuming tasks into smaller, interruptible chunks. Each chunk is represented by a callback function and an associated data pointer; calling ISHELL_Resume() schedules the callback function to be invoked at a later time with the data pointer as its only parameter. ISHELL_CanStartApplet() is used to check if it is possible to start a BREW application, which may not be true if higher-priority activities are in progress on the device.

Dialogs, message boxes, and prompts

A dialog consists of a screen containing one or more BREW controls that allow the device user to enter data or select an item from a menu. Although you can create such a screen with the BREW control interfaces (IDateCtl Interface, IMenuCtl Interface, ITextCtl Interface and ITimeCtl Interface), BREW's IDialog Interface greatly simplifies this task:

- You can create a dialog using the BREW Resource Editor, including specification of each of the controls in the dialog.
- The BREW Application Execution Environment (AEE) maintains a stack of the dialogs associated with the currently executing application. When you create a dialog, it is placed at the top of the stack; when a dialog is ended, it is removed from the stack, and the dialog below it is restored. This makes it easy to implement applications that step the user through a sequence of screens (for example, a menu hierarchy).
- When a dialog is active, it receives all events and distributes them to the currently active controls. The dialog also handles control tabbing, which allows the device user to move between controls in a multicontrol dialog. This frees you from having to implement code to distribute and handle these events yourself.

You use the ISHELL_CreateDialog() function to create a dialog. This function accepts either the identifier of a dialog you have created in the *BREW Resource Editor* or a pointer to a data structure that you populate in your code to specify the controls in the dialog. If successful, the function displays the dialog's controls on the screen and pushes it onto the dialog stack. To end the dialog, you call the ISHELL_EndDialog() function, which terminates the dialog at the top of the stack and displays the dialog immediately below it on the stack (if any). This function also sends the **EVT_DIALOG_END** event to your application, which allows you to perform any processing associated with the dialog's termination, such as retrieving values entered by the user in the dialog's controls. ISHELL_EndDialog() also frees all the resources that are being used by the dialog.

ISHELL_CreateDialog() does not return an interface pointer to the dialog it creates. The function ISHELL_GetActiveDialog() is used to obtain an IDialog Interface pointer for the dialog at the top of the stack. You can use this pointer to invoke the functions that comprise the IDialog Interface: IDIALOG_GetControl(), which returns interface pointers to any of the dialog's controls, and IDIALOG_SetFocus(), which is used to specify which control in a multicontrol dialog receives input from the user.

The AEE Shell also provides functions that can be used to create some simple, commonly used dialogs with a single function call. ISHELL_Prompt() displays a dialog with a SoftKey control menu that prompts the user to make a selection; when the user does so, the selection is returned to your application via an **EVT_COMMAND** and the dialog is automatically terminated. There are also two functions that create a dialog that displays a read-only text message and title: ISHELL_MessageBox() reads the title and message text from a BREW resource file, while ISHELL_MessageBoxText() accepts pointers to title and message text strings that you specify in your code. The dialogs created by these functions end when the user presses a key.

Device and application configuration information

These functions allow configuration information about the device itself and particular applications to be obtained. ISHELL_GetDeviceInfo() returns a pointer to an AEEDeviceInfo structure for the device, which includes information about its screen size and color support, amount of available memory, character encoding, and other items.

ISHELL_GetPrefs() and ISHELL_SetPrefs() provide a general-purpose mechanism for applications to register configuration information. You can use ISHELL_SetPrefs() to associate a pointer to configuration data and a version number for this data with the ClassID of your application. Other applications can obtain this data by calling ISHELL_GetPrefs() with the desired ClassID and version number (you also need to make available a structure declaration that defines the content of each version of your application's configuration data).

Miscellaneous

ISHELL_Beep() allows an application to provide several types of audible or vibrating signals to the user. Depending on what a given device supports, there are audible signals that correspond to device-off, alert, reminder, message arrival and error events, and vibrating signals for alerts and reminders. There is also a Boolean loudness parameter that can be specified.

Notifications

IShell's notification mechanism allows a BREW class to notify other classes that certain events have occurred. A class wishing to receive a notification must register its interest with the AEE Shell, specifying the ClassID of the notifier class and the events for which notification is desired. When an event requiring notification occurs, the notifier class calls ISHELL_Notify(), which sends notification to each class that has registered to be notified of the occurrence of that event.

The AEE Shell provides two ways for a class to register for notification of an event:

- You can register by specifying information about the notification in your application's MIF file using the MIF Editor. This method of registering is used by applications that must be notified of events even when they are not running. One example is a call-logging application that receives notification of each incoming and outgoing call; such an application would need to process notifications even while the user was not running the app to display the call log.
- If notification is required only at certain times while your application is running, you can call ISHELL_RegisterNotify() to initiate event notification. For example, a game application might display a message when an incoming call arrives that would allow the user to accept the call or continue playing the game. This application requires notification of incoming calls only while the user is actually playing the game, so it would call ISHELL_RegisterNotify() when the user starts to play the game.

The events for which a notifier class provides notifications are represented by a 32-bit variable. The low-order 16 bits of this variable contain the notification mask, with each bti corresponding to one event. The high-order 16 bits contain the notifier match value, which can be used for data associated with an event (for example, events representing activity on a UDP socket use the match value for the port on which the activity has occurred). When it registers for notification, a class provides a value for the mask variable with the bit for each event of interest set to 1. If a class no longer requires notification of an event, it can call ISHELL_RegisterNotify() with that event's bit set to 0 (zero) in the mask variable (for example, an application might call ISHELL_RegisterNotify() with the mask variable set to 0 (zero) upon termination).

> **NOTE:** The AEE Shell creates an instance of the notifier class when another class registers for notification of its events. This instance is released when all classes have un-registered for event notification.

When a notifier class calls ISHELL_Notify(), the AEE Shell sends an event of type EVT_NOTIFY to each application that has registered for the event. This results in the invocation of the application's IAPPLET_HandleEvent() function. If the application is not currently running, the AEE Shell creates an instance of it to process the event, after which the application terminates (the application can choose to

send itself a start event if it wishes to continue running). In the call to IAPPLET_HandleEvent(), the application also is passed a pointer to a structure of type AEENotify, which identifies the event that occurred, the class that generated the notification, and some additional data specific to the notification.

At present, the INetMgr class provides notifications. It notifies other interested classes of changes in the network connection state and the arrival of data on particular UDP ports. You can also implement your own notifier classes. To provide notifier functionality in your class, perform the following steps:

- Your class must implement the INotifier Interface (refer to the sample notifications application for details on how to declare this interface when defining your class). The INotifier Interface requires you to implement a INOTIFIER_SetMask() function for your class. The AEE Shell calls this function whenever another class registers for event notification from your class. INOTIFIER_SetMask() has a single argument, which is a 32-bit variable that is the logical OR of the mask variables that each class used when it registered for notifications from your class. This variable has a value of 1 for a bit if at least one class has registered for notification of the corresponding event. You can use this variable to perform any initialization needed when a class first registers to be notified of an event, and any finalization when there are no longer any classes requiring notification of a particular event.

- Provide an include file that contains constants defining the bits assigned to each notification event, and **typedefs** for the event-specific data that is provided as part of an event notification. This include file is used by applications that register for notifications from your class.

- Call ISHELL_Notify() whenever a notification event occurs and at least one class has registered to be notified of that event's occurrence.

Resource files and file handlers

The AEE Shell provides a number of functions that your application can use to read in various types of data from files. These files can be BREW resource (.bar) files created with the *BREW Resource Editor,* or they can be files whose content is associated with a MIME type and/or identified by the file's extension. You can also extend the set of file types that BREW recognizes by defining your own handler classes and using them to manipulate files of particular MIME types.

You can use the following functions to access BREW resource files (each function's parameters include the name of the resource file and the integer resource ID):

- ISHELL_LoadResData() is used to load resources other than strings and bitmap images. At present, these include several resource types associated with dialogs. The memory used to store the resource information is freed by calling ISHELL_FreeResData().
- ISHELL_LoadResImage() loads a bitmap image from a specified resource file and returns a pointer to an instance of the IImage Interface that contains the bitmap. IIMAGE_Release() frees the data used to store the bitmap.
- ISHELL_LoadResObject() is a utility function used in the implementation of the sound and image loading functions. The functions ISHELL_LoadImage(), ISHELL_LoadResImage(), ISHELL_LoadSound(), and ISHELL_LoadResSound() are all macros that invoke this function with different parameters.
- ISHELL_LoadResSound() loads a sound resource from the specified resource file and returns a pointer to an instance of the ISoundPlayer Interface that contains the sound file. ISOUNDPLAYER_Release() frees the data used to store the sound data.
- ISHELL_LoadResString() reads a string resource into a character buffer, a pointer to which is one of the function's arguments.

The functions ISHELL_LoadImage() and ISHELL_LoadSound() can be used to load image and sound files directly, without first placing their contents into a BREW resource file. The files must contain one of the built-in MIME types supported by BREW, which include Windows bitmap (.BMP) or a device-specific native bitmap format for images and MIDI (.midi) or CMX (.cmx) sound files.

You can use the function ISHELL_RegisterHandler() to associate a MIME file type with the ClassID of the BREW handler class you have implemented to handle files of that type. The function ISHELL_GetHandler() returns the ClassID of the handler class associated with a given MIME type (including the BREW built-in types mentioned above).

Call ISHELL_GetHandler() to get the ClassID of the handler class for the file's MIME type.

Call ISHELL_CreateInstance() to create an instance of the handler class.

Use the IFileMgr Interface and IFile Interface to read the contents of the file into memory and associate the file's contents with the handler class instance you have created (for example, see the Return to the List of functions function). You can also register your MIME type handlers in your application's MIF.

Timers

The AEE Shell's timer facility is used by a currently instantiated application (that is, an application whose reference count is non-zero) to perform an action when a specified amount of time has passed. These time periods are typically short (on the order of seconds or milliseconds); you can use the AEE Shell's alarm functions to obtain notification when longer time periods have passed, even when your application is not currently instantiated.

To start a timer, you call the ISHELL_SetTimer() function, specifying the timer duration in milliseconds, the address of a callback function, and a pointer to an application-specific data structure. When the timer expires, the AEE Shell calls the callback function with the application-specific data pointer as its only parameter. BREW timers are one-shot, nonrecurring timers. It is not possible to specify a timer that repeats at a fixed time interval. To obtain this behavior, you can call ISHELL_SetTimer() within your callback function and specify the function's own address as ISHELL_SetTimer()'s callback function parameter.

ISHELL_GetTimerExpiration() can be used to determine the number of milliseconds remaining before a particular timer expires; a timer is identified by the callback function and data structure addresses supplied when it was created. ISHELL_CancelTimer() cancels a running timer. If a null value is supplied for the callback function parameter, all timers associated with the specified data structure address are canceled. When an application instance's reference count drops to 0 (zero), all timers associated with that application are canceled.

List of functions

Functions in this interface include:

ISHELL_ActiveApplet()

ISHELL_AddRef()

ISHELL_AlarmsActive()

ISHELL_Beep()

ISHELL_BrowseFile()

ISHELL_BrowseURL()

ISHELL_Busy()

ISHELL_CancelAlarm()

ISHELL_CancelTimer()

ISHELL_CanStartApplet()

ISHELL_CheckPrivLevel()

ISHELL_CloseApplet()

ISHELL_CreateDialog()

ISHELL_CreateInstance()

ISHELL_EndDialog()

ISHELL_EndDialog()

ISHELL_EnumAppletInit()

ISHELL_EnumNextApplet()

ISHELL_ForceExit()

ISHELL_FreeResData()

ISHELL_GetActiveDialog()

ISHELL_GetDeviceInfo()

ISHELL_GetHandler()

ISHELL_GetItemStyle()

ISHELL_GetPosition()

ISHELL_GetPrefs()

ISHELL_GetTimerExpiration()

ISHELL_HandleEvent()

ISHELL_IsValidResource()

Return to the Contents

ISHELL_ActiveApplet()

Description:

This function returns the AEECLSID associated with the currently running applet.

Prototype:

```
AEECLSID ISHELL_ActiveApplet(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

applet ID	ID of the active applet, if applet is running
0 (zero)	If no applet is running

Comments:

None

Side Effects:

None

See Also:

ISHELL_StartApplet()
ISHELL_CloseApplet()
ISHELL_CanStartApplet()
Return to the List of functions

ISHELL_AddRef()

Description:

This function increments the reference count of the IShell Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ISHELL_Release().

Prototype:

```
uint32 ISHELL_AddRef(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

ISHELL_Release()
Return to the List of functions

ISHELL_AlarmsActive()

Description:

This function returns TRUE if any of ANNUN_ALARMCLOCK, ANNUN_COUNTDOWN and ANNUN_STOPWATCH are set. This function does not check for any other type of alarms.

Prototype:

```
boolean ISHELL_AlarmsActive(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

TRUE	If there are active annunciators
FALSE	If there are no active annunciators

Comments:

On Windows, this function always returns FALSE because annunciators are not supported in the BREW Emulator.

Side Effects:

None

See Also:

IDISPLAY_SetAnnunciators()
Return to the List of functions

ISHELL_Beep()

Description:

This function provides a very simple interface to play system beeps and/or vibrate the device.

Prototype:

```
boolean ISHELL_Beep(IShell * pIShell, BeepType nBeep, boolean bLoud)
```

Parameters:

pIShell	Pointer to the IShell Interface object
nBeep	Type of beep to play
bLoud	TRUE if the beep can be played at a higher volume

Return Value:

TRUE	If the beep was played
FALSE	If the beep was not played

Comments:

None

Side Effects:

None

See Also:

BeepType
Return to the List of functions

ISHELL_BrowseFile()

Description:

This function attempts to find the associated registered applet for the extension of the file whose path is given by the **pszFile** parameter. If a handler applet is found, this function creates an instance of it and sends the instance the EVT_APP_START event. If the applet starts successfully, ISHELL_BrowseFile() sends it the EVT_APP_BROWSE_FILE event along with the file path (the handler applet can then browse the file).

Prototype:

```
void ISHELL_BrowseFile (IShell * pIShell, const char * pszFile);
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszFile	Pointer to a string containing the name of the file to be browse. The file's extension (e.g., gif, htm) is used to locate the handler applet that will browse the file.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_RegisterHandler()
ISHELL_GetHandler()
ISHELL_BrowseURL()
Return to the List of functions

ISHELL_BrowseURL()

Description:

This function attempts to find the associated registered applet for the URL scheme of the URL contained in the string pointed to by the **pszURL** parameter. If a handler applet is found, this function creates an instance of it and sends the instance the EVT_APP_START event. If the applet starts successfully, ISHELL_BrowseURL() sends it the EVT_APP_BROWSE_URL event along with the URL (the handler applet can then browse the URL).

Prototype:

```
void ISHELL_BrowseURL (IShell * pIShell, const char * pszURL);
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszURL	Pointer to a string containing the URL to be browsed. The URL's scheme (e.g., http, mailto) is used to locate the handler applet that will browse the URL.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_RegisterHandler()
ISHELL_GetHandler()
ISHELL_BrowseFile()
Return to the List of functions

ISHELL_Busy()

Description:

This function returns TRUE if the applet must exit. This function is provided only for very special use by browsers or other applications that may do substantial processing before returning from the ISHELL_HandleEvent() function.

Prototype:

```
boolean ISHELL_Busy(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

TRUE	If the applet must stop processing and exit
FALSE	If the applet can continue to do processing

Comments:

This function uses ISHELL_ForceExit()

Side Effects:

None

See Also:

ISHELL_ForceExit()
Return to the List of functions

ISHELL_CancelAlarm()

Description:

This function cancels an alarm set via ISHELL_SetAlarm(). The proper class and 16-bit user code must be specified.

Prototype:

```
int ISHELL_CancelAlarm(IShell * pIShell, AEECLSID cls, uint16 nUserCode)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	Applet class of alarm to cancel
nUserCode	16-bit use code of alarm to cancel

Return Value:

SUCCESS	If alarm is found and canceled
EFAILED	If alarm can not be found or canceled

Comments:

None

Side Effects:

None

See Also:

ISHELL_SetTimer()
Return to the List of functions

ISHELL_CancelTimer()

Description:

This function cancels a timer that has been set by ISHELL_SetTimer(). If **pfn** is non-NULL, the timer associated with **pfn** and **pUser** is canceled. If **pfn** is NULL, all timers associated with the **pUser** value are canceled.

Prototype:

```
int ISHELL_CancelTimer(IShell * pIShell, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pfn	User callback
pUser	User data

Return Value:

SUCCESS	Always

Comments:

Attempting to cancel timers that have not been set is harmless.

Side Effects:

None

See Also:

ISHELL_SetTimer()
ISHELL_GetTimerExpiration()
Return to the List of functions

ISHELL_CanStartApplet()

Description:

This function queries the AEE Shell to determine whether it is safe to start an applet. Under normal conditions, this call returns TRUE. However, in some cases, new applet startup may be prevented when critical dialogs are displayed or other applets are running.

Prototype:

```
boolean ISHELL_CanStartApplet(IShell * pIShell, AEECLSID cls)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	ClassID of the applet that needs to be checked

Return Value:

TRUE	If it is safe to start the applet
FALSE	If it is not safe to start the applet

Comments:

None

Side Effects:

None

See Also:

ISHELL_StartApplet()
ISHELL_CloseApplet()
ISHELL_ActiveApplet()
Return to the List of functions

ISHELL_CheckPrivLevel()

Description:

This function checks the privilege level of the currently executing application against the specified value. If the privilege level matches, the function returns TRUE. If the privilege level does not match, the function returns FALSE and conditionally terminates the application with a system message.

Prototype:

```
boolean ISHELL_CheckPrivLevel(IShell * pIShell, uint16 wPrivWant, boolean
bQueryOnly)
```

Parameters

pIShell	Pointer to the IShell Interface object
wPrivWant	Desired privilege bits to check
bQueryOnly	TRUE if this is a query only. If FALSE, the app will terminate if it does not have the privileges specified in wPrivWant.

Return Value:

TRUE	App supports this privilege level mask
FALSE	App does not support this privilege level

Comments:

If the **bQueryOnly** value is FALSE and the function returns FALSE, no further processing is done on the application as it will be terminated.

Side Effects:

None

See Also:

None
Return to the List of functions

ISHELL_CloseApplet()

Description:

This function instructs the AEE Shell to close the active applet. The applet is sent the EVT_APP_STOP message and the IAPPLET_Release() function is called. The **bReturnToIdle** parameter indicates whether the AEE must close all other active applications and return the idle screen.

Prototype:

```
int ISHELL_CloseApplet(IShell * pIShell,boolean bReturnToIdle)
```

Parameters:

pIShell	Pointer to the IShell Interface object
bReturnToIdle	This parameter indicates whether the AEE Shell must close all other active applications and Return to the idle screen

Return Value:

SUCCESS	Operation was successful
EFAILED	Operation failed

Comments:

The **bReturnToIdle** parameter is not supported on the BREW Emulator.

Side Effects:

None

See Also:

ISHELL_StartApplet()
ISHELL_CanStartApplet()
ISHELL_ActiveApplet()
Return to the List of functions

ISHELL_CreateDialog()

Description:

This function instructs the AEE Shell to start a dialog associate with the input DialogInfo data structure or the dialog information in the associated resource file. This call causes the AEE Shell to create an IDialog Interface. This interface processes the dialog input parameters to create all associated controls, and other items. The dialog is placed at the top of the active dialog stack.

During creation, execution, and termination, the IDialog Interface sends a number of dialog-related AEE Events (such as: EVT_DIALOG_INIT, EVT_DIALOG_START, EVT_DIALOG_END) to the active applet. This allows the applet to control the initial contents of controls, manage control changes and retrieve the contents of controls at termination. The applet can obtain pointers to the underlying, dialog controls by calling the IDIALOG_GetControl() function.

Prototype:

```
int ISHELL_CreateDialog(IShell * pIShell, const char * pszResFile, uint16
wID, DialogInfo * pInfo)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Pointer to the resource file containing the dialog information
wID	ID of the dialog inside the resource file
pInfo	Alternate dialog information structure. If this structure is provided, the **pszResFile**/**wID** parameters are ignored.

Return Value:

SUCCESS	If successful
ENOMEMORY	Insufficient memory
EBADPARM	Invalid parameter
EFAILED	If unsuccessful

Comments:

Display of the dialog title is not supported in this release.

Side Effects:

None

See Also:

DialogInfo
ISHELL_GetActiveDialog(),
ISHELL_EndDialog(),
IDIALOG_SetFocus()
IDIALOG_GetControl()
EVT_DIALOG_INIT,
EVT_DIALOG_START,
EVT_DIALOG_END
Return to the List of functions

ISHELL_CreateInstance()

Description:

This function is called to create an object associated with the 32-bit ClassID specified. The object return must match the interface supported by the ClassID provided. Upon success, the **ppobj** is filled with an object of the specified class. The object is returned with a positive reference count.

Prototype:

```
int ISHELL_CreateInstance(IShell * pIShell, AEECLSID cls, void * * ppobj)
```

Parameters:

pIShell	[in]	Pointer to the IShell Interface object
cls	[in]	32-bit ClassID of the requested interface
ppobj	[out]	pointer to the memory to fill with the pointer to the object

Return Value:

SUCCESS	Class created
ENOMEMORY	Insufficient memory
ECLASSNOSUPPORT	Class specified is not supported
EBADPARM	If null ppobj was passed in

Comments:

None

Side Effects:

None

See Also:

IMODULE_CreateInstance()
Return to the List of functions

ISHELL_EndDialog()

Description:

This function closes the currently active dialog.

Prototype:

```
ISHELL_EndDialog(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

SUCCESS	Active dialog successfully closes
EFAILED	No dialog is active

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateDialog()
ISHELL_GetActiveDialog()
IDIALOG_SetFocus()
IDIALOG_GetControl()
Return to the List of functions

ISHELL_EnumAppletInit()

Description:

This function resets the AEE Shell's internal applet enumeration index. This call is used in conjunction with ISHELL_EnumNextApplet() as follows:

```
ISHELL_EnumAppletInit(pShell) ; while(ISHELL_EnumNextApplet(pShell, &ai) )
{ process... }
```

Prototype:

```
void ISHELL_EnumAppletInit(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_EnumNextApplet()
Return to the List of functions

ISHELL_EnumNextApplet()

Description:

This function returns information regarding the next applet. It is used in conjunction with ISHELL_EnumAppletInit() as follows:

```
ISHELL_EnumAppletInit(pShell) ; while(ISHELL_EnumNextApplet(pShell, &ai) )
{ process... }
```

Prototype:

```
AEECLSID ISHELL_EnumNextApplet(IShell * pIShell, AEEAppInfo * pai)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pai	Pointer to AEEAppInfo structure to fill

Return Value:

ClassID	Of the next applet, if successful
0 (zero)	When there are no more applets to enumerate or if **pai** is NULL

Comments:

None

Side Effects:

None

See Also:

AEEAppInfo
ISHELL_EnumAppletInit()
Return to the List of functions

ISHELL_ForceExit()

Description:

This function returns TRUE if the applet must exit. This function is provided only for very special use by browsers or other applications that may do substantial processing before returning from the ISHELL_HandleEvent() function.

Prototype:

```
boolean ISHELL_ForceExit(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

TRUE	If the applet must stop processing and exit
FALSE	If the applet can continue to do processing

Comments:

None

Side Effects:

None

See Also:

ISHELL_Busy()
Return to the List of functions

ISHELL_FreeResData()

Description:

This function frees the data previously returned by ISHELL_LoadResData().

Prototype:

```
void ISHELL_FreeResData(IShell * pIShell, void * pData)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pData	Resource Data that is to be freed that must have been returned by a prior call to ISHELL_LoadResData()

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_LoadResData()
Return to the List of functions

ISHELL_GetActiveDialog()

Description:

This function returns the current active dialog. This call is provided so that applets may query the dialog for underlying controls, and other items.

Prototype:

```
IDialog * ISHELL_GetActiveDialog(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

Pointer	To the currently active dialog, if successful
NULL	If there is no active dialog or if unsuccessful

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateDialog()
ISHELL_EndDialog()
IDIALOG_SetFocus()
IDIALOG_GetControl()
Return to the List of functions

ISHELL_GetDeviceInfo()

Description:

This function queries the AEE Shell for information regarding the capabilities of the device. This includes such information as the amount of supported RAM, display information, and other items. In order to obtain values for the fields "dwNetLinger" and "dwSleepDefer" of the AEEDeviceInfo structure, you MUST fill-in the wStructSize element of the structure before passing this to the GetDeviceInfo call. The element wStructSize must be set to be equal to the sizeof(AEEDeviceInfo) structure before making the call to this function.

Example:

```
AEEDeviceInfo di;
di.wStructSize = sizeof(AEEDeviceInfo);
ISHELL_GetDeviceInfo(&di);
```

Prototype:

```
void ISHELL_GetDeviceInfo(IShell * pIShell, AEEDeviceInfo * pi)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pi	Pointer to AEEDeviceInfo structure to fill

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEEDeviceInfo

Return to the List of functions

ISHELL_GetHandler()

Description:

This function provides query access to the AEE Shell's database of registered content viewers and protocol scheme handlers. This database is provided to allow browsers and content viewers to expose handlers for content or data protocols they may support.

The AEEHandlerType data structure indicates whether the input string is a viewer (for example image type) or a sound type.

Prototype:

```
AEECLSID ISHELL_GetHandler(IShell * pIShell,AEEHandlerType t, const char *
pszIn)
```

Parameters:

pIShell	Pointer to the IShell Interface object
t	Handler type (HTYPE_VIEWER, HTYPE_BROWSE, or HTYPE_SOUND)
pszIn	Input string

Return Value:

AEECLSID	Of the associated handler class
0 (zero)	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_RegisterHandler()
Return to the List of functions

ISHELL_GetItemStyle()

Description:

This function queries the AEE Shell for information regarding the default style for menu, icon, list items. The information is placed into the two specified pointers. The first (pNormal) contains information regarding drawing the item in a normal (non-selected) case. The second (pSel) contains information regarding drawing the item in the selected case.

Prototype:

```
boolean ISHELL_GetItemStyle(IShell * pIShell, AEEItemType t, AEEItemStyle *
pNormal, AEEItemStyle * pSel)
```

Parameters:

pIShell	[in]	Pointer to the IShell Interface object
t	[in]	Item type
pNormal	[out]	Pointer to AEEItemStyle to fill for items that are not selected
pSel	[out]	Pointer to AEEItemStyle to fill for selected items

Return Value:

TRUE	If successful
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

AEEItemType,
AEEItemStyle'
Return to the List of functions

ISHELL_GetPosition()

Description:

This method provides access to the gpsOne location feature on the handset. The precision specified indicates how exact the location will be returned. The precision is also directly related to the time it will take to satisfy the request.

> **Warning:** Requests for position location may be protected by the privacy policies determined by the OEM or carrier.

Upon completion, the callback will be made to the user with the position or an appropriate error.

Prototype:

```
int ISHELL_GetPosition(IShell * pIShell, AEEPosAccuracy prc, PFNPOSITIONCB pfn, void * pUser)
```

Parameters:

pIShell	Pointer to the IShell object
rc	Precision
pfn	Pointer to callback
pUser	Pointer to callback data

Return Value:

SUCCESS	Call is in progress
EUNSUPPORTED	gps is not supported

Comments:

Currently, this function is not implemented. It returns EUNSUPPORTED.

Side Effects:

None

See Also:

None
Return to the List of functions

ISHELL_GetPrefs()

Description:

This function provides a means of retrieving a structure containing applet or class level preferences.

Prototype:

```
int ISHELL_GetPrefs(IShell * pIShell, AEECLSID cls, uint16 wVer, void * pCfg,
uint16 nSize)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	AEECLSID of the preference type
wVer	Version of the preference
pCfg	Pointer to memory to fill with preference data
nSize	Size of memory block to fill

Return Value:

SUCCESS	Operation successful
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory
EFAILED	Operation fails

Comments:

None

Side Effects:

None

See Also:

ISHELL_SetPrefs(),
Return to the List of functions

ISHELL_GetTimerExpiration()

Description:

This function returns the remaining time in miliseconds before the timer associated with the user callback/data expires.

Prototype:

```
uint32 ISHELL_GetTimerExpiration(IShell * pIShell, PFNNOTIFY pfn, void *
pUser)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pfn	User callback
pUser	User data

Return Value:

time	If there is a timer associated with specified user callback/data and it hasn't expired, this function returns the remaining expiration time
0 (zero)	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_SetTimer()
ISHELL_CancelTimer()
Return to the List of functions

ISHELL_HandleEvent()

Description:

This function sends one of the specified standard AEE Events (**evt** < EVT_USER) to the currently active applet. It internally invokes ISHELL_SendEvent() with ClassID set to 0 (zero).

Prototype:

```
boolean ISHELL_HandleEvent(IShell * pIShell, AEEEvent evt, uint16 wp, uint32 dwp)
```

Parameters:

pIShell	Pointer to the IShell Interface object
evt	AEE Events
wp	Event-specific 16-bit value
dwp	Event-specific 32-bit value

Return Value:

TRUE	If the event was processed
FALSE	If the event was not processed

Comments:

The event sent must be one of the standard AEE Events (for example, it must be < EVT_USER) . This cannot be an EVT_USER event. Use ISHELL_PostEvent() to send user events.

Side Effects:

None

See Also:

ISHELL_SendEvent()
ISHELL_PostEvent()
Return to the List of functions

ISHELL_IsValidResource()

Description:

This function checks to see if the specified resource file entry is valid for the type specified.

Prototype:

```
boolean ISHELL_IsValidResource(IShell * pIShell, const char * pszResFile,
uint16 wID, ResType t, AEEHandlerType ht);
```

Parameters:

pIShell	Pointer to the IShell interface object
pszResFile	Resource file name
wID	ID of the resource
t	Resource type
ht	Handler type (ignored if t != RESTYPE_IMAGE)

Return Value:

TRUE	Resource exists in file
FALSE	Resource does not exist

Side Effects:

None

See Also:

ResType
AEEHandlerType
Return to the List of functions

ISHELL_LoadImage()

Description:

This function loads an image from a file directly and returns a pointer to an IImage handler that can be used to display the image.

Prototype:

```
IImage * ISHELL_LoadImage(IShell * pIShell, const char * pszResFile)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Image file

Return Value:

Pointer	To an IShell Interface that can be used for viewing the image

Comments:

Example:

```
IImage * pImage;
pImage = ISHELL_LoadResImage(pShell, “Test.bmp“);
If (IImage) {
         IIMAGE_draw(pImage, 10,70) ;
         IIMAGE_Release(pImage);
         }
```

Side Effects:

None

See Also:

ISHELL_LoadResData()
ISHELL_LoadResImage()
ISHELL_LoadResObject()
ISHELL_LoadResSound()

ISHELL_LoadResString()
ISHELL_LoadSound()
ISHELL_FreeResData()
Return to the List of functions

ISHELL_LoadResData()

Description:

This function returns a void * data block associated with the specified resource file, ID, and type. The memory returned must be freed using the ISHELL_FreeResData() call.

Prototype:

```
void * ISHELL_LoadResData(IShell * pIShell, const char * pszResFile, int16
nResID, ResType nType)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Resource file containing the data
nResID	ID of the data in the resource file
nType	The type of resource [RESTYPE_STRING|RESTYPE_IMAGE|RESTYPE_DIALOG|RESTYPE_CONTROL|RESTYPE_ LISTITEM|RESTYPE_BINARY]. If **nType** is RESTYPE_IMAGE, then the first byte indicates the offset where the actual image data begins. The second byte is zero. Starting from the third byte, the string indicates the mime type followed by the actual image data.

Return Value:

void *	A void * pointing to the resource data, if successful
NULL	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_LoadImage()
ISHELL_LoadResImage()
ISHELL_LoadResObject()
ISHELL_LoadResSound()
ISHELL_LoadResString()

ISHELL_LoadSound()
ISHELL_FreeResData()
Return to the List of functions

ISHELL_LoadResImage()

Description:

This function loads a bitmap resource from the given resource file and returns a valid IImage Interface pointer. This pointer can then be used for viewing the image.

Prototype:

```
IImage * ISHELL_LoadResImage(IShell * pIShell, const char * pszResFile, int16 nResID)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Resource file containing the bitmap image
nResID	ID of the bitmap in the resource file

Return Value:

pointer	to an IImage Interface that can be used for viewing the image if succesful
NULL	if unsuccessful

Comments:

This function uses SHELL_LoadResObject

Side Effects:

None

See Also:

ISHELL_LoadImage()
ISHELL_LoadResData()
ISHELL_LoadResObject()
ISHELL_LoadResSound()
ISHELL_LoadResString()
ISHELL_LoadSound()
ISHELL_FreeResData()
Return to the List of functions

ISHELL_LoadResObject()

Description:

This function loads the specified resource and creates an handler that can be used on the resource Data. The type of handler created is indicated by the parameter hType to this function.

Prototype:

```
IBase * ISHELL_LoadResObject(IShell * pIShell, const char * pszResFile, int16
nResID, AEEHandlerType hType)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Resource file containing the specified resource
nResID	ID of the resource in the resource file
hType	Type of handler that must be created and associated with the resource data. The interface pointer returned from this function depends on the handler type specified. The AEEHandlerType must coincide with the actual type of the resource data.

Return Value:

Valid interface pointer to handle the resource data	If successful
NULL	If unsuccessful

Comments:

Example:

If the resource data is of type HTYPE_SOUND and if **hType** parameter is set to HTYPE_VIEWER, the behavior of this function is unpredictable.

If **hType** is HTYPE_VIEWER, it indicates that the resource data is a bitmap resource. Hence, this function creates an IImage Interface and associates the resource data with it. The IImage Interface pointer is then returned from this function. This interface pointer can be used to view the image.

If **hType** is HTYPE_SOUND, it indicates that the resource data is a sound stream. In this case, this function creates and returns a ISoundPlayer Interface pointer that can then be used to play the sound.

The ISHELL_LoadResImage() and ISHELL_LoadResSound() functions are specific usages of this function with the **hType** set to HTYPE_VIEWER and HTYPE_SOUND respectively.

Side Effects:

None

See Also:

ISHELL_LoadImage()
ISHELL_LoadResData()
ISHELL_LoadResImage()
ISHELL_LoadResSound()
ISHELL_LoadResString()
ISHELL_LoadSound()
ISHELL_FreeResData()
Return to the List of functions

ISHELL_LoadResSound()

Description:

This function can be used when the sound data is included in a resource file as a raw stream of bytes. This function loads a raw sound buffer from the given resource file, creates an ISoundPlayer Interface pointer, sets the sound data into this interface by using the ISOUNDPLAYER_Set() function (with AEESoundPlayerInput data parameter set to SDT_BUFFER). The ISoundPlayer Interface pointer can then be used for playing the sound.

Prototype:

```
ISoundPlayer * ISHELL_LoadResSound(IShell * pIShell, const char *
pszResFile, int16 nResID)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Resource file containing the raw sound data
nResID	ID of the raw sound data

Return Value:

Pointer	To an ISoundPlayer Interface used for playing the sound if successful
NULL	If load unsuccessful

Comments:

This function uses ISHELL_LoadResObject().

See Also:

ISHELL_LoadImage()
ISHELL_LoadResData()
ISHELL_LoadResImage()
ISHELL_LoadResObject()
ISHELL_LoadResString()
ISHELL_LoadSound()
Return to the List of functions

ISHELL_LoadResString()

Description:

This function allows the caller to retrieve UNICODE or ISOLATIN strings stored in the specified resource file. The returned string is placed into the buffer provided.

Prototype:

```
int ISHELL_LoadResString(IShell * pIShell, const char * pszResFile, int16
nResID, AECHAR * pBuff, int nSize)
```

Parameters:

pIShell	[in]	Pointer to the IShell Interface object
pszResFile	[in]	Resource file containing the string
nResID	[[in]	ID of the string in the resource file
pBuff	[out]	Buffer to fill with the string
nSize	[in]	Size in bytes of the input buffer

Return Value:

Number	Number of bytes filled, if successful
0 (zero)	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_LoadImage()
ISHELL_LoadResData()
ISHELL_LoadResImage()
ISHELL_LoadResObject()
ISHELL_LoadResSound()
ISHELL_LoadSound()
Return to the List of functions

ISHELL_LoadSound()

Description:

This function loads a sound file from the file system and returns the ISoundPlayer Interface object for the file.

Prototype:

```
ISoundPlayer * ISHELL_LoadSound(IShell * pIShell, const char * pszResFile)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Sound file

Return Value:

Pointer	To an ISoundPlayer Interface that can be used for playing the sound, if successful
NULL	If otherwise

Comments:

This function uses ISHELL_LoadResObject()

Side Effects:

None

See Also:

ISHELL_LoadImage()
ISHELL_LoadResData()
ISHELL_LoadResImage()
ISHELL_LoadResObject()
ISHELL_LoadResSound()
ISHELL_LoadResString()
ISHELL_FreeResData()
Return to the List of functions

ISHELL_MessageBox()

Description:

This function instructs the AEE Shell to display a message box to the user. A message box is a simple window with a title and text. The message box is dismissed via the END or CLR keys. The title and text are retrieved from the specified resource file.

Prototype:

```
boolean ISHELL_MessageBox(IShell * pIShell, const char * pszResFile, uint16
wTitleID, uint16 wTextID)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pszResFile	Resource file containing the title/text. If this is NULL, then the resource ID of the text (for example, **wTextID**) is used to retrieve an error message from the BREW resource file (AEEControls.bar) located in the language directory (ex:"en" for English). This error message is then displayed in the message box.
wTitleID	ID of the title, the maximum size of which can be 128 bytes (64 UNICODE characters)
wTextID	ID of the text, the maximum size of which can be 256 bytes (128 UNICODE characters)

Return Value:

TRUE	If the message box was created
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_MessageBoxText()
Return to the List of functions

ISHELL_MessageBoxText()

Description:

This function instructs the AEE Shell to display a message box to the user. A message box is a simple window with a title and text. The message box is dismissed via the END or CLR keys. Unlike ISHELL_MessageBox(), this function uses the title and text strings provided.

Prototype:

```
boolean ISHELL_MessageBoxText(IShell * pIShell, const AECHAR * pTitle, const AECHAR * pText)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pTitle	Pointer to title string
pText	Pointer to text string

Return Value:

TRUE	If successful
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

ISHELL_MessageBox()
Return to the List of functions

ISHELL_Notify()

Description:

This function is called by an object when it detects an event that may be associated with a notification requested by another class of object. For example, the INetMgr Interface calls this function when INetMgr-related events occur. The result is that any applets that have requested notification, whether actively loaded or not, are sent the EVT_NOTIFY with data regarding the specific event. The notification mask indicates what event occurred. The data pointer provided is specific to the mask of the event that occurred and is defined by the class that triggered the notification. All registered applets for a specific type of notification are called. If the applet is not currently active, it is loaded and the event is sent to it. It is not sent the EVT_APP_START/EVT_APP_STOP events under these conditions. If the applet wishes to start based upon the event it must call ISHELL_StartApplet().

Prototype:

```
void ISHELL_Notify(IShell * pIShell, AEECLSID clsType, uint32 dwMask, void
* pData)
```

Parameters:

pIShell	Pointer to the IShell Interface object
clsType	Class that issued the notification
dwMask	Mask of events in which normally only 1 bit is set for any given event
pData	Context sensitive data

Return Value:

SUCCESS	If successful
EBADCLASS	Invalid ClassID
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory
EREENTERED	Attempt to re-enter ISHELL_Notify()
EBADTASK	Invalid task issuing notify

Comments:

None

Side Effects:

None

See Also:

ISHELL_RegisterNotify()
Return to the List of functions

ISHELL_PostEvent()

Description:

This function posts an asynchronous event to the specified applet. This function is very similar to ISHELL_SendEvent(). The main difference is that this function posts the event to the applet while ISHELL_SendEvent() immediately sends the event to the applet. Event posting is provided for special cases where the caller either wishes to post a event from another task or wishes to defer the processing of the event until the next iteration of the event loop. This is useful in providing continued execution while allowing other events to be processed. Private events can be sent to an applet by defining the event at or above EVT_USER level and specifying the applet ClassID. An error is returned if events in the range of EVT_USER and above are sent without an associated ClassID.

Prototype:

```
boolean ISHELL_PostEvent(IShell * pIShell,AEECLSID clsApp,AEEEvent evt,
uint16 wp, uint32 dwp)
```

Parameters:

pIShell	Pointer to the IShell Interface object
clsApp	ClassID of the applet for the event. This parameter is required and must denote an applet. If the **clsApp** specified here does not belong to an applet, this function may still return TRUE, but the actual sending of the event fails when the event is popped out of the queue and an attempt is made to send to the applet specified by **clsApp**.
evt	AEE Events
wp	Event-specific 16-bit value
dwp	Event-specific 32-bit value

Return Value:

TRUE	If the event was posted. The return status only indicates whether or not the event was successfully placed in the queue. It does not indicate that the event was successfully sent to the receiving applet.
FALSE	If the event was not posted

Comments:

None

Side Effects:

None

See Also:

ISHELL_SendEvent()
Return to the List of functions

ISHELL_Prompt()

Description:

This function provides a mechanism for an application to display a multiselection prompt. The text can be specified from a resource file or directly passed in the AEEPromptInfo data structure. The AEE Shell first examines the text pointers before attempting to load the text from the resource file. The buttons are specified by the values in the **AEEPromptInfo->pBtnIDs** list. For convenience, it is assumed that the IDs for the buttons are associated with both the text and command ID for the button. If the passed-in button pointer is NULL, the prompt must successfully be displayed without any button. The prompt dialog is automatically dismissed when any of the button selections is made by the user or when the CLR key is pressed. In this case, the EVT_COMMAND message is sent to the application with the 16-bit extra parameter indicating the ID of the selection.

Prototype:

```
boolean ISHELL_Prompt(IShell * pIShell, AEEPromptInfo * pi)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pi	Pointer to AEEPromptInfo data structure

Return Value:

TRUE	If the prompt was created
FALSE	If the prompt can not be created

Comments:

None

Side Effects:

None

See Also:

AEEPromptInfo,
Return to the List of functions

ISHELL_QueryClass()

Description:

This function queries the AEE Shell to determine if the specified class or applet is supported. If the AEEAppInfo data structure pointer is provided, AEE Shell assumes the requested class is an applet. In that case, the structure is filled if the requested applet class was found. If the class is supported but is not an applet, the function returns FALSE. If the AEEAppInfo pointer is not passed, the AEE Shell assumes the request is simply made to check the availability of the class, regardless of whether it is an applet.

Prototype:

```
boolean ISHELL_QueryClass(IShell * pIShell, AEECLSID cls, AEEAppInfo * pai)
```

Parameters:

pIShell	[in]	Pointer to the IShell Interface object
cls	[[in]	32-bit ClassID of the requested interface
pai	[out]	Pointer to AEEAppInfo structure to fill with the applet information

Return Value

TRUE	Class is supported
FALSE	Class not supported

Comments:

If you provide an AEEAppInfo, this function returns FALSE if the class requested is available but is not an applet.

Side Effects:

None

See Also:

AEEAppInfo,
ISHELL_CreateInstance()
Return to the List of functions

ISHELL_RegisterHandler()

Description:

This function provides a mechanism for a content viewer or protocol engine to register itself with the AEE Shell. This allows other component to share its functionality when content or protocol types are encountered. As the handler is specified by class, it can be loaded dynamically on an as needed basis. In order to update a handler, the existing handler must be deleted from the database. This is done by calling ISHELL_RegisterHandler() with a 0 (zero) ClassID.

Prototype:

```
int ISHELL_RegisterHandler(IShell * pIShell,AEEHandlerType t, const char *
pszIn, AEECLSID cls)
```

Parameters:

pIShell	Pointer to the IShell Interface object
t	Handler type (HTYPE_VIEWER,HTYPE_BROWSE, or HTYPE_SOUND)
pszIn	Input string. If this is NULL, EBADPARM is returned. The string contains a comma-delimited list of the MIME types and/or schemes handled by the specified class.
cls	AEECLSID of the handler. If this is set to 0 (zero), EBADPARM is returned. No other validations are done on this parameter.

Return Value:

SUCCESS	Operation successful
EBADCLASS	Invalid ClassID
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory
EFAILED	Operation failed
EALREADY	Handler already set.

Comments:

None

Side Effects:

None

See Also:

AEEHandlerType
ISHELL_GetHandler()
Return to the List of functions

ISHELL_RegisterNotify()

Description:

This function provides a clean mechanism for applets to register for notifications that are issued from other classes. For example, an applet can use this mechanism to register for notification from the AEECLSID_NET interface for events specific to that class. The notification mask provided indicates the type of events of interest to the caller. These bits are defined on a per-class basis. This allows classes to define and share notifications without requiring all such notifications to be defined by the IShell Interface. The notification mask specified is used explicitly. The values are not OR'd with the existing notification mask if present. When a notification is requested of a specific object class, the object class is created or the reference count is incremented. When the notification is removed (mask=0), the object's reference count is decremented.

Prototype:

```
int ISHELL_RegisterNotify(IShell * pIShell, AEECLSID clsNotify, AEECLSID
clsType, uint32 dwMask)
```

Parameters:

pIShell	Pointer to the IShell Interface object
clsNotify	The applet to create and notify when the event occurs
clsType	The class that issues the event. This is the Notifier class. This class must implement the INotifier Interface functions.
dwMask	The mask of events to trigger

Return Value:

SUCCESS	If successful
EBADCLASS	Invalid ClassID
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory

Comments:

None

Side Effects:

None

See Also:

ISHELL_Notify()
INOTIFIER_SetMask()
Return to the List of functions

ISHELL_Release()

Description:

This function decrements the reference count of the IShell Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ISHELL_Release(IShell * pIShell)
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

0 (zero)	Always

Comments:

This function doesn't do anything. A call to this function is ignored by AEE.

Side Effects:

None

See Also:

ISHELL_AddRef()
Return to the List of functions

ISHELL_Resume()

Description:

This function allows a callback to be registered with the AEE Shell. It adds the callback to AEE Shell's list of pending operations. The AEE Shell invokes the callback function the next time the event loop is called. This allows an application or object to cooperatively multitask. If the callback has already been registered, it is cancelled (de-registered) and then re-registered.

Prototype:

```
void ISHELL_Resume(IShell * pIShell, AEECallback * pcb)
```

Parameters:

pIShell	Pointer to the IShell Interface object
pcb	Pointer to AEECallback structure. When the application invokes thIs function, the following members in the AEECallback structure must be set by the caller: a. **pfnNotify member** must be filled by the caller. This is the callback function that is invoked by AEE when the event loop is called. b. **pNotifyData** must be filled by caller. This is the data that is passed to the callback function. c. **pfnCancel member** must be set to NULL by the caller before registering the callback for the first time. The AEE Shell fills this member with the right value when this function returns.

Return Value:

None

Comments:

When ISHELL_Resume() is executed, the AEE Shell automatically fills up some of the data members of the AEECallback structure passed to it (for example, the members pfnCancel, pCancelData are filled by the AEE Shell.) To cancel a callback that has been registered, execute the pfnCancel member of the AEECallback structure and pass the pCancelData as a parameter to it.

Example:

```
AEECallback cb;
// Fill up the members of the cb structure
```

```
cb.pfnNotify = MyCB; // My callback function. It must be declared
as void MyCB(void * )

cb.pNotifyData = pme; // Applet Specific data

cb.pfnCancel = NULL; // Initialize to NULL. It is updated by Shell

//To register the callback do the following:

ISHELL_Resume(myShellPtr,&cb) ;

// To cancel the CB do the following:

if(cb.pfnCancel)

cb.pfnCancel(&cb) ;
```

Side Effects:

None

See Also:

AEECallback
Return to the List of functions

ISHELL_SendEvent()

Description:

The entire execution model of the AEE is based around a semi-cooperative event passing model. Under this model, events are sent to the active dialog or applet using the ISHELL_SendEvent() function. The ISHELL_SendEvent() function allows for control over the destination applet. This function sends the event directly to the destination applet. Events from one task to another must always be posted. Attempts to send the events from one task to another is rejected. Events can be sent to a specific applet by specifying the destination applet. If the applet is not currently running it is loaded and the event is sent directly to it. Under these conditions, the applet is started in the background (EVT_APP_START/EVT_APP_STOP events are not sent to the applet). Private events can be sent to an applet by defining the event at or above EVT_USER and specifying the applet ClassID. An error is returned if events in the range of EVT_USER and above are sent without an associated ClassID. If there is a dialog active (ISHELL_CreateDialog()) , the event is passed to the dialog before being passed to the applet. The event is only passed to the applet if the dialog did not process the event. The **wp** and **dwp** parameters associated with the event are specific to the event. The AEE Shell does not examine these values.

Prototype:

```
boolean ISHELL_SendEvent(IShell * pIShell, AEECLSID clsApp,AEEEvent evt,
uint16 wp, uint32 dwp)
```

Parameters:

pIShell	Pointer to the IShell Interface object
clsApp	Class of the applet for the event. This parameter is required for events in the range of EVT_USER and above.
evt	AEE Events
wp	Event-specific 16-bit value
dwp	Event-specific 32-bit value

Return Value:

TRUE	If the event was processed
FALSE	If the event was not processed

Comments:

None

Side Effects:

None

See Also:

ISHELL_PostEvent()
Return to the List of functions

ISHELL_SetAlarm()

Description:

This function allows the caller to set a long-term alarm for an applet. When the alarm expires, the applet is loaded and passed an EVT_ALARM event with the specified 16-bit nUserCode as the user parameter. If the applet is not active at the time of the alarm, it is loaded but is not sent an EVT_APP_START event. If the applet wishes to be activated, it must call ISHELL_StartApplet(). More than one alarm can be set for an applet by specifying a different 16-bit alarm nUserCode.

Prototype:

```
int ISHELL_SetAlarm(IShell * pIShell, AEECLSID cls, uint16 nUserCode, uint32 nMins)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	Applet class to call when the alarm expires
nUserCode	16-bit code passed to the applet
nMins	Number of minutes to set the alarm from the current time

Return Value:

SUCCESS	if operation is successful
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory

Comments:

None

Side Effects:

None

See Also:

ISHELL_CancelAlarm()
Return to the List of functions

ISHELL_SetPrefs()

Description:

This function provides a means of storing a structure containing applet or class level preferences.

Prototype:

```
int ISHELL_SetPrefs(IShell * pIShell, AEECLSID cls, uint16 wVer, void * pCfg,
uint16 nSize)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	AEECLSID of the preference type
wVer	Version of the preference
pCfg	Pointer to the preference data to be stored
nSize	Size of memory block to store

Return Value:

SUCCESS	Operation successful
EBADPARM	Invalid parameter
ENOMEMORY	Insufficient memory
EFAILED	Operation failed

Comments:

None

Side Effects:

None

See Also:

ISHELL_GetPrefs(),
Return to the List of functions

ISHELL_SetTimer()

Description:

This function allows the caller to set a short-term timer. Upon expiration, the specified callback function is called, passing it the specified user data pointer as its first argument. Note the following:

- The timer expires at Current Time + <Milliseconds specified>
- Timer callbacks are made in the application's task state. The system shields the application developer from managing non-task callbacks.
- Any normal processing can be done in the callback. This includes drawing to the screen, writing to files, and other items.
- Timers do not repeat. The users must restart the timer if they desire a repeating timer.
- Specifying the same callback/data pointers automatically overrides a pending timer with the same callback/data pointers.
- Upon termination of the currently active applet, the AEE Shell scans the timer list. If the terminated applet was deleted as a result of its termination (that is the reference count went to 0), and an associated timer was found with the data pointer pointing to the applet, the timer is deleted.

Prototype:

```
int ISHELL_SetTimer(IShell * pIShell, int32 dwMSecs, PFNNOTIFY pfn, void *
pUser)
```

Parameters:

pIShell	Pointer to the IShell Interface object
dwMSecs	Timer expiration in milliseconds. The expiration occurs at Current Time + **dwMSecs.**
pfn	The user callback that is called when the timer expires
pUser	The user data pointer that is passed as the only parameter to the callback

Return Value:

SUCCESS	Timer successfully set
EBADPARM	Invalid parameter

EINVALIDTIME	Invalid expiration time
ENOMEMORY	Not enough memory left on heap to create timer

Comments:

None

Side Effects:

None

See Also:

ISHELL_GetTimerExpiration()
ISHELL_CancelTimer()
Return to the List of functions

ISHELL_ShowCopyright()

Description:

This function provides a means for the application to display the copyright information about the applet. An applet can use this function to display copyright and other applet related information on the screen. The information to be displayed is obtained from the Module Information File (MIF) for that applet. The following pieces of information can be displayed:

- Application Name
- Application Icon
- Copyright String
- Company Name

These four items are obtained from the MIF for that applet.

NOTE: The lengths of the strings must be less than 24 characters for them to be displayed. This information is displayed in the form of a dialog box, and it stays on the screen until the user dismisses the dialog (by pressing the clear button).

Prototype:

```
boolean ISHELL_ShowCopyright(IShell * pIShell);
```

Parameters:

pIShell	Pointer to the IShell Interface object

Return Value:

TRUE	If the copyright information was successfully displayed
FALSE	If the copyright information was not successfully displayed

Comments:

This function internally uses ISHELL_Prompt() for the implementation.

Side Effects:

None

See Also:

None
Return to the List of functions

ISHELL_StartApplet()

Description:

This function instructs the AEE Shell to start the applet associated with the specified 32-bit ClassID. If the requested class is supported and can be started, the AEE Shell loads and starts the applet. This call returns immediately to the caller before starting the applet. The applet is started asynchronously. Hence, if the applet classID being specified is not found, this function returns TRUE but the applet is not started. When the applet is started or when it is resumed the display is cleared and the IAPPLET_HandleEvent() is sent the EVT_APP_START or EVT_APP_RESUME event along with the AEEAppStart parameter block. If an applet is started by another applet, the currently active applet is suspended.

Prototype:

```
int ISHELL_StartApplet(IShell * pIShell, AEECLSID cls)
```

Parameters:

pIShell	Pointer to the IShell Interface object
cls	32-bit applet ClassID

Return Value:

SUCCESS	Applet created and started
ENOMEMORY	Insufficient memory
ECLASSNOSUPPORT	Class specified is not supported
EEXPIRED	Applet has expired

Comments:

None

Side Effects:

None

See Also:

ISHELL_CreateInstance()
ISHELL_CanStartApplet()
ISHELL_CloseApplet()
ISHELL_ActiveApplet()
Return to the List of functions

ISocket Interface

The ISocket Interface provides methods to connect, transmit and receive data over, and close TCP and UDP sockets that are opened using the INETMGR_OpenSocket().

> **CAUTION:** Your application must have a privilege level of Network or All to be able to invoke the functions in this interface.

The function ISOCKET_Connect() is called immediately after opening a socket. If the network subsystem (physical layer, RLP and PPP) of the device is not active, this function first establishes the necessary lower-layer protocol connections. For TCP sockets, ISOCKET_Connect() then sets up a TCP connection to the specified IP address and port number. For UDP sockets, the IP address and port number supplied when calling ISOCKET_Connect() are used as defaults for write operations to the socket. Therefore, for UDP sockets, the caller may specify the destination IP address and port numbers for each read or write operation, thereby allowing data to be sent to and received from multiple IP addresses and ports.

Once the socket has been connected, read and write operations may be used to exchange data over it. All data transfer operations are non-blocking; callback functions are used to notify the caller of the availability of socket for read or write operations. The caller can use ISOCKET_Readable() to provide the address of a callback function that must be invoked when there is data available to read. Similarly, ISOCKET_Writeable() registers the callback function that gets invoked when the socket is available for writing. Callbacks are also invoked whenever BREW detects any error conditions on the socket that require the application using this interface to take action. The function ISOCKET_Cancel() is used to cancel a pending callback operation, thereby preventing the application from receiving notification when there is a change in the status of the socket it is waiting on.

The functions ISOCKET_Read() and ISOCKET_Write() are used for reading data from and writing data to TCP sockets. If the read or write do not make progress, ISOCKET_Read() and ISOCKET_Write()return an indication that blocking has occured. The caller can then use ISOCKET_Readable() or ISOCKET_Writeable() to arrange to be notified when blocking is no longer present. If the number of bytes actually read or written is less than the number requested, repeat calls to ISOCKET_Read() or ISOCKET_Write() to complete the data transfer.

The functions ISOCKET_ReadV() and ISOCKET_WriteV() are used to receive and send data on TCP sockets when the application uses multiple, non-contiguous buffers for reading and writing. In place of the single buffer pointer supplied as a parameter to ISOCKET_Read() and ISOCKET_Write()), ISOCKET_ReadV() and ISOCKET_WriteV() each take an array of buffer descriptors as input, with each

array element specifying the length in bytes and starting address of a buffer. These functions attempt to transfer an amount of data equal to the sum of the buffer lengths, starting with the first buffer in the array.

The functions ISOCKET_RecvFrom() and ISOCKET_SendTo() are used to exchange data over UDP sockets. Both these functions allow data to be sent to and received from multiple IP addresses and port numbers. As with TCP sockets, the application may call ISOCKET_Readable() or ISOCKET_Writeable() to designate a callback function if the read or write operation does not make progress immediately.

The function ISOCKET_Bind() associates a local port number with a socket (if an application does not call this function before ISOCKET_Connect(), a default value is used for the local port number). The function ISOCKET_GetLastError() returns the error code for the last error detected by a function in the ISocket Interface . In cases where such a function returns something other than success, ISOCKET_GetLastError() provides more detailed information about why the function failed to perform its task. The function ISOCKET_GetPeerName() returns the IP address and port number of the entity with which data was most recently exchanged on the socket.

After completion of data transfer, a call to ISOCKET_Release() closes the socket connection and releases the ISocket Interface and frees the associated resources.

To use the socket services, perform the following steps:

1. Call ISHELL_CreateInstance() to create an instance of the INetMgr Interface.
2. Call INETMGR_OpenSocket() to create an instance of the ISocket Interface (to open a TCP or UDP socket).
3. Call ISOCKET_Connect() to establish the socket connection with the network entity with which the application will communicate.
4. Use the functions in the ISocket Interface to operate on the socket.

To read or write data over the socket:

1. Call the relevant (TCP/UDP, single/multiple I/O buffers) read or write functions described above.

2 If the function returns a blocking indication, call ISOCKET_Readable() or ISOCKET_Writeable() to attempt the operation at a later time.

3 If less than the requested number of bytes were transferred, call the read or write function again to effect the transfer of the remaining data.

4 Repeat steps 1-3, until all of your data has been transferred.

5 Call ISOCKET_Bind() to specify the socket's local port number.

6 Call ISOCKET_GetPeerName() to obtain the IP address and port number of the most recent communication partner on the socket.

7 Call ISOCKET_Release() to close the socket when after completing the data transfer.

8 Call INETMGR_Release() to release the INetMgr Interface instance.

List of functions

Functions in this interface include:

ISOCKET_AddRef()

ISOCKET_Bind()

ISOCKET_Cancel()

ISOCKET_Connect()

ISOCKET_GetLastError()

ISOCKET_GetPeerName()

ISOCKET_IOCtl()

ISOCKET_Read()

ISOCKET_Readable()

ISOCKET_ReadV()

ISOCKET_RecvFrom()

ISOCKET_Release()

ISOCKET_SendTo()

ISOCKET_Writeable()

ISOCKET_Write()

ISOCKET_WriteV()

Return to the Contents

ISOCKET_AddRef()

Description:

This function increments the reference count of the ISocket Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ISOCKET_Release().

Prototype:

```
uint32 ISOUNDPLAYER_AddRef(ISocket * pISocket)
```

Parameters:

pISocket Pointer to the ISocket Interface object

Return Value:

Incremented reference count for the object. .

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

ISOCKET_Release()
Return to the List of functions

ISOCKET_Bind()

Description:

This function associates a local address and port with the socket. The ISOCKET_Bind() function is used on an unconnected socket. It is used to bind to either connection-oriented (stream) or connection-less (datagram) sockets. When a socket is created with a call to the socket function, it exists in a name space (address family), but it has no name assigned to it. Passing AEE_INADDR_ANY (zero) for the address explicitly requests that the socket be assigned to any local address.

This ability to bind to a specific local IP address is not presently supported, and AEE_INADDR_ANY is the only valid value for the INAddr parameter. The local IP address is assigned automatically by the sockets library.

Also note the possibility of an AEE_NET_WOULDBLOCK result. This normally occurs only in cases where a network connection (for example, a PPP link) must be established. In that event, ISOCKET_Bind() returns AEE_NET_WOULDBLOCK, and can be called again to obtain the final result (error or success). For notification of when to call ISOCKET_Bind() again, the ISOCKET_Writeable() call can be used. Just as with ISOCKET_Write(), a ISOCKET_Writeable() callback guarantees completion of the ISOCKET_Bind() call, so the caller must be prepared to receive AEE_NET_WOULDBLOCK again.) The user is not required to call ISOCKET_Bind() again; the user can proceed to attempt to send and/or receive data using the ISOCKET_SendTo() or ISOCKET_RecvFrom() -- the bind operation is attempted in the background when possible.

Prototype:

```
int ISOCKET_Bind(ISocket * pISocket, INAddr a, uint16 wPort)
```

Parameters:

pISocket	Pointer to the ISocket Interface object that needs to be connected
a	IP Address
wPort	Port

Return Value:

AEE_NET_SUCCESS	If successful
AEE_NET_ERROR	On error

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError(). One of the following error codes is returned:

AEE_NET_EBADF	Invalid socket descriptor is specified
AEE_NET_EOPNOTSUPP	Operation not supported
AEE_NET_EADDRINUSE	Local address is already in use
AEE_NET_EINVAL	Socket is already attached to a local name
AEE_NET_EFAULT	Invalid address parameter has been specified
AEE_NET_EAFNOSUPPORT	Address family not supported
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer
AEE_NET_WOULDBLOCK	No data available now; try again later (See ISOCKET_Readable())

Specific addresses not currently supported.

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Cancel()

Description:

This function allows an application to cancel (un-register) a callback function that has been previously registered using the ISOCKET_Readable() or ISOCKET_Writeable(). Cancellation simply annuls the effect of the prior ISOCKET_Readable() or ISOCKET_Writeable() , so that the callback function is not called when the socket makes progress. When both the **pfn** and **pUser** parameters are NULL, all registered callbacks are canceled.

Prototype:

```
void ISOCKET_Cancel(ISocket * pISocket, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pISocket	Pointer to the ISocket Interface for which the callback function needs to be de-registered
pfn	Pointer to the callback function that needs to be cancelled
pUser	Callback data that was used to register the callback

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Connect()

Description:

For sockets of type AEE_SOCK_STREAM, this attempts to initiate a TCP connection to the specified address and port. For sockets of type AEE_SOCK_DGRAM, the specified host and port are associated with the socket, and they serve as the default destination address for subsequent ISOCKET_Write() calls.

Prior to performing the above socket-specific behavior, ISOCKET_Connect() tries to acquire an IP address and ensure the underlying network layer is ready for communication. This may involve the establishment of an Internet connection using CDMA Packet Data or QNC if such a connection has not already been established.

After establishment of a connection to the Internet (and in the case of AEE_SOCK_STREAM sockets, the TCP connection) or in the event of a failure, the specified callback is called. The callback is passed the error code that describes how the connect operation completed. As with all network callbacks, the callback is called within the same thread context, at some point in time after the caller returns control to the AEE event loop.

Prototype:

```
int ISOCKET_Connect(ISocket * pISocket, INAddr a, INPort wPort, PFNCONNECTCB pfn, void * pUser)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
a	IP Address
wPort	Port
pfn	Address of the callback function that is invoked by AEE when the connect operation either succeeds or fails
pUser	User defined data that is passed to the callback function when it is invoked

Return Value:

AEE_NET_SUCCESS	If successful, this indicates that the callback is called after the caller relinquishes control
AEE_NET_ERROR	If the call can not schedule the callback (for example, the callback function or parameter was NULL)

Comments:

The **INAddr** and **INPort** arguments are assumed to be in network byte order (for example, big-endian). This issue is important for portability across simulator environments and potential device targets.

The callback is passed one of the following values describing the completion of the connect operation:

AEE_NET_SUCCESS	Connect completed successfully and the socket is prepared for reading and/or writing
AEE_NET_EBADF	Invalid socket descriptor is specified
AEE_NET_ECONNREFUSED	Connection attempt refused
AEE_NET_ETIMEDOUT	Connection attempt timed out
AEE_NET_EFAULT	address parameter is invalid
AEE_NET_EIPADDRCHANGED	IP address changed due to PPP resync
AEE_NET_EISCONN	Socket is already connected (result was already received by a previous call to Connect())
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EOPNOTSUPP	Invalid server address specified
AEE_NET_EADDRREQ	Destination address is required
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_GetLastError()

Description:

This function returns the last error that occurred with the given Socket. The value returned depends on the most recently called function. The documentation for each function describes what this function can return when called right after that function.

Prototype:

```
int ISOCKET_GetLastError(ISocket * pISocket)
```

Parameters:

pISocket	Pointer to the ISocket Interface object

Return Value:

The most recently occurred error

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functionst

ISOCKET_GetPeerName()

Description:

This function returns the IP address and port of the peer (for example, the IP address of the entity that this socket last communicated with).

Prototype:

```
int ISOCKET_GetPeerName(ISocket * pISocket, INAddr * pa, uint16 * pwPort)
```

Parameters:

pISocket	Pointer to the I ISocket Interface object
pa	Pointer to IP Address
pwPort	Pointer to port

Return Value:

AEE_NET_SUCCESS	If successful
AEE_NET_EBADF	If unsuccessful or socket descriptor is invalid

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_IOCtl()

Description:

This function is under development and not available, as a result a call always returns EFAILED.

Prototype:

```
int ISocket_IOCtl(ISocket * pISocket, int nOption, uint32 dwVal)
```

Parameters:

pISocket	Function is under development and not available
nOption	Function is under development and not available
dwVal	Function is under development and not available

Return Value:

EFAILED	Function is under development and not available.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Read()

Description:

This function reads data from a socket. It reads data into a single buffer specified as a parameter to this function. This function always returns immediately. If there is no data available and the connection is still active, ISOCKET_Read() returns AEE_NET_WOULDBLOCK. To be notified when to call ISOCKET_Read() again, the caller must call ISOCKET_Readable().

Prototype:

```
int32 ISOCKET_Read(ISocket * pISocket, byte * pbDest, uint16 wSize)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
pbDest	Pointer to the buffer to hold the data that is to be received
wSize	Specifies the number of bytes to read

Return Value:

bytes_read (> 0)	Any positive number indicates a number of bytes that have been successfully read into the provided buffer
0 (zero)	There is no more data to be received; the peer has shut down the connection
AEE_NET_WOULDBLOCK	No data available now; try again later (See ISOCKET_Readable())
AEE_NET_ERROR	The socket is not in a valid state to receive data

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError(). Error codes returned:

AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_ENOTCONN	Socket not connected
AEE_NET_ECONNRESET	TCP connection reset by server
AEE_NET_ECONNABORTED	TCP connection aborted due to timeout or other failure
AEE_NET_EIPADDRCHANGED	IP address changed, causing TCP connection reset

AEE_NET_EPIPE	Broken pipe
AEE_NET_EADDRREQ	Destination address required
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Readable()

Description:

This function allows an application to register a callback function to be invoked by the AEE when a non-blocking read operation (Read, ReadV, or RecvFrom) on the specified socket can make progress. Progress can involve returning data, returning an error code, or returning 0 (zero) to indicate a closed connection -- anything but AEE_NET_WOULDBLOCK. This function would typically be used after a previous read attempt returned AEE_NET_WOULDBLOCK, but it can be used even if no read function has been called.

NOTE: There is no absolute guarantee that the read function makes progress after the callback, so the caller must always be prepared to call ISOCKET_Readable() again when the read function return AEE_NET_WOULDBLOCK.

Prototype:

```
void ISOCKET_Readable(ISocket * pISocket, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
pfn	Address of the callback function. This function is invoked by AEE when the socket becomes ready to be read or when it is ready to be closed. If this is NULL, it cancels the Readable callback function registered by a previous call to ISOCKET_Readable (if any). It always returns AEE_NET_SUCCESS if pfn is passed as NULL.
pUser	User defiend data that is passed to the callback function when it is invoked

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_ReadV()

Description:

This function reads data from a socket. It reads data into one or more buffers described by the entries in the iov[] array. This function always returns immediately. If there is no data available and the connection is still active, ISOCKET_ReadV() returns AEE_NET_WOULDBLOCK. To be notified when to call ISOCKET_ReadV() again, the caller must call ISOCKET_Readable().

Prototype:

```
int32 ISOCKET_ReadV(ISocket * pISocket, SockIOBlock iov[] , int16 iovcount)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
iov	Array of SockIOBlock structures into which data can be read
iovcount	Specifies the number of entries in the **iov** array

Return Value:

bytes_read (> 0)	Any positive number indicates the total of all the bytes read into the provided buffers
0 (zero)	There is no more data to be received; the peer has shut down the connection
AEE_NET_WOULDBLOCK	No data available now; try again later (See ISOCKET_Readable())
AEE_NET_ERROR	The socket is not in a valid state to receive data

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError().

Error codes returned:

AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_ENOTCONN	Socket not connected
AEE_NET_ECONNRESET	TCP connection reset by server
AEE_NET_ECONNABORTED	TCP connection aborted due to timeout or other failure

AEE_NET_EIPADDRCHANGED	IP address changed, causing TCP connection reset
AEE_NET_EPIPE	Broken pipe
AEE_NET_EADDRREQ	Destination address required
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_EWOULDBLOCK	Operation would block
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_RecvFrom()

Description:

This functions reads data from UDP socket and records the IP address and port of the sender. This function always returns immediately with the number of bytes read. If no packets have arrived and the socket is still in a valid state, ISOCKET_RecvFrom() returns AEE_NET_WOULDBLOCK. ISOCKET_Readable() may be used to receive notification of when to try ISOCKET_RecvFrom() again.

Prototype:

```
int32 ISOCKET_RecvFrom(ISocket * pISocket, byte * pBuff, uint16 wbytes,
uint16 wflags, INAddr * pa, INPort * pwPort)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
pBuff	Buffer to hold the received data
wBytes	Size of the buffer, in terms of number of bytes
wflags	Not used and must be set to 0
pa	Pointer to IP Address
pwPort	Pointer to port

Return Value:

bytes_read (> 0)	Any positive number indicates a number of bytes that have been successfully read into the provided buffer
0 (zero)	There is no more data to be received and the peer has shut down the connection
AEE_NET_WOULDBLOCK	No data available now; try again later (See ISOCKET_Readable())
AEE_NET_ERROR	Socket is not in a valid state to receive data

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError() and one of the following error codes is returned:

AEE_NET_EEOF	End of file
AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_EAFNOSUPPORT	Address family not supported

AEE_NET_EADDRREQ	Destination address required
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_EOPNOTSUPP	Option not supported
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Release()

Description:

This function decrements the reference count for the ISocket Interface object. If the reference count reaches 0 (zero), the funhction closes the socket.

Prototype:

```
uint32 ISOCKET_Release(ISocket * pISocket)
```

Parameters:

pISocket Pointer to the ISocket Interface object

Return Value:

The updated reference count

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_SendTo()

Description:

This function sends a UDP packet to the specified IP address and port from the local port bound to the socket. This function must be used with sockets of type AEE_NET_DGRAM (UDP) -- not with sockets of type AEE_NET_STREAM (TCP). If the socket has not been bound to a local address and port, this function binds it to a port.

Sockets always operate in non-blocking mode. This function returns immediately and, if successful, returns the number of bytes written. If no bytes can be successfully sent and the connection is still active, the function returns AEE_NET_WOULDBLOCK. To be notified when to call ISOCKET_SendTo() again, the caller must call ISOCKET_Writeable().

Prototype:

```
int32 ISOCKET_SendTo(ISocket * pISocket,
         byte * pBuff,
         uint16 wBytes,
         uint16 wflags,
         IPAddr a,
         INPort wPort)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
pBuff	Buffer containing data to be sent
wBytes	Size of the buffer, in terms of number of bytes
wflags	Not used and must be set to 0
a	IP Address
wPort	Port

Return Value:

AEE_NET_WOULDBLOCK	Cannot send data at this time. Use ISOCKET_Writeable() to wait for readiness. This condition can be encountered if transmit buffers have been filled, or when a network connection is being established.
AEE_NET_ERROR	Indicates failure

Comments:

The specific error code can be retrieved by calling GetLastError(). One of the following error codes is returned:

AEE_NET_EEOF	End of file
AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_EAFNOSUPPORT	Address family not supported
AEE_NET_EADDRREQ	Destination address required
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_EOPNOTSUPP	Option not supported
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Writeable()

Description:

This function allows an application to register a callback function to be invoked by the AEE when a non-blocking write operation (ISOCKET_Write(), ISOCKET_WriteV(), or ISOCKET_SendTo()) on the specified socket can make progress. Progress can involve writing data or returning an error code -- anything but AEE_NET_WOULDBLOCK. This function would typically be used after a previous write attempt returned AEE_NET_WOULDBLOCK, but it can be used even if no read function has been called.

> **NOTE:** There is no absolute guarantee that the read function makes progress after the callback, so the caller must always be prepared to call ISOCKET_Readable() again when the write function return AEE_NET_WOULDBLOCK.

Prototype:

```
void ISOCKET_Writeable(ISocket * pISocket, PFNNOTIFY pfn, void * pUser)
```

Parameters:

pISocket	Pointer to the ISocket Interface object for which the callback function needs to be registered
pfn	Address of the callback function. This function is invoked by AEE when the socket becomes ready to receive data or when it is ready to be closed. If this is NULL, it cancels the Writeable callback function registered by a previous call to ISOCKET_Writeable().
pUser	User defined data that is passed to the callback function when it is invoked

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_Write()

Description:

This function writes data to a connected socket. It writes data from a single buffer. This function always returns immediately with the number of bytes that were successfully written. If no bytes can be successfully written and the connection is still active, ISOCKET_Write() returns AEE_NET_WOULDBLOCK. To be notified when to call ISOCKET_Write() again, the caller must call ISOCKET_Writeable().

Prototype:

```
int32 ISOCKET_Write(ISocket * pISocket, byte * pBuffer, uint16 wBytes)
```

Parameters:

pISocket	Pointer to the I ISocket Interface object
pBuffer	Pointer to the buffer from which the data is sent
wByteCount	Specifies the size of the buffer in terms of number of bytes in the buffer

Return Value:

bytes_written (> 0)	Any positive number indicates the number of bytes successfully written
AEE_NET_WOULDBLOCK	No bytes can be written at this time; try again later (See ISOCKET_Writeable())
AEE_NET_ERROR	Failed to write any bytes

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError(). One of the following possible error codes is returned.

AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_ENOTCONN	Socket not connected
AEE_NET_ECONNRESET	TCP connection reset by server
AEE_NET_ECONNABORTED	TCP connection aborted due to timeout or other failure
AEE_NET_EIPADDRCHANGED	IP address changed, causing TCP connection reset
AEE_NET_EPIPE	Broken pipe
AEE_NET_EADDRREQ	Destination address required

AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISOCKET_WriteV()

Description:

This function writes data to a connected socket. It gathers data from one or more buffers described by the entries in the iov[] array into a single write operation. With AEE_SOCK_STREAM (TCP) sockets, behaviour can differ from separate calls to ISOCKET_Write() in that multiple calls to ISOCKET_Write() might unnecessarily generate multiple TCP packets. With AEE_SOCK_DGRAM (UDP) sockets, the separate buffers are treated as an individual packet.

This function always returns immediately with the number of bytes that were successfully written. If no bytes can be successfully written and the connection is still active, ISOCKET_Write() returns AEE_NET_WOULDBLOCK. To be notified when to call ISOCKET_WriteV() again, the caller must call ISOCKET_Writeable().

Prototype:

```
int32 ISOCKET_WriteV(ISocket * pISocket, SockIOBlock iov[] , uint16
wiovcount)
```

Parameters:

pISocket	Pointer to the ISocket Interface object
iov	An array of SockIOBlocks containing the individual buffers to be sent
wiovcount	Number of entries inside the SockIOBlock structure

Return Value:

bytes_written (> 0)	Any positive number indicates the total number of bytes successfully written from all of the iov[] buffers
AEE_NET_WOULDBLOCK	No bytes can be written at this time; try again later (See ISOCKET_Writeable()
AEE_NET_ERROR	Failed to write any bytes

Comments:

The specific error code can be retrieved by calling ISOCKET_GetLastError(). One of the following error codes is returned

AEE_NET_EBADF	Invalid socket descriptor is specfied
AEE_NET_ENOTCONN	Socket not connected
AEE_NET_ECONNRESET	TCP connection reset by server
AEE_NET_ECONNABORTED	TCP connection aborted due to timeout or other failure
AEE_NET_EIPADDRCHANGED	IP address changed, causing TCP connection reset
AEE_NET_EPIPE	Broken pipe
AEE_NET_EADDRREQ	Destination address required
AEE_NET_ENETDOWN	Network subsystem unavailable
AEE_NET_EFAULT	Application buffer not valid part of address space
AEE_NET_GENERAL_FAILURE	NULL socket interface pointer

Side Effects:

None

See Also:

None
Return to the List of functions

ISound Interface

The **ISound Interface** provides the basic sound services. These services enable playing of beeps, rings, vibrations, various tones and list of tones. ISound defines a set of tones that can be played for a period of time or continuously until explicitly stopped. The tone identifiers can be put in a list and all the tones in the list can be played using a single function call. The **ISound Interface** also provides functions to get and set the volume.

To use the ISound Interface

1 Create an instance of the ISound Interface with ClassID: **AEECLSID_SOUND**

2 Register the status callback function using the ISOUND_RegisterNotify(). ISound always communicates with applets through this callback function. The Callback function carries information regarding user data, status callback types (AEE_SOUND_STATUS_CB, AEE_SOUND_VOLUME_CB) and status (AEE_SOUND_SUCCESS, AEE_SOUND_PLAY_DONE, AEE_SOUND_FAILURE) Data, if any, is returned in the callback through the **dwParam** parameter which points to AEESoundPlayerCmdData. Otherwise **dwParam** is NULL. It is not always required to register a callback function; applet developer can choose not to get any ISound events as follows:

```
ISOUND_RegisterNotify(pISound, NULL, NULL);
```

3 Set ISound parameters using the ISOUND_Set(). This step is optional and ISound assumes default values for the parameters if it is not performed. After these steps, ISound is ready for service. See the ISound function description for more details on each function.

Tone-Database:

On the target device, device manufacturers can map elements of the AEESoundTone type to the corresponding IDs in the tone database. For example, AEE_TONE_0 maps to SND_0 on an MSM3300-based device.

In the BREW Emulator, device manufacturers or applet developers can define their own tone files in .WAV format and add them to BREW Emulator tone-database. The tones are saved in the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed) based on AEESoundTone type. For example, to add AEE_TONE_0 to BREW Emulator, save your .WAV file as "AEE_TONE_0.wav" in the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed). Same mechanism applies to BeepType required in ISHELL_Beep() function. For example,

to create your own BEEP_ALERT tone, save your .WAV file as "BEEP_ALERT.wav" in the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed). A few sample tones and all beep types are provided. Their usage is illustrated in the sample Sound applet.

List of functions

Functions in this interface include:

ISOUND_AddRef()

ISOUND_Get()

ISOUND_GetVolume()

ISOUND_PlayFreqTone()

ISOUND_PlayTone()

ISOUND_PlayToneList()

ISOUND_RegisterNotify()

ISOUND_Release()

ISOUND_Set()

ISOUND_SetDevice()

ISOUND_SetVolume()

ISOUND_StopTone()

ISOUND_StopVibrate()

ISOUND_Vibrate()

Return to the Contents

ISOUND_AddRef()

Description:

This function increments the reference count of the ISound Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ISOUND_Release().

Prototype:

```
uint32 ISOUND_AddRef(ISound * pISound)
```

Parameters:

pISound	Pointer to the ISound Interface object

Return Value:

Returns the incremented reference count for the object. A valid object returns a positive reference count.

Comments:

None

Side Effects:

None

See Also:

ISOUND_Release()
Return to the List of functions

ISOUND_Get()

Description:

This function gets the device attributes of ISound Interface object.

Prototype:

```
void ISOUND_Get(ISound * pISound, const AEESoundInfo * pSoundInfo)
```

Parameters:

pISound	[in]	Pointer to ISound Interface object
pSoundInfo	[out]	Structure containing ISound device attributes

Return Value:

None

Comments:

See ISOUND_Set() function description for details on AEESoundInfo

Side Effects:

None

See Also:

None
Return to the List of functions

ISOUND_GetVolume()

Description:

This function retrieves the current volume used for the device/method pair.

Prototype:

```
void ISOUND_GetVolume(ISound * pISound)
```

Parameters:

pISound	Pointer to ISound Interface object

Return Value:

None

Comments:

None

Side Effects:

The volume level is sent back to the client via the callback function the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed). The result includes the status as well as the current volume level. The status can be one of the following values:

AEE_SOUND_SUCCESS	Successful retrieving the current volume level. The **dwParam** of the callback function contains volume
AEE_SOUND_FAILURE	Failed to retrieve the current volume level and the volume level is reset to 0

The current volume level is included in the status event as additional data.

See Also:

None
Return to the List of functions

ISOUND_PlayFreqTone()

Description:

This function plays a tone given a high frequency and a low frequency for the specified amount of time.

Prototype:

```
void ISOUND_PlayFreqTone(ISound * pISound, uint16 wHiFreq, uint16 wLoFreq,
uint16 wDuration)
```

Parameters:

pISound	Pointer to ISound Interface object
wHiFreq	Higher frequency of the DTMF pair
wLoFreq	Lower frequency of the DTMF pair
wDuration	Tone play duration in milliseconds

Return Value:

None

Comments:

If the duration is set to 0 (zero), the tone plays until the ISOUND_StopTone() call is issued. This function is not supported in the BREW Emulator but it is supported on the target device.

Side Effects:

The result of the operation is sent to the client via the callback function pointer. Also, when the operation is done, it notifies the client through the same callback function pointer.

The status can be one of the following values:

AEE_SOUND_SUCCESS	Successful playing the tone
AEE_SOUND_FAILURE	Failed playing the tone
AEE_SOUND_PLAY_DONE	Done playing the tone or replaced by another tone

See Also:

None
Return to the List of functions

ISOUND_PlayTone()

Description:

This function plays a tone given a tone ID for the specified amount of time.

Prototype:

```
void ISOUND_PlayTone(ISound * pISound, AEESoundToneData toneData)
```

Parameters:

pISound	Pointer to ISound Interface object
toneData	Structure containing a ToneID and the duration in milliseconds for which that tone is to be played

Return Value:

None

Comments:

If the duration is set to 0 (zero) for any tone in the list, the tone plays until the ISOUND_StopTone() call is issued.

Side Effects:

The result of the operation is sent to the client via the callback function the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed). Also, when the operation is done, it notifies the client through the same callback function the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed).

The status can be one of the following values:

AEE_SOUND_SUCCESS	Successful playing the tone
AEE_SOUND_FAILURE	Failed to play the tone
AEE_SOUND_PLAY_DONE	Done playing the tone or replaced by another tone

See Also:

None
Return to the List of functions

ISOUND_PlayToneList()

Description:

This function plays a list of tones, given a tone ID for the specified amount of time for each tone.

Prototype:

```
void ISOUND_PlayToneList(ISound * pISound, AEESoundToneData * pToneData,
uint16 wDataLen)
```

Parameters:

pISound	Pointer to ISound Interface object
pToneData	List(Array) of AEESoundToneData structures, each containing **ToneID** and duration in milliseconds to be played for that tone
wDataLen	Number of tones in the pToneData list

Return Value:

None.

Comments:

Call ISOUND_StopTone() to end the playing of tone list. If the duration is set to 0 (zero) for any tone in the list, the tone plays until the ISOUND_StopTone() call is issued.

Side Effects:

The result of the operation is sent to the client, for each tone in the tone list, via the callback function pointer. Also, when the operation is done, it notifies the client through the same callback function pointer. The status can be one of the following values:

AEE_SOUND_SUCCESS	Received for each successful playing of tone in the tone list
AEE_SOUND_FAILURE	Failed playing the tone in the tone list and tone list playback ends
AEE_SOUND_PLAY_DONE	Received for each tone, in the tone list, after the tone playback is complete or replaced by another tone

See Also:

None

Return to the List of functions

ISOUND_RegisterNotify()

Description:

This function registers the status callback function.

Prototype:

```
void ISOUND_RegisterNotify(ISound * pISound, PFNSOUNDSTATUS pfn, const void
* pUser)
```

Parameters:

pISound	Pointer to ISound Interface object
pfn	Status callback function pointer (can be NULL, if no callback is required)
pUser	User data for unique correlation/identification of the transaction. This piece of information is not retrieved or processed by ISound and can be NULL if no identifying data is required. This same data pointer is passed back to the client along with callback status to correlate the transactions.

Return Value:

None

Comments:

The following functions and their callback status types are listed here:

ISOUND_PlayTone	AEE_SOUND_STATUS_CB
ISOUND_PlayToneList	AEE_SOUND_STATUS_CB
ISOUND_PlayFreqTone	AEE_SOUND_STATUS_CB
ISOUND_StopTone	AEE_SOUND_STATUS_CB
ISOUND_Vibrate	AEE_SOUND_STATUS_CB
ISOUND_StopVibrate	AEE_SOUND_STATUS_CB
ISOUND_GetVolume	AEE_SOUND_VOLUME_CB
ISOUND_SetVolume	AEE_SOUND_STATUS_CB

The above list also indicates what types of status are sent out through the callback functions. The status types determine the way the data portion of the status is interpreted. The status indicates the status of the operations or the result of the query in case of **get**-operations. See each of the function descriptions to find out the effects of this registered callback function.

Side Effects:

None

See Also:

None
Return to the List of functions

ISOUND_Release()

Description:

This function decrements the reference count of an object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ISOUNDPLAYER_Release(ISound * pISound)
```

Parameters:

pISound	Pointer to the ISound Interface object

Return Value:

reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

ISOUND_AddRef()
Return to the List of functions

ISOUND_Set()

Description:

This function sets the device attributes of ISound Interface object. The device attributes include Device, Method, APath, Ear-piece Mute Control and Mic Mute Control.

Prototype:

```
int ISOUND_Set(ISound * pISound, const AEESoundInfo * pSoundInfo)
```

Parameters:

pISound	Pointer to ISound Interface object
pSoundInfo	Structure containing ISound device attributes

Return Value:

SUCCESS	ISound device attributes are set
EBADPARAM	One or more AEESoundInfo parameters are wrong

Comments:

The device is not connected until you call ISOUND_SetDevice().

Side Effects:

If **SUCCESS** is NOT returned, then ISound Interface retains the previous values. If it is created first time it has the default values.

See Also:

AEESoundInfo
Return to the List of functions

ISOUND_SetDevice()

Description:

This function commands ISound to connect to a specific audio device. There is always one, and only one, audio device connected at any particular time. The audio device is specified by the call to the ISOUND_Set() function.

Prototype:

```
void ISOUND_SetDevice(ISound * pISound)
```

Parameters:

pISound	Pointer to ISound Interface object

Return Value:

None

Comments:

This function is not supported in BREW Emulator but it is supported on the target device.

Side Effects:

The result of the operation is sent to the client via the callback function the <\bin\DataFiles> subdirectory under the <\BREW> directory(where BREW is installed). The status can be one of the following values:

AEE_SOUND_SUCCESS	Successful connecting to the new sound device
AEE_SOUND_FAILURE	Failed to connect to the new sound device

See Also:

ISOUND_Set()
Return to the List of functions

ISOUND_SetVolume()

Description:

This function sets the volume to be used for the device/method pair specified in the call to ISOUND_Set().

Prototype:

```
void ISOUND_SetVolume(ISound * pISound, uint16 wVolume)
```

Parameters:

pISound	Pointer to ISound Interface object
wVolume	New volume level for device/method

Return Value:

None

Comments:

The range of volume is from 0 (zero, lowest) to AEE_MAX_VOLUME (highest). In the BREW Emulator, the volume increases linearly. This may not be the case on target device.

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The result can be one of the following:

AEE_SOUND_SUCCESS	Successful setting the new volume level
AEE_SOUND_FAILURE	Failed setting the new volume level; the old level still prevails

See Also:

ISOUND_Set()
Return to the List of functions

ISOUND_StopTone()

Description:

This function stops playing current tone or ends the playback of tone list.

Prototype:

```
void ISOUND_StopTone(ISound * pISound)
```

Parameters:

pISound	Pointer to ISound Interface object

Return Value:

None.

Comments:

In BREW Emulator, this function stops the vibration tone, if vibration is active.

Side Effects:

The result of the operation is sent to the client, via the callback function pointer. Also, when the operation is done, it notifies the client through the same callback function pointer. The status can be one of the following values:

AEE_SOUND_SUCCESS	Successful and accepted the stop command. If playback of a tone or tone list is active, then it also generates AEE_SOUND_PLAY_DONE callback
AEE_SOUND_FAILURE	Failed and did not accept the stop command

See Also:

ISOUND_Vibrate()
Return to the List of functions

ISOUND_StopVibrate()

Description:

This stops the current vibration.

Prototype:

```
void ISOUND_StopVibrate(ISound * pISound)
```

Parameters:

pISound	Pointer to ISound Interface object

Return Value:

None

Comments:

In the BREW Emulator, this function is the same as calling ISOUND_StopTone(). As a result, it may stop an ongoing ISOUND_PlayTone() playback.

Side Effects:

This function does not cause any callback.

See Also:

ISOUND_Vibrate()
Return to the List of functions

ISOUND_Vibrate()

Description:

This function causes the device to vibrate for the specified amount or until ISOUND_StopVibrate() is called.

Prototype:

```
void ISOUND_Vibrate(ISound * pISound, uint16 w6Duration)
```

Parameters:

pISound	Pointer to the public ISound object
wDuration	Duration of vibration in milliseconds

Return Value:

None

Comments:

In the BREW Emulator, this function is the same as calling ISOUND_PlayTone() with a sample tone. As a result, it may stop ongoing ISOUND_PlayTone() playback or vice versa.

Side Effects:

This function does not cause any callback.

See Also:

ISOUND_StopVibrate()
Return to the List of functions

ISoundPlayer Interface

This interface provides the sound player services for MIDI and MP3.

To use the services of ISoundPlayer Interface

1 Create the ISoundPlayer object by calling ISHELL_CreateInstance() with AEECLSID_SOUNDPLAYER as ClassID. .

2 Call ISOUND_RegisterNotify() to register the ISoundPlayer callback function. The ISoundPlayer Interface always uses this callback function to notify the applet of any events or status changes. There are five types of callback functions invoked by the ISoundPlayer Interface.

AEE_SOUNDPLAYER_PLAY_CB,

AEE_SOUNDPLAYER_SOUND_CB,

AEE_SOUNDPLAYER_STATUS_CB,

AEE_SOUNDPLAYER_TIME_CB.

AEE_SOUNDPLAYER_VOLUME_CB

Each callback type, in turn, contains a set of status indications that are valid only for callbacks of that type.

For example, for AEE_SOUNDPLAYER_PLAY_CB, the status type can be

AEE_SOUNDPLAYER_SUCCESS,

AEE_SOUNDPLAYER_PAUSE,

AEE_SOUNDPLAYER_RESUME, and so forth

Any Data is returned in a callback through the **dwParam** parameter which points to an AEESoundPlayerCmdData structure, if data is present. Otherwise the **dwParam** is NULL. It is not always required to register a callback function; you can choose not to get any ISoundPlayer events by calling ISOUNDPLAYER_RegisterNotify() as follows:

```
ISOUNDPLAYER_RegisterNotify(pISoundPlayer, NULL, NULL);
```

3 Set the source of the MIDI/MP3 media (file/buffer/MIME) using the ISOUNDPLAYER_Set() function.

4 In the applet event handler, call

ISOUNDPLAYER_Play()	To play the MIDI/MP3 file
ISOUNDPLAYER_Stop()	To stop the playback
ISOUNDPLAYER_Rewind()	To rewind playback for **n** milliseconds
ISOUNDPLAYER_FastForward()	To fast forward playback for **n** milliseconds
ISOUNDPLAYER_Pause()	To pause the playback
ISOUNDPLAYER_Resume()	To resume the playback
. . .	And other ISoundPlayer functions

5 Before exit, always:

a. Un-register the callback function by calling

```
ISOUNDPLAYER_RegisterNotify(pISoundPlayer, NULL, NULL);
```

b. Call ISOUNDPLAYER_Stop()

More on callback:

During playback, every second, the applet is notified of AEE_SOUNDPLAYER_TICK_UPDATE status via AEE_SOUNDPLAYER_PLAY_CB. This can be used by applet to update the progress of playback.

After rewind, fast forward, pause or resume, the **dwParam** parameter of AEE_SOUNDPLAYER_PLAY_CB callback function points to the current position, in milliseconds, of the playback.

Again, applets can use this to update the playback progress. These mechanisms are demonstrated in the sample MIDI-MP3 applet.

ISoundPlayer internally uses an instance of ISound for setting the device and for setting/getting the volume of the output. This makes it easy. You need not create an ISound instance for these operations. These mechanisms are demonstrated in the sample MIDI-MP3 Player applet

List of functions

Functions in this interface include:

ISOUNDPLAYER_AddRef()

ISOUNDPLAYER_FastForward()

ISOUNDPLAYER_GetTotalTime()

ISOUNDPLAYER_GetVolume

ISOUNDPLAYER_Pause()

ISOUNDPLAYER_Play()

ISOUNDPLAYER_RegisterNotify()

ISOUNDPLAYER_Release()

ISOUNDPLAYER_Resume()

ISOUNDPLAYER_Rewind()

ISOUNDPLAYER_Set()

ISOUNDPLAYER_SetSoundDevice()

ISOUNDPLAYER_SetStream()

ISOUNDPLAYER_SetTempo()

ISOUNDPLAYER_SetTune()

ISOUNDPLAYER_SetVolume()

ISOUNDPLAYER_Stop()

Return to the Contents

ISOUNDPLAYER_AddRef()

Description:

This function increments the reference count of the ISoundPlayer Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ISOUNDPLAYER_Release().

Prototype:

```
uint32 ISOUNDPLAYER_AddRef(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to the ISoundPlayer Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

ISOUNDPLAYER_Release()
Return to the List of functions

ISOUNDPLAYER_FastForward()

Description:

This function issues a command to fast-forward the current MIDI/MP3 playback.

Prototype:

```
void ISOUNDPLAYER_FastForward(ISoundPlayer * pISoundPlayer, uint32 dwTime)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
dwTime	Amount of time, in milliseconds, to fast-forward the current playback

Return Value:

None

Comments:

None

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful and accepted the fast-forward command
AEE_SOUNDPLAYER_FAILURE	Failed and did not accept the fast-forward command

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_FFORWARD value for the playback transaction.

See Also:

None
Return to the List of functions

ISOUNDPLAYER_GetTotalTime()

Description:

This function issues a command to request the calculation of the total playback time of the specified MP3/MIDI input source.

Prototype:

```
void ISOUNDPLAYER_GetTotalTime(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object

Return Value:

None

Comments:

Only one play or get-total-time command executes at a time; a play command overrides any previous play or get-total-time command. However, a get-total-time command does not interrupt a play command.

Side Effects:

The result of the operation is sent to the client via the AEE_SOUNDPLAYER_TIME_CB callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful getting the total playback time
AEE_SOUNDPLAYER_FAILURE	Failed getting the total playback time
AEE_SOUNDPLAYER_DONE	Successful getting the total playback time. The **dwParam** parameter to the callback function points to AEESoundPlayerCmdData . The **dwTotalTime** parameter contains the current time in milliseconds.
AEE_SOUNDPLAYER_ABORTED	Command is aborted by the ISOUNDPLAYER_Play() command

See Also:

None
Return to the List of functions

ISOUNDPLAYER_GetVolume

Description:

This function gets the volume for the current playback device and method.

Prototype:

```
void ISOUNDPLAYER_GetVolume(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to the ISoundPlayer Interface object

Return Value:

None.

Comments:

The volume range is from 0 (zero, lowest) to AEE_MAX_VOLUME (highest).

Side Effects:

The result of the operation is sent to the client via the AEE_SOUND_VOLUME_CB callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful getting volume value. The **dwParam** points to AEESoundPlayerCmdData and the **wVolume** contains the current volume.
AEE_SOUNDPLAYER_FAILURE	Failed to get volume value

See Also

AEESoundCmd
ISOUNDPLAYER_SetVolume()
Return to the List of functions

ISOUNDPLAYER_Pause()

Description:

This function issues a command to pause the current MIDI/MP3 playback.

Prototype:

```
void ISOUNDPLAYER_Pause(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to the ISoundPlayer Interface object

Return Value:

None

Comments:

None

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful and accepted the pause command
AEE_SOUNDPLAYER_FAILURE	Failed and did not accept the pause command

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_PAUSE value for the playback transaction.

See Also:

None
Return to the List of functions

ISOUNDPLAYER_Play()

Description:

This function issues a command to play a MIDI/MP3 audio from the input source specified in the call to ISOUNDPLAYER_Set().

Prototype:

```
void ISOUNDPLAYER_Play(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to the ISoundPlayer Interface object

Return Value:

None

Comments:

Only one play or get-total-time command is executed at a time. A play command overrides any previous play or get-total-time command. However, a get-total-time command does not interrupt a play command.

Side Effects:

The registered callback function, with AEE_SOUNDPLAYER_PLAY_CB, is triggered multiple times during the playback to indicate the status of the current playback transaction. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Play command was accepted and begins playback
AEE_SOUNDPLAYER_REWIND	Playback is rewinding
AEE_SOUNDPLAYER_FFORWARD	Playback is fast-forwarding
AEE_SOUNDPLAYER_PAUSE	Playback is paused
AEE_SOUNDPLAYER_RESUME	Playback is resuming after a pause, rewind, fast-forward process
AEE_SOUNDPLAYER_TEMPO	Playback tempo factor has been updated
AEE_SOUNDPLAYER_TUNE	Playback tune factor has been updated

AEE_SOUNDPLAYER_AUDIO_SPEC	Relevant file specs are provided. In BREW Emulator, only file type is returned.
AEE_SOUNDPLAYER_TICK_UPDATE	periodic tick update every second
AEE_SOUNDPLAYER_DATA_ACCESS_DELAY	Data access delay is causing the playback to be temporarily delayed
AEE_SOUNDPLAYER_ABORTED	Playback was aborted
AEE_SOUNDPLAYER_DONE	Playback is finished
AEE_SOUNDPLAYER_FAILURE	Play command was not accepted or there was an error processing the audio input

The following information is passed along in the status event in the **pServerData** parameter.

If MIDI/CMX	Pointer to AEESoundPlayerMIDISpec structure
If MP3	Pointer to AEESoundPlayerMP3Spec structure

See Also:

ISOUNDPLAYER_Set()
Return to the List of functions

ISOUNDPLAYER_RegisterNotify()

Description:

This function registers the status event callback function.

Prototype:

```
void ISOUNDPLAYER_RegisterNotifyCB
        (
        ISoundPlayer * pISoundPlayer,
        PFNSOUNDPLAYERSTATUS pfn,
        void * pUser
        )
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
pfn	Status callback function pointer which can be NULL
pUser	User data for unique correlation/identification of the transaction. This piece of information is not retrieved or processed by ISound and can be NULL if no identifying data is required. This same data pointer is passed back to the client along with status callback to correlate the transactions.

Return Value:

None

Comments:

The following functions and their callback status types are listed here:

ISOUNDPLAYER_Play	AEE_SOUNDPLAYER_PLAY_CB
ISOUNDPLAYER_Stop	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_Rewind	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_FastForward	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_Pause	AEE_SOUNDPLAYER_STATUS_CB

ISOUNDPLAYER_Resume	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_SetTempo	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_SetTune	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_SetVolume	AEE_SOUNDPLAYER_STATUS_CB
ISOUNDPLAYER_GetVolume	AEE_SOUNDPLAYER_VOLUME_CB
ISOUNDPLAYER_GetTotalTime	AEE_SOUNDPLAYER_TIME_CB
ISOUNDPLAYER_SetSoundDevice	AEE_SOUNDPLAYER_STATUS_CB

The above list also indicates what types of status are sent back through the callback functions. The status type determines the way the data portion of the status is interpreted. The status indicates the status of the operations or the result of the query in case of get-operations. See each of the function descriptions to find out the effects of this registered callback function.

Side Effects:

None

See Also:

None
Return to the List of functions

ISOUNDPLAYER_Release()

Description:

This function decrements the reference count of an object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ISOUNDPLAYER_Release(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer Pointer to the ISoundPlayer Interface object

Return Value:

Decremented reference count for the object. The object has been freed and is no longer valid if 0 (zero) is returned.

Comments:

None

Side Effects:

None

See Also:

ISOUNDPLAYER_AddRef()
Return to the List of functions

ISOUNDPLAYER_Resume()

Description:

This function issues a command to resume the current MIDI/MP3 playback.

Prototype:

```
void ISOUNDPLAYER_Resume(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object

Return Value:

None

Comments:

Playback must have been paused with the ISOUNDPLAYER_Pause() function.

If playback is not currently paused by the ISOUNDPLAYER_Pause() function, this function call has no effect.

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful and accepted the resume command
AEE_SOUNDPLAYER_FAILURE	Failed and did not accept the resume command

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_RESUME value for the playback transaction.

See Also:

ISOUNDPLAYER_Pause()
Return to the List of functions

ISOUNDPLAYER_Rewind()

Description:

This function issues a command to rewind the current MIDI/MP3 playback.

Prototype:

```
void ISOUNDPLAYER_Rewind(ISoundPlayer * pISoundPlayer, uint32 dwTime)
```

Parameters:

pISoundPlayer	Pointer to an ISoundPlayer Interface object
dwTime	Amount of time, in milliseconds, to rewind the current playback

Return Value:

None

Comments:

None

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful and accepted the rewind command
AEE_SOUNDPLAYER_FAILURE	Failed and did not accept the rewind command

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_REWIND value for the playback transaction.

See Also:

None
Return to the List of functions

ISOUNDPLAYER_Set()

Description:

This function sets the source of the MIDI/MP3 audio.

Prototype:

```
void ISOUNDPLAYER_Set(ISoundPlayer * pISoundPlayer, AEESoundPlayerInput t,
void * pData)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
pData	Pointer to data of specified type
t	Source type of pData. Supported values include: SDT_FILE -Specified data is file name SDT_BUFFER -Specified data is raw buffer

Return Value:

None

Comments:

Only SDT_FILE is supported in the BREW Emulator.

Side Effects:

None

See Also:

None
Return to the List of functions

ISOUNDPLAYER_SetSoundDevice()

Description:

This function commands the device to connect to a specific audio device; there is always one, and only one, audio device connected at any particular time.

Prototype:

```
void ISOUNDPLAYER_SetSoundDevice(ISoundPlayer * pISoundPlayer,
AEESoundDevice eDevice, AEESoundMuteCtl eEarMute, AEESoundMuteCtl eMicMute )
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
eDevice	Audio device to be connected
eEarMute	Mute or unmute the ear piece
eMicMute	Mute or unmute the microphone

Return Value:

None

Comments:

This function is not supported in the BREW Emulator, but it is supported on the target device.

Side Effects:

The result of the operation is sent to the client via the AEE_SOUNDPLAYER_SOUND_CB callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful connecting to the new sound device
AEE_SOUNDPLAYER_FAILURE	Failed connecting to the new sound device

See Also:

None,
Return to the List of functions

ISOUNDPLAYER_SetStream()

Description:

This function allows an IAStream interface to be associated with an ISoundPlayer instance to allow audio input to be streamed from a file or socket.

Prototype:

```
void ISOUNDPLAYER_SetStream (ISoundPlayer * pISoundPlayer, IAStream * ps)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object that will receive the streaming audio input
ps	Pointer to an instance of a class that implements the IAStream interface (e.g., IFile or ISocket)

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

IASTREAM_Read()
IASTREAM_Readable()
Return to the List of functions

ISOUNDPLAYER_SetTempo()

Description:

This function issues a command to modify the playback tempo of the current MIDI/MP3 playback as a percentage of the normal playback tempo.

Prototype:

```
void ISOUNDPLAYER_SetTempo(ISoundPlayer * pISoundPlayer, uint32
dwTempoFactor)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
dwTempoFactor	This parameter is used to modify the playback tempo as a percentage of the normal playback tempo

Return Value:

None

Comments:

For example, a dwTempoFactor of 50 means half of the normal playback tempo.

The maximum supported tempo factor is 500%, or five times of the normal playback tempo.

The minimum tempo factor is 1% of the normal playback tempo. The tempo factor is initially 100% when the ISOUNDPLAYER_Play() function is called. This function is not supported in the BREW Emulator, but it is supported on the target device.

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful setting a new tempo value
AEE_SOUNDPLAYER_FAILURE	Failed setting a new tempo value; old value remains

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_TEMPO value for the playback transaction.

See Also:

ISOUNDPLAYER_Play()
Return to the List of functions

ISOUNDPLAYER_SetTune()

Description:

It issues a command to modify the tune (pitch level) of the current MIDI/MP3 playback in half-step increments.

Prototype:

```
void ISOUNDPLAYER_SetTune(ISoundPlayer * pISoundPlayer, uint8 nStep)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object
nStep	This parameter is used to modify the tune of the playback in half-step increments. For example, a tune factor 0 (zero) indicates normal playback. A tune factor 1 indicates 1 half-step higher than the normal playback. Only values from -12 to 12 are supported. Namely the tune can be 1 octave higher than the normal playback at maximum, and 1 octave lower than the normal playback at minimum. This parameter is used to modify the playback tune as a percentage of the normal playback tune.

Return Value:

None

Comments:

This function is not supported in the BREW Emulator, but it is supported on the target device.

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful setting a new tune value
AEE_SOUNDPLAYER_FAILURE	Failed setting a new tune value; old value remains

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_TUNE value for the playback transaction.

See Also:

ISOUNDPLAYER_Play()
Return to the List of functions

ISOUNDPLAYER_SetVolume()

Description:

This function sets the volume for the current playback device.

Prototype:

```
void ISOUNDPLAYER_SetVolume(ISoundPlayer * pISoundPlayer, uint16 wVolume)
```

Parameters:

pISoundPlayer	Pointer to the ISoundPlayer Interface object
wVolume	New volume level for device/method

Return Value:

None

Comments:

The volume range is from 0 (zero, lowest) to AEE_MAX_VOLUME (highest).

Side Effects:

The result of the operation is sent to the client via the AEE_SOUNDPLAYER_SOUND_CB callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful setting a new volume value
AEE_SOUNDPLAYER_FAILURE	Failed to set a new volume value; old value remains

See Also:

ISOUNDPLAYER_GetVolume
Return to the List of functions

ISOUNDPLAYER_Stop()

Description:

This function issues a command to stop the current MIDI/MP3 playback.

Prototype:

```
void ISOUNDPLAYER_Stop(ISoundPlayer * pISoundPlayer)
```

Parameters:

pISoundPlayer	Pointer to ISoundPlayer Interface object

Return Value:

None

Comments:

None

Side Effects:

The result of the operation is sent to the client via the callback function pointer. The status can be one of the following values:

AEE_SOUNDPLAYER_SUCCESS	Successful and accepted the stop command
AEE_SOUNDPLAYER_FAILURE	Failed and did not accept the stop command

If this operation is successful, the function also triggers another event to be sent to the client with the AEE_SOUNDPLAYER_DONE value for the playback transaction.

See Also:

None
Return to the List of functions

IStatic Interface

The IStatic Interface allows you to create a static text control, which consists of a read-only text message and a title that appears at the top of the control's rectangle. Unlike the text controls created with the ITimeCtl Interface, the text in a static text control cannot be entered or modified by the device user. The text message is broken into lines within the control rectangle, with each line containing an integral number of words (that is, words are not broken across lines). If the text message is too long to completely fit on the device screen, the static text control automatically scrolls the message. When the end of the message is reached, the continuous scrolling starts again from the beginning. Instead of using a text message as the control's contents, you can choose to display an animated bitmap created with the IImage Interface. If this option is selected, the bitmap is displayed centered inside the static text control's rectangle.

The IStatic Interface provides a number of properties that can be used to customize the appearance and behavior of the static text control. These properties, which are all initially unset and can be set with the ISTATIC_SetProperties() function, are as follows:

- ST_ICONTEXT indicates that an animated bitmap is used instead of a text message. When this property is set, ISTATIC_SetText() expects a pointer to an IImage Interface instance instead of a pointer to the message text string (see later in this section).
- ST_TEXTALLOC tells ISTATIC_SetText() not to allocate storage for the message text string (you set this property if you have already allocated storage for the message string in your code).
- ST_TITLEALLOC tells ISTATIC_SetText() not to allocate storage for the control title string (you set this property if you have already allocated storage for the title in your code).
- ST_NOSCROLL disables the automatic scrolling of long text messages (in later releases, setting this property will allow the device user to manually scroll the text with the UP and DOWN keys).
- ST_MIDDLETEXT specifies that the message text to be vertically centered within the control's rectangle (by default, the text begins at the top of the rectangle, immediately below the control title).
- ST_UNDERLINE causes the control title to be underlined.
- ST_CENTERTEXT centers each line of the message text (by default, the lines are left-justified within the control rectangle).

- ST_CENTERTITLE centers the control title (by default, the title is left-justified).

The following steps are typically followed to use a static text control (refer to the function descriptions in the following subsections and the IStatic Interface usage examples for details on the parameter lists and return values of each IStatic Interface function):

To use a static text control

1. Create an instance of an IStatic Interface using the ISHELL_CreateInstance().
2. Set the dimensions of the control's rectangle using the ISTATIC_SetRect().
3. Obtain the text for the control's title and text message strings, either by reading them from a BREW resource file or from text strings in your code. If a bitmap is being used in place of the text message, read in the bitmap and use IIMAGE_SetParm() function to set its animation properties (number of frames, animation rate, and other items).
4. Call ISTATIC_SetText() to specify the control's title string, message string or bitmap, and the fonts to be used for the title and text. You must call ISTATIC_SetRect() (see step 2) before calling ISTATIC_SetText(), since the latter uses information about the control rectangle to determine how to display the control.
5. Call ISTATIC_Redraw() to display the control's contents on the screen.
6. When the control is no longer needed, call ISTATIC_Release(). This function frees the memory for the control's message text and title, but it does NOT release an animated bitmap, so you must explicitly release the IImage Interface if you created an instance of one in step 3.

List of functions

Functions in this interface include:

ISTATIC_AddRef()

ISTATIC_GetProperties()

ISTATIC_GetRect()

ISTATIC_HandleEvent()

ISTATIC_Redraw()

ISTATIC_Release()

ISTATIC_Reset()

ISTATIC_SetProperties()

ISTATIC_SetRect()

ISTATIC_SetText()

Return to the Contents

ISTATIC_AddRef()

Description:

This function increments the reference count of the IStatic Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 ISTATIC_AddRef(IStatic * pIStatic)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

ISTATIC_Release()
Return to the List of functions

ISTATIC_GetProperties()

Description:

This function retrieves the properties of the IStatic Interface object. The properties determine the physical look and feel of the object.

Prototype:

```
uint32 ISTATIC_GetProperties(IStatic * pIStatic)
```

Parameters

pIStatic	Pointer to the IStatic Interface object

Return Value:

ST_CENTERTEXT	Center text
ST_CENTERTITLE	Center title
ST_NOSCROLL	Do not scroll text
ST_TEXTALLOC	Text allocated - dialog now owns it
ST_TITLEALLOC	Title allocated - dialog now owns it
ST_MIDDLETEXT	Text is drawn in the middle of the screen
ST_UNDERLINE	Underline the title
ST_ICONTEXT	Text is IImage

Comments:

None

Side Effects:

None

See Also:

ISTATIC_SetProperties()
Return to the List of functions

ISTATIC_GetRect()

Description:

This function retrieves the location and size of the IStatic Interface object.

Prototype:

```
void ISTATIC_GetRect(IStatic * pIStatic, const AEERect * prc)
```

Parameters

pIStatic	Pointer to the IStatic Interface object
prc	Pointer to the IStatic object rectangle (location and size)

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISTATIC_SetRect()
AEERect
Return to the List of functions

ISTATIC_HandleEvent()

Description:

This function handles key events for the active IStatic Interface object.

Prototype:

```
boolean ISTATIC_HandleEvent(IStatic * pIStatic, AEEEvent eCode, uint16
wParam, uint32 dwParam)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object
eCode	Event code
wParam	16-bit first event parameter
dwParam	32-bit second event parameter

Return Value:

TRUE	Event has been processed
FALSE	Event has not been processed

Comments:

None

Side Effects:

None

See Also:

AEE Events
Return to the List of functions

ISTATIC_Redraw()

Description:

This function refreshes the IStatic object on the screen

Prototype:

```
boolean ISTATIC_Redraw(IStatic * pIStatic)
```

Parameters:

pIStatic	Pointer to the IStatic InterfaceIStatic Interface object

Return Value:

TRUE	Always

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISTATIC_Release()

Description:

This function decrements the reference count of the IStatic Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ISTATIC_Release(IStatic * pIStatic)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object

Return Value:

Decremented reference count for the object.

Comments:

The object has been freed and is no longer valid if 0 is returned.

Side Effects:

None

See Also:

ISTATIC_AddRef()
Return to the List of functions

ISTATIC_Reset()

Description:

This function resets an IStatic Interface object. It cleans up the title and text of the IStatic Interface object.

> **NOTE:** Any timer(s) scheduled with AEE using this object as **pUser** parameter of ISHELL_SetTimer() are also cancelled.

Prototype:

```
void ISTATIC_Reset(IStatic * pIStatic)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ISTATIC_SetProperties()

Description:

This function set the properties of the IStatic Interface object. The properties value determines the physical look and feel of the object.

Prototype:

```
void ISTATIC_SetProperties(IStatic * pIStatic, uint32 nProperties)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object
nProperties	Properties value
ST_CENTERTEXT	Center text
ST_CENTERTITLE	Center title
ST_NOSCROLL	Do not scroll text
ST_TEXTALLOC	Text allocated - dialog now owns it
ST_TITLEALLOC	Title allocated - dialog now owns it
ST_MIDDLETEXT	Text is drawn in the middle of the screen
ST_UNDERLINE	Underline the title
ST_ICONTEXT	Text is IImage

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISTATIC_GetProperties()
Return to the List of functions

ISTATIC_SetRect()

Description:

This function set the location and size of the IStatic Interface object.

Prototype:

```
void ISTATIC_SetRect(IStatic * pIStatic, const AEERect * prc)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object
prc	Pointer to the IStatic Interface object rectangle (location and size)

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISTATIC_GetRect()
AEERect
Return to the List of functions

ISTATIC_SetText()

Description:

This function sets the text and title of the IStatic Interface object.

Prototype:

```
boolean ISTATIC_SetText(IStatic * pIStatic, AECHAR * pTitle, AECHAR * pText,
AEEFont fntTitle, AEEFont fntText)
```

Parameters:

pIStatic	Pointer to the IStatic Interface object
pTitle	Pointer to the title string
pText	Pointer to the text string
fntTitle	Title font type
fntText	Text font type

Return Value:

TRUE	If text and title were set correctly
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

AEEFont
Return to the List of functions

ITAPI Interface

ITAPI is a simple interface to the telephony layer in the device. This function provides the following services:

- Retrieving status
- Placing voice calls
- Extracting SMS text from SMS messages
- Obtaining caller ID on incoming or in-progress calls

The ITAPI interface is obtained through the ISHELL_CreateInstance() function.

List of functions

Functions in this interface include:

ITAPI_AddRef()

ITAPI_ExtractSMSText()

ITAPI_GetCallerID()

ITAPI_GetStatus()

ITAPI_MakeVoiceCall()

ITAPI_Release()

Return to the Contents

ITAPI_AddRef()

Description:

This function increments the reference count of ITAPI Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero). See ITAPI Interface.

Prototype:

```
uint32 ITAPI_AddRef(ITapi * pITapi)
```

Parameters:

pITapi	Pointer to the ITAPI Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count

Side Effects:

None

See Also:

ITAPI_Release()
Return to the List of functions

ITAPI_ExtractSMSText()

Description:

This function extracts the formatted text from a raw SMS message. The format of the input parameter is AEESMSMsg. The buffer returned is valid until the next call to ITAPI_ExtractSMSText or until the interface is released.

Prototype:

```
// C Users
 AEESMSTextMsg * ITAPI_ExtractSMSText(ITAPI * po,const AEESMSMsg * pMsg)
```

Parameters:

po	Pointer to the ITAPI Interface object
pMsg	Pointer to input AEESMSMsg

Return Value:

NULL	If this function fails

Comments:

This function is not supported on the device emulator.

Side Effects:

None

See Also:

Return to the List of functions

ITAPI_GetCallerID()

Description:

This function retrieves the ID, in digits, of an incoming or outgoing voice call.

Prototype:

```
// C Users
  boolean ITAPI_GetCallerID(ITAPI * po,AECHAR * pDest, int nSize)
```

Parameters:

po	Pointer to the ITAPI Interface object
pDest	Destination pointer
nSize	Size in bytes of the destination buffer

Return Value:

TRUE	Call in progress and buffer filled
FALSE	No call in progress or invalid buffer

Comments:

None

Side Effects:

None

See Also:

Return to the List of functions

ITAPI_GetStatus()

Description:

This function obtains the current status of the telephony device, including service and call status. Applications can also register to receive updated TAPIStatus information on any changes through the ISHELL_RegisterNotify() function.

Prototype:

```
// C Users
  int ITAPI_GetStatus(ITPAI * po, TAPIStatus * ps)
```

Parameters:

po	Pointer to the ITAPI Interface object
ps	Pointer to the status information to be filled

Return Value:

SUCCESS	Valid status information
EBADPARM	Bad parameter

Comments:

None

Side Effects:

None

See Also:

ISHELL_RegisterNotify()

Return to the List of functions

ITAPI_MakeVoiceCall()

Description:

This function is called to place a voice call. The number dialed is specified in the digits string. Note the following when using this function:

- No call is placed if the input string is empty or NULL. Only specific digits are allowed, 0 (zero) through 9, #, and *. All other digits are ignored.
- If a voice call is in progress, EALREADY is returned.
- If a data call is in progress and no network activity is in progress (for example, TCP), the data call is ended, and the call is placed.
- This function enforces the privacy policies established by the carrier. These may include intermediate prompts to you.

Prototype:

```
//C Users
 int ITAPI_MakeVoiceCall(ITAPI * po, const char * pszNumber,AEECLSID
 clsReturn)
```

Parameters:

po	Pointer to the ITAPI Interface object
pszNumber	Pointer to number to dial
clsReturn	The application to be started when the call ends

Return Value:

SUCCESS	Function in progress
EBADPARM	Invalid number
EALREADY	Voice call already in progress

Comments:

None

Side Effects:

None

See Also:

Return to the List of functions

ITAPI_Release()

Description:

This function decrements the reference count of ITAPI Interface object. The object is freed from memory and is no longer valid once the reference count reaches 0 (zero).

Prototype:

```
uint32 ITAPI_Release(ITapi * pITapi)
```

Parameters:

pITapi	Pointer to the ITAPI Interface object

Return Value:

Reference count	Decremented reference count for the object
0 (zero)	If the object has been freed and is no longer valid

Comments:

None

Side Effects:

None

See Also:

ITAPI_AddRef()
Return to the List of functions

ITextCtl Interface

A text control enables the device user to enter a string of text using the keys on the device. The text control consists of an optional title and a rectangular window containing one or more lines in which the entered text is displayed to the user. The text control handles the translation of user key presses into characters, so your application only needs to pass keypress events to the text control while it is active and retrieve the text from the control when user text entry has completed. The translation process depends on the text entry modes the device supports (for example, the standard multi-tap mode in which the user selects from the characters mapped to each key, and Tegic's T9 predictive text input mode). If more than one text entry mode is supported, your application can make it possible for the user to select the desired mode while the text control is active. The text control allows you to specify a softkey menu that is used for this purpose. While the text control is active, your application must send all keypress events to it by calling ITEXTCTL_HandleEvent().

Text controls support the following properties, which can be set with ITEXTCTL_SetProperties() (the property names are the names of the bit-mask constants used to set and test the property values):

TP_MULTILINE allows multiple lines of text to appear in the text entry window (by default, only a single line appears).

TP_FRAME draws a frame around the text control.

TP_T9_MODE specifies that T9 is the default text entry mode for the device (if this property is not set, multi-tap is the default mode). This mode is not supported on the BREW Emulator.

Text controls provide several functions in addition to those in the IControl Interface. ITEXTCTL_SetTitle() and ITEXTCTL_SetText() specify values for the control's title and for the text string that appears in the text entry window (the latter function can be used to provide an initial value for the window's contents that the user can edit). ITEXTCTL_GetText() retrieves the current value of the control's text string and copies it into a buffer. ITEXTCTL_GetTextPtr() is similar, except that it returns a pointer to the character string in the text control that is used to store the text, without making a copy of it. ITEXTCTL_SetMaxSize() determines the maximum number of characters that can be entered into the text control. ITEXTCTL_EnableCommand() enables the sending of an **EVT_COMMAND** to your application when the user presses the SELECT key (this function is not supported at present).

ITEXTCTL_SetSoftKeyMenu() associates a SoftKey menu control with the text control. This is typically a SoftKey menu that you have created and that appears on the screen while the text control is displayed. ITEXTCTL_SetSoftKeyMenu() adds an item to the SoftKey menu that allows the user to change the text entry mode (the text string for this item indicates the currently selected mode). When it receives this command, the text control displays a menu allowing the user to select the new text entry mode. After the user selects the new mode, the text control is reactivated and the user continues

entering text. While entering text, the user can press the SELECT key to leave text-edit mode and activate the SoftKey menu. While the SoftKey menu is active, the user can press the UP key to return to edit mode without making a menu selection.

To use a text control in your application

1 Call ISHELL_CreateInstance() to create an instance of the text control.

2 Call ITEXTCTL_SetRect() to specify the screen rectangle that will contain the text control.

3 If desired, call ITEXTCTL_SetTitle() and/or ITEXTCTL_SetText() to specify the control's title and the initial value of its text string.

4 Call ITEXTCTL_SetProperties() to set any text control properties.

5 Call Call ITEXTCTL_SetSoftKeyMenu() to specify the SoftKey menu that is associated with the text control, if any.

6 Call ITEXTCTL_SetActive() to activate the text control and draw its contents on the screen.

7 While the text control is active, call ITEXTCTL_HandleEvent() to pass it any key events generated by the user.

8 When the user has completed entering text, call ITEXTCTL_GetText() or ITEXTCTL_GetTextPtr() to retrieve the text the user has entered. (If you are using a SoftKey menu, the user may signal the completion of text entry with a "Done" item in the menu, or by pressing the SELECT or other key if no SoftKey menu is present).

9 Call ITEXTCTL_Release() to free the text control when you no longer need it.

List of functions

Functions in this interface include:

ITEXTCTL_AddRef()

ITEXTCTL_EnableCommand()

ITEXTCTL_GetProperties()

ITEXTCTL_GetRect()

ITEXTCTL_GetText()

ITEXTCTL_GetTextPtr()

ITEXTCTL_HandleEvent()

ITEXTCTL_IsActive()

ITEXTCTL_Redraw()

ITEXTCTL_Release()

ITEXTCTL_Reset()

ITEXTCTL_SetActive()

ITEXTCTL_SetInputMode()

ITEXTCTL_SetMaxSize()

ITEXTCTL_SetProperties()

ITEXTCTL_SetRect()

ITEXTCTL_SetSoftKeyMenu()

ITEXTCTL_SetText()

ITEXTCTL_SetTitle()

Return to the Contents

ITEXTCTL_AddRef()

Description:

This function increments the reference count for the text control object

Prototype:

uint32 ITEXTCTL_AddRef(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

Incremented reference count for the object

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_Release()
Return to the List of functions

ITEXTCTL_EnableCommand()

Description:

This function is used to enable sending of specified command by the text control object to the shell object upon receiving the event generated by pressing the SELECT key.

Prototype:

void ITEXTCTL_EnableCommand(ITextCtl * pITextCtl, boolean bEnable, uint16 nCmdId)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
bEnable	Boolean value for enable/disable flag
nCmdId	Command id

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_GetProperties()

Description:

This function returns the text control-specific properties or flags.

Prototype:

uint32 ITEXTCTL_GetProperties(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

32-bit properties for the text control

Following properties are returned by the text control object

TP_MULTILINE	If set, text control object is multiple line control
TP_FRAME	If set, text control object has a frame
TP_T9_MODE	If set, text control object is in T9 mode

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_SetProperties()
Return to the List of functions

ITEXTCTL_GetRect()

Description:

This function fills given pointer to AEERect with the coordinates of the current bounding rectangle of the text control object. This is particularly useful after a control is created to determine its optimal/default size and position.

Prototype:

void ITEXTCTL_GetRect(ITextCtl * pITextCtl, AEERect * prc)

Parameters:

pITextCtl te	Pointer to the ITextCtl Interface object
prc	Rectangle to be filled with the coordinates of the text control object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_SetRect()
Return to the List of functions

ITEXTCTL_GetText()

Description:

This function is used to read text associated with the ITextCtl Interface object in the given buffer subject to the maximum of **nMaxChars**.

Prototype:

boolean ITEXTCTL_GetText(ITextCtl * pITextCtl, TCHAR * pBuffer, unsigned int nMaxChars)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
pBuffer	Placeholder for the text
nMaxChars	Maximum number of characters to be read

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_GetTextPtr()
Return to the List of functions

ITEXTCTL_GetTextPtr()

Description:

It returns the pointer to the text maintained by the ITextCtl object. The difference between this function and GetText is that latter copies the content to a destination buffer, and the former just returns the pointer to the text inside the ITextCtl object.

Prototype:

TCHAR * ITEXTCTL_GetTextPtr(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

pointer to the text buffer of the test control object

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_GetText()
Return to the List of functions

ITEXTCTL_HandleEvent()

Description:

This function is used to handle the events received by text control object. If the text control object is in non edit mode, it processed only set title, set text and press of UP and DOWN key events. In text edit mode, it processes various events like key up, key down, key held, set title, set text, command event from soft key menu.

Prototype:

boolean ITEXTCTL_HandleEvent(ITextCtl * pITextCtl, AEEEvent evt, uint16 wp, uint32 dwp)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
evt	Event code
wp	16-bit event data
dwp	32-bit event data

Return Value:

TRUE	If the event was processed by the text control
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_IsActive()

Description:

This function returns the active state of the text control object.

Prototype:

boolean ITEXTCTL_IsActive(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

TRUE	If the text control is active
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_Redraw()

Description:

This function instructs the text control object to redraw its contents. The ITextCtl Interface object does not redraw its contents every time the underlying data behind the text control changes. This allows several data updates to occur while minimizing screen flashes. For example, several changes can be made to the contents of the text control object with no visible effect until ITEXTCTL_Redraw() function is called.

Prototype:

boolean ITEXTCTL_Redraw(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

TRUE	If the text control was redrawn
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_Release()

Description:

This function decrements the reference count for the ITextCtl Interface object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

uint32 ITEXTCTL_Release(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object whose reference count needs to be decremented

Return Value:

Updated reference count for the object.

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_AddRef()
Return to the List of functions

ITEXTCTL_Reset()

Description:

This function instructs the text control to reset (free/delete) its contents as well as to immediately leave active/focus mode.

Prototype:

void ITEXTCTL_Reset(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ITEXTCTL_SetActive()
Return to the List of functions

ITEXTCTL_SetActive()

Description:

This function is used to make a text control object active. Only an active text control object handles the event sent to it. Inactive text control object just ignores the events. Also an inactive text control object does not draw its frame.

Prototype:

void ITEXTCTL_SetActive(ITextCtl * pITextCtl,boolean bActive)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
bActive	Boolean flag that specifies: TRUE: to activate the text control object FALSE: to deactivate the text control object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_SetInputMode()

Description:

This function allows the caller to set the selected text input mode.

Prototype:

AEETextInputMode ITEXTCTL_SetInputMode(ITextCtl * pITextCtl, AEETextInputMode wMode)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
wMode	Text input mode

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

AEETextInputMode
Return to the List of functions

ITEXTCTL_SetMaxSize()

Description:

This function is used to set the maximum text size supported by the text control object. If the size being set is more than the size already set, this leads to the freeing up of the memory associated with the previous size and allocation of the memory per the new size.

Prototype:

void ITEXTCTL_SetMaxSize (ITextCtl * pITextCtl, uint16 nMaxSize)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
nMaxSize	Maximum text size in AECHAR characters excluding NULL and if 0 (zero) then no effect

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_SetProperties()

Description:

This function sets text control-specific properties or flags.

Prototype:

void ITEXTCTL_SetProperties(ITextCtl * pITextCtl, uint32 dwProps)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
dwProps	32-bit set of flags/properties

Following properties are used for text control object:

TP_MULTILINE	If set, text control object is multiple line control.
TP_FRAME	If set, text control object has a frame.
TP_T9_MODE	If set, text control object is in T9 mode.

Return Value:

None

Comments:

None

Side Effects:

It deactivates the text control.

See Also:

ITEXTCTL_GetProperties()
Return to the List of functions

ITEXTCTL_SetRect()

Description:

This function can be used to set the coordinates specified by prc as control rectangle of the text control object. A call to this function, also leads to calculate control rectangle for text line.

Prototype:

void ITEXTCTL_SetRect(ITextCtl * pITextCtl, const AEERect * prc)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
prc	Bounding rectangle for the text control object

Return Value:

None

Comments:

By default, the control rectangle of the text control object has device screen width as width and (device screen height - text height) as height starting from upper left corner.

Side Effects:

None

See Also:

ITEXTCTL_GetRect()
Return to the List of functions

ITEXTCTL_SetSoftKeyMenu()

Description:

It replaces the existing SoftKey menu of the text control object with the specified menu control object.

Prototype:

void ITEXTCTL_SetSoftKeyMenu(ITextCtl * pITextCtl, IMenuCtl * pm)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
pm	New menu control object for the soft key menu

Return Value:

None

Comments:

None

Side Effects:

IMenuCtl's reference count is bumped up and a new menu item is added to the menu if an entry mode string is maintained by the text manager.

See Also:

None
Return to the List of functions

ITEXTCTL_SetText()

Description:

This function is used to assign given string as text of the text control object.

Prototype:

boolean ITEXTCTL_SetText(ITextCtl * pITextCtl, const TCHAR * psz, int cch)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
psz	Text string to be set
cch	Number of AECHAR characters to be assigned from the string to the text of the text control object. If cch is negative or greater than the length of psz string, then the length of string is used.

Return Value

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITEXTCTL_SetTitle()

Description:

This function is used to set title of a text control object. If pText is not NULL, it sets the string specified by pText as the title of the text control object. If pText is NULL, it reads title string corresponding to the given resource identifier from resource file and sets it as the title of the text control object.

Prototype:

boolean ITEXTCTL_SetTitle(ITextCtl * pITextCtl, const char * pszResFile, uint16 wResID, TCHAR * pText)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
pszResFile	File containing resource string
wResID	Resource identifier
pText	Null terminated title string

Return Value:

TRUE	If successful
FALSE	If otherwise

Comments:

None

Side Effects:

If **pText** is NULL and **pszResFile**, **WResID** are valid, this function assigns the text control object title string to **pText**.

See Also:

None
Return to the List of functions

ITimeCtl Interface

Time controls allow the device user to enter a time value in several different formats. They can also be used to display a time value to the user. When entering a time value, the user presses the LEFT and RIGHT keys to select the time field that is to be modified (that is, hours, minutes or seconds), and presses the UP and DOWN keys to increase or decrease the value of the edited field. When the user presses the SELECT key and command sending is enabled (see later in this section), an EVT_COMMAND event is sent to the application or dialog that created the time control, which signals to your application that the user has completed entry of a time value. You can also specify the time value that is displayed in the time control. This feature can be used to repeatedly update the control to reflect a changing time value. For example, to implement a stopwatch, the time control does not measure time itself by setting timers or accessing the current time of day, (so you must obtain the necessary time values in your code using functions like GET_TIMEMS()).

There are three types of time controls, each of which displays the time in a different format (you select the type you want by specifying its ClassID when you create an instance of the time control):

- A clock control (ClassID AEECLSID_CLOCKCTL) displays the number of hours and minutes along with an AM/PM indicator. This type of control can be used to specify the time at which an alarm clock is to go off.
- A countdown control (ClassID AEECLSID_COUNTDOWNCTL) displays the number of hours, minutes and seconds. This display format is useful for a countdown timer that allows the user to set the timer duration and then displays the time remaining until the timer expires.
- A stopwatch control (ClassID AEECLSID_STOPWATCHCTL) displays the number of hours, minutes, seconds and hundredths of seconds. It can be used to implement a stopwatch.

A time control sends a control tabbing event (EVT_CTL_TAB) when the user presses the LEFT and RIGHT keys while editing the LEFT-most and RIGHT-most fields in the time control, respectively. You can use control tabbing to move between controls in a multicontrol screen (if your time control is part of a dialog, the dialog intercepts the control tabbing events and changes control focus appropriately).

Time controls support a number of properties that can be set with ITIMECTL_SetProperties() (the property names are the names of the bitmask constants you use to test and set the properties):

TP_AUTOREDRAW causes the time control to be redrawn whenever it is set active.

TP_NO_SECONDS prevents the display and editing of the number of seconds in a countdown control.

TP_NO_MSECONDS suppresses the display of the number of hundredths of seconds in a stopwatch control.

TP_NOEDIT_AMPM prevents the user from editing the AM/PM field in a clock control.

Time controls implement several functions in addition to those in the IControl Interface. ITIMECTL_SetTime() sets the value of the time (in number of milliseconds) stored in the time control and displays its value in the time control's format. ITIMECTL_SetTimeEx() is an extended version of this function that updates only the displayed value of the time but not the internally stored value. ITIMECTL_GetTime() retrieves the current time in milliseconds that is stored in the time control. ITIMECTL_GetTimeString() takes a time value in milliseconds as input and converts it into a text string that contains the corresponding number of hours and minutes, and can optionally include the number of seconds and hundredths of seconds and an AM/PM indicator. The numerical portions of the time string are separated by colons.

ITIMECTL_EnableCommand() is used to enable or disable the sending of EVT_COMMAND events to your application when the user presses the SELECT key (command sending is disabled by default). ITIMECTL_SetIncrement() sets the number of minutes by which the minutes field of the text control is incremented or decremented when the user presses the UP or DOWN arrow keys while editing the field (the default value is one minute). ITIMECTL_SetEditField() is used to specify which field of the time control is being edited by the user. Possible values are the hours, minutes, or seconds fields (if the seconds field is specified for a clock control, the AM/PM field is selected, assuming that it is editable; the hundredth-seconds field of a stopwatch control is not editable). The user selects the field to be edited with the LEFT and RIGHT arrow keys, so ITIMECTL_SetEditField() can be used to select the initial field to be edited.

To create and use a time control

1. Call ISHELL_CreateInstance() to create the time control instance and obtain an interface pointer to it, specifying which of the three types of time control you would like by its ClassID.

2. Call ITIMECTL_SetTime() to specify an initial time value for the time control if one is desired.

3. Call ITIMECTL_SetRect() to define the screen rectangle in which the time control will be drawn.

4 Call ITIMECTL_SetProperties() if needed to set any of the time control properties, and call ITIMECTL_SetIncrement() and/or ITIMECTL_SetEditField() if you would like to change the default minutes increment or initial edit field.

5 When you have completely specified the contents and properties of the time control, call ITIMECTL_SetActive() to draw the control on the screen and (if your control supports editing by the user) enable it to receive key events from the user to select a time value. While the time control is active, your application's IAPPLET_HandleEvent() function must call ITIMECTL_HandleEvent() to pass all handled key events to the time control for processing.

6 To display a changing time value to the user, call ITIMECTL_SetTime() or ITIMECTL_SetTimeEx() to update the display each time the value changes.

7 To access the time value entered by the user, call ITIMECTL_GetTime() when the user has completed entering the value (the user can signal completion by pressing the SELECT key or dismissing the screen that contains the time control). You can then call ITIMECTL_GetTimeString() to convert the number of milliseconds into a printable form.

8 When you no longer need the time control, call ITIMECTL_Release() to free it.

List of functions

Functions in this interface include:

ITIMECTL_AddRef()

ITIMECTL_EnableCommand()

ITIMECTL_GetProperties()

ITIMECTL_GetRect()

ITIMECTL_GetTime()

ITIMECTL_GetTimeString()

ITIMECTL_HandleEvent()

ITIMECTL_IsActive()

ITIMECTL_Redraw()

ITIMECTL_Release()

ITIMECTL_Reset()

ITIMECTL_SetActive()

ITIMECTL_SetEditField()

ITIMECTL_SetIncrement()

ITIMECTL_SetProperties()

ITIMECTL_SetRect()

ITIMECTL_SetTime()

ITIMECTL_SetTimeEx()

Return to the Contents

ITIMECTL_AddRef()

Description:

This function increments the reference count of the ITimeCtl Interface object. This allows the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 ITIMECTL_AddRef(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

Side Effects:

None

See Also:

ITIMECTL_Release()
Return to the List of functions

ITIMECTL_EnableCommand()

Description:

This function is used to enable sending of specified command by the time control object to the AEE Shell upon receiving the event generated by pressing center key.

Prototype:

```
void ITIMECTL_EnableCommand(ITimeCtl * pITimeCtl, boolean bEnable, uint16 nCmdId)
```

Parameters:

pITimeCtl	Pointer to I the ITimeCtl Interface object
bEnable	Boolean value for enable/disable flag
nCmdId	Command id

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_GetProperties()

Description:

This function returns the time control-specific properties or flags.

Prototype:

```
uint32 ITIMECTL_GetProperties(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

32-bit properties for the time control

Following properties are returned by time control object:

TP_AUTOREDRAW	If set, control redraws on SetActive.
TP_NO_SECONDS	If set, control doesn't show the seconds in the COUNTDOWN control.
TP_NO_MSECONDS	If set, control doesn't show milliseconds in STOPWATCH.
TP_NOEDIT_AMPM	If set, control doesn't allow to edit AM/PM.

Comments:

None

Side Effects:

None

See Also:

ITIMECTL_SetProperties()
Return to the List of functions

ITIMECTL_GetRect()

Description:

This function fills given pointer to AEERect with the coordinates of the bounding rectangle of the time control object. This is particularly useful after a control is created to determine its optimal/default size and position.

Prototype:

```
void ITIMECTL_GetRect(ITimeCtl * pITimeCtl, AEERect * prc)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
prc	Rectangle to be filled with the coordinates of the time control

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ITIMECTL_SetRect()
AEERect
Return to the List of functions

ITIMECTL_GetTime()

Description:

This function gets the time value from the time control object. The time value is in milliseconds.

Prototype:

```
int32 ITIMECTL_GetTime(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

Time	In milliseconds elapsed since 00:00:00 for this object

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_GetTimeString()

Description:

This function converts the time in milliseconds to a string in the specified format.

Prototype:

```
void ITIMECTL_GetTimeString(ITimeCtl * pITimeCtl, uint32 dwSecs, AECHAR *
pDest, unsigned int nSize, uint16 wFlags)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
dwSecs	Time in milliseconds to be converted into a string
pDest	Converted time in a string
nSize	Size Of **Pdest** In Bytes
wFlags	Bitmap that specifies the time format to use for the string conversion the following are supported: GTS_MSECS: use centiseconds, and display 2 digit centiseconds GTS_SECS: use seconds GTS_AMPM: use AM or PM

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_HandleEvent()

Description:

This function is used to handle the events received by time control object. A time control object handles events received by it only if it is active. The events processed by the time control object are the press of UP, DOWN, LEFT and RIGHT keys. If command sending is enabled for the time control object, upon receiving event generated by the press of center key, it sends the command specified by ITIMECTL_EnableCommand() function as command event to the AEE Shell.

Prototype:

```
boolean ITIMECTL_HandleEvent(ITimeCtl * pITimeCtl, AEEEvent evt, uint16 wp,
uint32 dwp)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
evt	Event code
wp	16-bit event data
dwp	32-bit event data

Return Value:

TRUE	If the event was processed by the time control
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_IsActive()

Description:

This function returns the active/inactive state of the ITimeCtl Interface object.

Prototype:

```
boolean ITIMECTL_IsActive(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

TRUE	If the time control is active
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_Redraw()

Description:

This function instructs the time control object to redraw its contents. The time control object does not redraw its contents every time the underlying data behind the time control changes. This allows several data updates to occur while minimizing screen flashes. For example, several changes can be made to the contents of the time control object with no visible effect until the Redraw function is called.

Prototype:

```
boolean ITIMECTL_Redraw(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

TRUE	If the time control was redrawn
FALSE	If otherwise

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_Release()

Description:

This function decrements the reference count for the ITimeCtl Interface object and does appropriate cleanup if the reference count reaches 0 (zero).

Prototype:

```
uint32 ITIMECTL_Release(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object whose reference count needs to be decremented

Return Value:

Updated reference count for the object.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_Reset()

Description:

This function instructs the time control to reset (free/delete) its contents as well as to immediately leave active/focus mode.

Prototype:

```
void ITIMECTL_Reset(ITimeCtl * pITimeCtl)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ITIMECTL_SetActive()
Return to the List of functions

ITIMECTL_SetActive()

Description:

This function is used to make a time control object active. Only an active time control object handles the event sent to it. An inactive time control object just ignores the events.

Prototype:

```
void ITIMECTL_SetActive(ITimeCtl * pITimeCtl,boolean bActive)
```

Parameters:

pITimeCtl	Pointer to t the ITimeCtl Interface object
bActive	Boolean flag that indicates: TRUE: to activate the time control object FALSE: to deactivate the time control object

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_SetEditField()

Description:

This function sets specified field for editing.

Prototype:

```
void ITIMECTL_SetEditField(ITimeCtl * pITimeCtl, ITField field)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
field	Field to be set for editing. Field can be one of ITF_HOUR ITF_MIN ITF_SEC

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_SetIncrement()

Description:

This function sets a new time increment value in minutes. The default is 1.

Prototype:

```
void ITIMECTL_SetIncrement(ITimeCtl * pITimeCtl, uint16 wMins)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
wMins	New time increment value in minutes

Return Value:

None

Comments:

The increment for only the minute field can be set. All other fields cannot be set.

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_SetProperties()

Description:

This function sets time control-specific properties.

Prototype:

```
void ITIMECTL_SetProperties(ITimeCtl * pITimeCtl, uint32 dwProps)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
dwProps	32-bit set of flags/properties. Following properties are used for time control object**:**
	TP_AUTOREDRAW: if set, redraw on SetActive
	TP_NO_SECONDS: if set, do not show the seconds in the COUNTDOWN control
	TP_NO_MSECONDS: if set, do not show milliseconds in STOPWATCH
	TP_NOEDIT_AMPM: if set, do not allow to edit AM/PM

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ITIMECTL_GetProperties()
Return to the List of functions

ITIMECTL_SetRect()

Description:

This function can be used to set the coordinates specified by **prc** as the control rectangle of the time control object.

Prototype:

```
void ITIMECTL_SetRect(ITimeCtl * pITimeCtl, const AEERect * prc)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
prc	Bounding rectangle for the time control

Return Value:

None

Comments:

By default, entire device screen is set as the control rectangle of the time control object.

Side Effects:

None

See Also:

ITIMECTL_GetRect()
Return to the List of functions

ITIMECTL_SetTime()

Description:

This function can be used to set given time and redraw time string on device screen. It is the same as calling ITIMECTL_SetTimeEx with the third parameter to be FALSE.

Prototype:

```
void ITIMECTL_SetTime(ITimeCtl * pITimeCtl, int32 tod)
```

Parameters:

pITimeCtl	Pointer to the ITimeCtl Interface object
tod	Time in milliseconds expired since 00:00:00. Negative value is ignored

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

ITIMECTL_SetTimeEx()

Description:

This function can be used to set a given time and redraw a time string on a device screen.

Prototype:

```
void ITIMECTL_SetTime(ITimeCtl * pITimeCtl, int32 tod, boolean bIncUpdate)
```

Parameters:

pITimeCtl	Pointer to I the ITimeCtl Interface object
tod	Time in milliseconds expired since 00:00:00, where negative values are ignored
bIncUpdate	Boolean flag: TRUE means to save the time string in the buffer of time control object. FALSE means to just draw on screen and don't save the time string.

Whether TRUE or FALSE, the time integer is saved.

Return Value:

None

Comments:

bIncUpdate affects only the time string of the ITimeCtl Interface object. Time integer is always saved.

Side Effects:

None

See Also:

None
Return to the List of functions

IViewer Interface

The IViewer Interface is identical to the IImage Interface. It includes an identical list of functions.

The **Descriptions** of the IImage functions also describe the IViewer functions (See List of functions)

The **Prototypes** are the same with the exception that each place where there is **IIMAGE**, it is replaced by **IVIEWER**.

The **Parameters** are the same as IImage's parameters. The parmeter **pIImiage** is replaced with **pIViewer** which is the IViewer Interface opject.

The **Return Values** are the same return values as shown in the IImage Interface.

The **Comments** applicable to the IImage functions are also applicable to the IViewer functions.

The **Side Effects** applicable to the IImage functions are also applicable to the IViewer functions.

The **See Also** relationship between functions in the IImage functions are also applicable between the IViewer functions.

List of functions

Functions in this interface include:

IVIEWER_AddRef()	**See** IIMAGE_AddRef()
IVIEWER_Draw()	**See** IIMAGE_Draw()
IVIEWER_DrawFrame()	**See** IIMAGE_DrawFrame()
IVIEWER_GetInfo()	**See** IIMAGE_GetInfo()
IVIEWER_HandleEvent()	**See** IIMAGE_HandleEvent()
IVIEWER_Notify()	**See** IIMAGE_Notify()
IVIEWER_Release()	**See** IIMAGE_Release()
IVIEWER_SetParm()	**See** IIMAGE_SetParm()
IVIEWER_SetStream()	**See** IIMAGE_SetStream()
IVIEWER_Start()	**See** IIMAGE_Start()
IVIEWER_Stop()	**See** IIMAGE_Stop()

Return to the Contents

Helper Functions

This section provides documentation for the various helper functions offered by AEE. This includes string functions, functions in the standard C library, utility functions and other items. standard C library refers to the ANSI standard C library supplied with C/C++ compilers/IDE. Applications must not directly invoke the standard C library functions (for example, memcpy). Instead, applications must use the functions provided by AEE (such as MEMCPY()). A distinct difference between the functions documented here and the rest of the AEE functions is that an interface pointer is not needed to access these functions. For example, to invoke the function MEMCPY(), no interface pointer is necessary. Applications can directly invoke MEMCPY(). Some of the functions provided here make direct calls to the standard C library functions (for example, MEMCPY() directly invokes the C library function memcpy()). However, these functions are provided here for two reasons:

- To prevent the need for every application to statically link with the standard C library. When there are multiple apps loaded on the device, each application has the extra baggage of carrying the standard C library. To avoid this, AEE maintains a single copy of the standard C library. All applications can make use of this copy. Applications must not make direct calls to the standard C library functions (thereby, preventing the application from being associated with the baggage of static C library)

- To use dynamic apps that must not have any static data associated with them. If applications were to make direct calls to the standard C library functions such as memcpy() , they need to include the corresponding header files (for example, memory.h) and these header files may contain static data. Hence, using the standard C library function would prevent the application from being dynamically loadable.

For the above two reasons, AEE offers the helper functions, some of which are wrappers that directly call the standard C library functions.

List of functions

Functions in this interface include:

ATOI()

CALLBACK_Cancel()

CALLBACK_Init()

CALLBACK_IsQueued()

CONVERTBMP()

CREATEOBJ()

DBGPRINTF()

FADD()

FCMP_E()

FCMP_G()

FCMP_GE()

FCMP_L()

FCMP_LE()

FDIV()

FLOAT_TO_WSTR()

FMUL()

FREE()

FREEOBJ()

FSUB()

GETAEEVERSION()

GET_APP_INSTANCE()

GETCHTYPE()

GET_JULIANDATE()

LOCALTIMEOFFSET()

GET_NOTIFIER_MASK()

GET_NOTIFIER_VAL()

GET_RAND()

GET_SECONDS()

GET_TIMEMS()

GET_UPTIMEMS()

MALLOC()

MEMCPY()

MEMSET()

OEMSTRLEN()

OEMSTRSIZE()

REALLOC()

SETAEERECT()

SPRINTF()

STR_TO_WSTR()

STRCAT()

STRCHR()

STRCMP()

STRCPY()

STRLEN()

STRNCPY()

STRRCHR()

STRTOUL()

SYSFREE()

UTF8_TO_WSTR()

WSPRINTF()

WSTR_TO_FLOAT()

WSTR_TO_STR()

WSTR_TO_UTF8()

WSTRCAT()

WSTRCHR()

WSTRCMP()

WSTRCOMPRESS()

WSTRCPY()

WSTRDUP()

WSTRLEN()

WSTRLOWER()

WSTRNCOPYN()

WSTRRCHR()

WSTRSIZE()

WSTRUPPER()

WWRITELONGEX()

Return to the Contents

ATOI()

Description:

This function is a wrapper around the atoi() function provided by the standard C library. Its behavior is identical to that of atoi(). This function converts the input string to an integer.

Prototype:

```
int ATOI( const char * src )
```

Parameters

src	Pointer to the source string

Return Value:

Integer	Represented by the input string

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

CALLBACK_Cancel()

Description:

This function cancels a callback that has been previously registered using the ISHELL_Resume() The callback can be cancelled only if the callback has not already happened. To check if the callback has already happened, check the member **pfnCancel** inside the AEECallback structure and check to see if it is NULL. It it has been set to NULL by the AEE, it cannot be cancelled.

Prototype:

```
void CALLBACK_Cancel(AEECallback * pcb)
```

Parameters:

pcb	Valid pointer to an AEECallback structure that has been registered using the ISHELL_Resume()

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_Resume()
Return to the List of functions

CALLBACK_Init()

Description:

This function initializes the members of an AEECallback structure.

Prototype:

```
void CALLBACK_Init(AEECallback * pcb, PFNNOTIFY pfn, void * pd)
```

Parameters:

pcb	Valid pointer to an AEECallback structure that must be initialized
pfn	Valid pointer to the callback function
pd	Pointer to data that must be passed to the **pfn** function when it is invoked

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

ISHELL_Resume()
Return to the List of functions

CALLBACK_IsQueued()

Description:

This function checks whether a callback that was registered with ISHELL_Resume() is still queued for processing. The callback is no longer queued if it has already occurred or if it was cancelled with the function CALLBACK_Cancel().

Prototype:

```
boolean CALLBACK_IsQueued(AEECallback * pcb)
```

Parameters:

pcb	Pointer to an AEECallback structure that has been registered with ISHELL_Resume()

Return Value:

TRUE	if the callback is still queued for processing
FALSE	otherwise

Comments:

If the callback is no longer queued, the **pfnCancel** member of the AEECallback structure referred by **pcb** is NULL. This function checks **pcb->pfnCancel** is NULL to determine whether callback is queued.

Side Effects:

None

See Also:

ISHELL_Resume()
Return to the List of functions

CONVERTBMP()

Description:

This function converts a Windows bitmap into the native format. The native format is specific to each device. On Windows, the native format is same as the Windows bitmap format. Typical usage of this function: If the user has a raw data buffer containing a Windows bitmap, it can be passed to this function to convert it into a native format so that the IDISPLAY_BitBlt() function can be used on this format to display the image. The BREW Emulator does not support 16 bit and 24 bit format .BMP file format.

Prototype:

```
void * CONVERTBMP(void * pSrcBuffer, AEEImageInfo * pii, boolean * pbRealloc)
```

Parameters:

pSrcBuffer	[in]	Pointer to a buffer containing the Windows bitmap. One of the ways of constructing this is to read the contents of a Windows .BMP file into a memory buffer. This buffer is converted to native format on return.
pii	[out]	On return, the AEEImageInfo structure pointed to by this member contains valid information about the converted image (such as width, height, color depth, and other items).
pbRealloc	[out]	On return, this is set to TRUE or FALSE depending on whether or not the CONVERTBMP() function did separate memory allocation for the buffer that is returned from this function. If this is set to TRUE, the caller must release the buffer returned from this function using SYSFREE() after the buffer is used.

Return Value:

Valid buffer containing the converted image	This is the incoming image **pSrcBuffer** converted into native format. If **pbRealloc** is set to true on return, the caller must release this buffer using SYSFREE() after using the buffer. On Windows, since the native format is the same as the .BMP format, the pointer returned from this function is identical to **pSrcBuffer**. Also, **bRealloc** is FALSE on Windows since no new allocation is done. The pii parameter, on return, contains information about the image (such as width, height, color depth, and other items).
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

CREATEOBJ()

Description:

This is an alias of MALLOC().

Prototype:

```
void * CREATEOBJ(dword dwSize)
```

Parameters:

dwSize	Specifies the size (in bytes) that must be allocated

Return Value:

Pointer	Pointing to a buffer of size dwSize bytes, if successful
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

MALLOC()
Return to the List of functions

DBGPRINTF()

Description:

This function is used in printing out debugging information.

Prototype:

```
void dbgprintf(const char * pszFormat,...)
```

Parameters:

pszFormat	Format-control string; please refer to the documentation of the standard C library function printf() on the corresponding platform (Microsoft Windows or ARM® Developer Suite)

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FADD()

Description:

This function does floating point "+" operation.

Prototype:

```
double FADD(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

Returns the result of "+"

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FCMP_E()

Description:

This function performs float point "==" comparison.

Prototype:

```
boolean FCMP_E(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

TRUE	If v1 == v2
FALSE	If v1 != v2

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FCMP_G()

Description:

This function performs float point ">" comparison.

Prototype:

```
boolean FCMP_G(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

TRUE	If v1 > v2
FALSE	If v1 <= v2

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FCMP_GE()

Description:

This function performs float point ">=" comparison.

Prototype:

```
boolean FCMP_GE(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

TRUE	If v1 >= v2
FALSE	If v1 < v2

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FCMP_L()

Description:

This function performs float point "<" comparison.

Prototype:

```
boolean FCMP_L(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

TRUE	If v1 < v2
FALSE	If v1 >= v2

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FCMP_LE()

Description:

This function performs float point "<=" comparison.

Prototype:

```
boolean FCMP_LE(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

TRUE	If v1 <= v2
FALSE	If v1 > v2

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FDIV()

Description:

This function does floating point "/" operation.

Prototype:

```
double FDIV(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

Returns the result of "/"

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FLOAT_TO_WSTR()

Description:

This function converts a floating point value to a wide string. Internally, this function first converts the given float value into a single-byte string using the standard C library function sprintf(). It then uses STR_TO_WSTR() to convert this single-byte string into a wide string.

Prototype:

```
boolean FLOAT_TO_WSTR(double v, AECHAR * psz, int nSize)
```

Parameters:

v	Floating point value that must be converted into a wide string
psz	Pointer to wide-string buffer to hold the resultant wide string
nSize	Size (in bytes) of the **psz** buffer

Return Value:

TRUE	If successful
FALSE	If unsuccessful, if **psz** is NULL, or if **nSize** <= 0

Comments:

None

Side Effects:

None

See Also:

WSTR_TO_FLOAT()
Return to the List of functions

FMUL()

Description:

This function does floating point "*" operation.

Prototype:

```
double FMUL(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

Returns the result of "*"

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

FREE()

Description:

This function corresponds to free() in the standard C library. Internally, it does more than just free(), though the external behavior is the same as free().

Prototype:

```
void FREE(void * po)
```

Parameters

po	Points to the memory to be freed

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

MALLOC()
REALLOC()()
Return to the List of functions

FREEOBJ()

Description:

This is an alias of FREE().

Prototype:

```
void FREEOBJ(void * pObj)
```

Parameters:

pObj	Pointer to the memory buffer that must be release. This buffer must have been allocated using the CREATEOBJ().

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

FREE()
Return to the List of functions

FSUB()

Description:

This function does floating point "-" operation.

Prototype:

```
double FSUB(double v1, double v2)
```

Parameters:

v1	Operand #1
v2	Operand #2

Return Value:

The result of "-"

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GETAEEVERSION()

Description:

This function retrieves information about the version number of the current BREW software being used. The version number is stored in w.x.y.z format where

w: denotes Major Version Number

x: denotes Minor Version Number

y: denotes a Sub-version number

z: denotes a build number

This function also places the information in the incoming string **pszString**. On the phone, this function can also be used to determine the chipset on which the BREW software is running.

Prototype:

```
uint32 GETAEEVERSION(byte * pszString, int nSize, uint16 wFlags)
```

Parameters:

pszString	Pointer to a buffer where the string containing the version number is to be placed. It can be of the form "1.0.0.15" or "1.0.0.15(MSM3100)" depending on whether or not the MSM information is requested.
nsize	Size of the **pszString** buffer in bytes
wFlags	Specifies the type of information to retrieve. It can be a combination of one or more of the following flags: **GAV_LATIN1: pszString** is returned as a Single byte string **GAV_MSM: pszString** contains the MSM chip number in addition to the build number For example, 1.0.0.15 (MSM3100). This flag is not supported on the BREW Emulator. It is supported only in the physical device environment.

Return Value:

A 32 bit number containing the version number

The information is organized as follows:

Hi Byte of Hi Word	Major Version Number
Low Byte of Hi Word	Minor Version Number

A 32 bit number containing the version number	The information is organized as follows:	
	Hi Byte of Lo Word	Sub-Version Number
	Lo Byte of Lo Word	Build Number

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GET_APP_INSTANCE()

Description:

This function returns the IApplet instance of the currently executing applet. It is valid during all BREW API calls, callbacks, and events.

Prototype:

```
IApplet * GET_APP_INSTANCE(void);
```

Parameters:

None

Return Value:

Returns the IApplet pointer to the currently executing applet.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GETCHTYPE()

Description:

This function returns the type (numeric, alpha, and other types) of a wide character.

Prototype:

```
TChType GETCHTYPE(AECHAR ch)
```

Parameters:

ch	Character whose type is to be determined

Return Value:

The type of the given character	The type can be any of following SC_ALPHA SC_DIGIT SC_WHITESPACE SC_UNKNOWN

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GET_JULIANDATE()

Description:

This function fills a JulianType data structure based upon the input second value. This value represents the number of seconds since Jan 6 1980 00:00:00 GMT, the device reference time. If the input value is 0 (zero), the current system time is returned.

Prototype:

```
void GET_JULIANDATE(uint32 dwSecs, JulianType * pDate)
```

Parameters:

dwSecs	Seconds since Jan 6 1980 GMT, the device reference time
pDate	Pointer to the structure that needs to be filled on return

Return Value:

None

Comments:

On Windows, the upper limit for **dwSecs** is the difference between the maximum value allowed by the **uint32** and the difference between device reference time Jan 6, 1980 GMT and PC reference time Jan 1, 1970.

Side Effects:

None

See Also:

GET_SECONDS()
GET_TIMEMS()
GET_UPTIMEMS()
Return to the List of functions

GET_NOTIFIER_MASK()

Description:

Returns Mask Value (lower 16-bits) from specified 32-bit notification mask.

Prototype:

```
uint16 GET_NOTIFIER_MASK(uint32 dwMasks)
```

Parameters:

dwMasks	32-bit notification mask

Return Value:

```
Mask Value (lower 16-bits)
```

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GET_NOTIFIER_VAL()

Description:

Returns Notification Match Value (upper 16-bits) from specified 32-bit notification mask.

Prototype:

```
uint16 GET_NOTIFIER_VAL(uint32 dwMasks)
```

Parameters:

dwMasks	32-bit notification mask

Return Value:

```
Notification Match Value (upper 16-bits)
```

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GET_RAND()

Description:

This function uses the random number generator on the device to fill an input buffer with an array of random values.

Prototype:

```
void Get_Rand(byte * pDest, int nSize)
```

Parameters:

pDest	Pointer to destination buffer
nSize	Size in bytes of the buffer

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

GET_SECONDS()

Description:

This function returns the number of seconds, adjusted for local time, since January 6, 1980 00:00:00 GMT.

Prototype:

```
uint32 GET_SECONDS()
```

Parameters:

None

Return Value:

Seconds	Elapsed since January 6, 1980 00:00:00 GMT

Comments:

None

Side Effects:

None

See Also:

GET_JULIANDATE()
GET_TIMEMS()
GET_UPTIMEMS()
Return to the List of functions

GET_TIMEMS()

Description:

This function returns the current time of day in milliseconds. The value returned by this call depends on the device's current time of day value. On phone, this value is obtained from the base station and may change dramatically when the phone first acquires system coverage.

Prototype:

```
uint32 GET_TIMEMS()
```

Parameters:

None

Return Value:

Millisecond	Elapsed since midnight

Comments:

None

Side Effects:

None

See Also:

GET_JULIANDATE()
GET_SECONDS()
GET_UPTIMEMS()
Return to the List of functions

GET_UPTIMEMS()

Description:

This function returns the millisecond elapsed since the time the device was powered on. Unlike GET_TIMEMS(), this value does not change dramatically due to initial acquisition of system coverage.

Prototype:

```
uint32 GET_UPTIMEMS()
```

Parameters:

None

Return Value:

Milliseconds	Elapsed since the time the device was powered on

Comments:

On Emulator, this function returns the number of milliseconds elapsed since the present device configuration was selected in the Emulator.

Side Effects:

None

See Also:

GET_TIMEMS()
GET_SECONDS()
GET_JULIANDATE()
Return to the List of functions

LOCALTIMEOFFSET()

Description:

This function returns the local time zone offset from UTC, in seconds. Optionally returns a flag indicating that daylight savings time is active (if it is, the value of the local time zone offset already takes the shift into account; the flag is just for controlling display of a time zone name if desired). The returned value is added to UTC to give local time, or subtracted from local time to give UTC time. UTC = local time - bias (where, bias is the value returned from this function).

Prototype:

```
int32 LOCALTIMEOFFSET(boolean * pbDaylightSavings);
```

Parameters:

pbDaylightSavings	[in/out]	If non NULL on input, this flag specifies whether or not Daylight savings time is active on return

Return Value:

Returns the local time zone offset from UTC in seconds.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

MALLOC()

Description:

This function is corresponding to malloc() in standard C library.

Prototype:

```
void * MALLOC(dword dwSize)
```

Parameters

dwSize	Size of buffer in bytes

Return Value:

Pointer	To the allocated memory

Comments:

None

Side Effects:

None

See Also:

FREE()
REALLOC()
Return to the List of functions

MEMCPY()

Description:

This function is a wrapper around the memcpy() function provided by the standard C library. Its behavior is identical to that of memcpy(). This function copies **count** bytes of **src** to **dest**.

Prototype:

```
void * MEMCPY( void * dest, const void * src, uint16 count )
```

Parameters:

dest	Pointer to the destination buffer
src	Pointer to the source buffer
count	Specifies the number of bytes that must be coped from **src** to **dest**

Return Value:

a pointer to the dest buffer	If successful

Comments:

None

Side Effects:

None

See Also:

MEMSET()
Return to the List of functions

MEMSET()

Description:

This function is a wrapper around the memset() function provided by the standard C library. Its behavior is identical to that of memset(). This function sets the first **count** bytes of the **dest** buffer to a specified byte **b**.

Prototype:

```
void * MEMSET( void * dest, byte b, uint16 count )
```

Parameters:

dest	Pointer to the destination buffer
b	Specifies the byte that must be copied into the dest buffer
count	Specifies the number of bytes in dest that must be set to the value b

Return Value:

Pointer	To the destination buffer, if successful

Comments:

None

Side Effects:

None

See Also:

MEMCPY()
Return to the List of functions

OEMSTRLEN()

Description:

This function returns the length of a byte string. The string must be null-terminated. If the first byte of a character is greater than 0x7f, it is treated as a 2-byte character. The two bytes are counted as one single character.

Prototype:

```
int OEMSTRLEN(byte * p)
```

Parameters:

p	Valid pointer to a null-terminated byte string

Return Value:

The length of byte string	If successful
0 (zero)	If p is NULL

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

OEMSTRSIZE()

Description:

This function returns the size, in bytes, of the byte string, which must be null-terminated.

Prototype:

```
int OEMSTRSIZE(byte * p)
```

Parameters:

p	Pointer to a null-terminated byte string

Return Value:

The size of byte string	In bytes, if successful,
0 (zero)	If p is NULL

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

REALLOC()

Description:

This function is corresponding to realloc() in standard C library.

Prototype:

```
void * REALLOC(void * pSrc, uint32 dwSize)
```

Parameters:

pSrc	Points to the previously allocated memory
dwSize	New size in bytes

Return Value:

Pointer	To the newly allocated memory

Comments:

None

Side Effects:

None

See Also:

MALLOC()
FREE()
Return to the List of functions

SETAEERECT()

Description:

This function initializes the specified rectangle with origin, width and height.

Prototype:

```
void SETAEERECT(AEERect * rc, int x, int y, int cx, int cy)
```

Parameters:

rc	Rectangle to be initialized
x	x-coordinate of the rectangle origin
y	y-coordinate of the rectangle origin
cx	width
cy	height

Return Value:

None

Comments:

No validation is performed on specified values. Proving a NULL rectangle resulst in a crash.

Side Effects:

None

See Also:

None.
Return to the List of functions

SPRINTF()

Description:

This function is a wrapper around the sprintf() function provided by the standard C library. Its behavior is identical to that of sprintf(). It writes formatted data into a string. This function always takes single-byte character string as input. To work with wide strings, use WSPRINTF().

Prototype:

```
int SPRINTF( char * buffer, const char * format [, argument] ... )
```

Parameters:

buffer	Storage location for output
format	Format-control string
argument	Optional arguments

For more information about the function or its arguments, please refer to the documentation of the standard C library function sprintf() on the corresponding platform (Windows / ARM).

Return Value:

Number	Of bytes stored in buffer, not counting the terminating null character

Comments:

This function does not support %f in the format string. If %f is found anywhere within the format string, this function returns 0 (zero) without doing any processing.

Side Effects:

None

See Also:

WSPRINTF()
Return to the List of functions

STR_TO_WSTR()

Description:

This function converts a single-byte string into a wide string

Prototype:

```
AECHAR * STR_TO_WSTR(char * pszIn, AECHAR * pDest, int nSize)
```

Parameters:

pszIn	Pointer to null terminated string comprised of single-byte characters
pDest	Pointer to destination buffer to receive the wide string
nSize	Size (in bytes) of **pDest** buffer. If this is <= 0, this function does not do any conversion. It returns **pDest** as is without any changes.

Return Value:

The destination string	If successful

Comments:

None

Side Effects:

None

See Also:

WSTR_TO_STR()
Return to the List of functions

STRCAT()

Description:

This function is a wrapper around the **strcat()** function provided by the standard C library. Its behavior is identical to that of **strcat()**. This function appends the **src** string into **dest** and returns the **dest** string. All characters up to and including the first null character in **src** are appended to **dest**. This function always takes single-byte character strings as input. Applications wanting to append the wide strings must always use the WSTRCAT() function.

Prototype:

```
char * STRCAT( char * dest, const char * src )
```

Parameters:

dest	Pointer to the NULL terminated destination string
src	Pointer to the NULL terminated source string

Return Value:

Pointer	To the destination string, if successful

Comments:

None

Side Effects:

None

See Also:

WSTRCAT()
Return to the List of functions

STRCHR()

Description:

This function is a wrapper around the **strchr()** function provided by the standard C library. Its behavior is identical to that of **strchr()**. This function finds a character in a string. This function always takes single-byte character as input.

Prototype:

```
char * STRCHR( const char * string, int c )
```

Parameters:

string	Null terminated string to search
c	Character to be located

Return Value:

Pointer	To the first occurrence of **c** in the **string**
NULL	If **c** is not found

Comments:

None

Side Effects:

None

See Also:

WSTRCHR()
Return to the List of functions

STRCMP()

Description:

This function is a wrapper around the strcmp() function provided by the standard C library. Its behavior is identical to that of strcmp(). This function compares the given two single-byte strings. This function always takes single-byte character as input.

Prototype:

```
int STRCMP( const char * str1, const char * str2 )
```

Parameters:

str1,str2	Pointers to the two null terminated strings that need to be compared

Return Value:

value < 0	If string1 less than string2
0 (zero)	If string1 identical to string2
value > 0	If string1 greater than string2

Comments:

None

Side Effects:

None

See Also:

WSTRCMP()
Return to the List of functions

STRCPY()

Description:

This function is a wrapper around the strcpy() function provided by the standard C library. Its behavior is identical to that of strcpy(). This function copies the src string into dest and returns the dest string. All characters up to and including the first null character in src are copied into dest. This function always takes single-byte character strings as input. Applications wanting to copy wide strings from one buffer into the other must always use the WSTRCPY() function.

Prototype:

```
char * STRCPY( char * dest, const char * src )
```

Parameters:

dest	Pointer to the destination string
src	Pointer to the source string

Return Value:

Pointer to the destination string	If successful

Comments:

None

Side Effects:

None

See Also:

WSTRCPY()
WSTRNCOPYN()
Return to the List of functions

STRLEN()

Description:

This function is a wrapper around the strlen() function provided by the standard C library. Its behavior is identical to that of strlen(). This function gets the length of the given null terminated string. This function always takes single-byte character as input. To get the length of a wide string, use WSTRLEN().

Prototype:

```
int STRLEN( const char * str)
```

Parameters:

str	Null terminated string whose length is to be determined

Return Value:

Number	Of characters in string, excluding the terminal NULL

Comments:

None

Side Effects:

None

See Also:

WSTRLEN()
Return to the List of functions

STRNCPY()

Description:

This function is a wrapper around the standard C library function **strncpy()**. It copies specified number of characters from one string to another. The **strncpy()** function copies the initial count characters of **strSource** to **strDest** and returns **strDest**. If count is less than or equal to the length of **strSource**, a null character is not appended automatically to the copied string. If count is greater than the length of **strSource**, the destination string is padded with null characters up to length count. The behavior of **strncpy** is undefined if the source and destination strings overlap.

Prototype:

```
char * STRNCPY( char * strDest, const char * strSource, size_t count );
```

Parameters:

strDest	[in/out]	Destination string
strSource	[in/out]	Source string
count	[in]	Number of characters to be copied

Return Value:

This function returns **strDest**.

Comments

None

Side Effects

None

See Also:

None
Return to the List of functions

STRRCHR()

Description:

This function is a wrapper around the strrchr() function provided by the standard C library. Its behavior is identical to that of strrchr(). This function searches a string for the last occurrence of a character This function always takes a single-byte character strings as input.

Prototype:

```
char * STRRCHR( const char * string, int c )
```

Parameters:

string	Null terminated string to search
c	Character to be located

Return Value:

Pointer	A pointer to the last occurrence of c in string, if found
NULL	If c is not found

Comments:

None

Side Effects:

None

See Also:

WSTRRCHR()
Return to the List of functions

STRTOUL()

Description:

This function is a wrapper around the standard C library function **strtoul()**. It converts strings to an unsigned long-integer value. It stops reading the string **nptr** at the first character it cannot recognize as part of a number. This may be the terminating null character, or it may be the first numeric character greater than or equal to base. If **endptr** is not NULL, a pointer to the character that stopped the scan is stored at the location pointed to by **endptr**. If no conversion can be performed (no valid digits were found or an invalid base was specified), the value of **nptr** is stored at the location pointed to by **endptr**.

Prototype:

```
unsigned long STRTOUL( const char * nptr, char * * endptr, int base );
```

Parameters:

nptr	[in]	Null-terminated string to convert
endptr	[out]	If Non-null in input, it points to character that stops scan on return
base	[in]	Number base to use

Return Value:

This function returns the converted value. It returns 0 if no conversion can be performed.

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

SYSFREE()

Description:

This function is provided to allow developers to return system memory pointers returned from CONVERTBMP(). This function is ONLY valid for memory objects returned by CONVERTBMP() where the **bReallocated** flag has been set to TRUE.

Prototype:

```
void SYSFREE(void * pBuff);
```

Parameters:

pBuff	Pointer to memory allocated by CONVERTBMP()

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

CONVERTBMP()
Return to the List of functions

UTF8_TO_WSTR()

Description:

This function converts a converts a UTF8 string to a wide string.

Prototype:

```
boolean UTF8_TO_WSTR(const byte * pSrc,int nLen, AECHAR * pDst, int nSize)
```

Parameters:

pSrc	pointer to Null terminated Input string
nLen	Length of input string in bytes
pDst	Destination string
nSize	Size in bytes of destination

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

WSTR_TO_UTF8()
Return to the List of functions

WSPRINTF()

Description:

This function is the wide-string equivalent of sprintf(). It writes formatted data into a string. This function has certain limitations compared to sprintf(). Conversion specifiers for strings must have the form "%s" (alignment and field-width information, for example, %-20.10s, is not allowed).

Prototype:

```
void WSPRINTF(AECHAR * pDest, int nSize, AECHAR * pFormat,...)
```

Parameters:

pDest	Storage location for output
nSize	Specifies the total size (in bytes) of pDest buffer
pFormat	Format control string

For more information about the function or its arguments, please refer to the documentation of the standard C library function sprintf() on the corresponding platform (Windows / ARM).

Return Value:

None

Comments:

This function does not support %f in the format string. If %f is found anywhere within the format string, this function fills a null character in the starting location of the destination string and returns immediately without doing any processing.

Side Effects:

None

See Also:

SPRINTF()
Return to the List of functions

WSTR_TO_FLOAT()

Description:

This function converts a wide string into a floating point value. Internally, this function first converts the given wide string into a single-byte string using the WSTR_TO_STR() and then uses the standard C library function atof() to convert the single-byte string to a double value.

Prototype:

```
double WSTR_TO_FLOAT(AECHAR * psz)
```

Parameters:

psz	Pointer to null terminated wide string that must be converted to float

Return Value:

Floating point value	Of the given string, if successful

Comments:

None

Side Effects:

None

See Also:

FLOAT_TO_WSTR()
Return to the List of functions

WSTR_TO_STR()

Description:

This function converts a wide string into a single-byte string.

Prototype:

```
char * WSTR_TO_STR(AECHAR * pIn, char * pszDest, int nSize)
```

Parameters:

pIn	Pointer to null terminated wide string that must be converted to single-byte character string
pszDest	Pointer to destination buffer to receive the single-byte string
nSize	Size (in bytes) of **pDest** buffer. If this is <= 0, this function does not do any conversion. It returns **pszDest** as is without any changes.

Return Value:

The destination string	If successful

Comments:

None

Side Effects:

None

See Also:

STR_TO_WSTR()
Return to the List of functions

WSTR_TO_UTF8()

Description:

This function converts a wide string to a UTF8 string.

Prototype:

```
boolean WSTR_TO_UTF8(const AECHAR * pSrc,int nLen, byte * pDst, int nSize)
```

Parameters:

pSrc	Input string
nLen	Length of input string in AECHARs
pDst	Destination string
nSize	Size in bytes of destination

Return Value:

TRUE	If successful
FALSE	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

WSTR_TO_UTF8()
Return to the List of functions

WSTRCAT()

Description:

This function appends the src string into dest. All characters up to and including the first null character in src are appended to dest. Both src and dest are wide strings.

Prototype:

```
AECHAR * WSTRCAT(AECHAR * pDest, AECHAR * pSrc)
```

Parameters:

pDest	pointer to the NULL terminated destination string
pSrc	pointer to the NULL terminated source string

Return Value:

Pointer	To the destination string, if successful

Comments:

None

Side Effects:

None

See Also:

STRCAT()
Return to the List of functions

WSTRCHR()

Description:

This function is the wide string counterpart of STRCHR(). Its behavior is identical to that of STRCHR(). This function always takes a wide string as input.

Prototype:

```
AECHAR * WSTRCHR( AECHAR * s1, AECHAR c )
```

Parameters:

s1	Null terminated wide string to search
c	Character to be located

Return Value:

Pointer	To the first occurrence of **c** in **s1**
NULL	If **c** is not found

Comments:

None

Side Effects:

None

See Also:

STRCHR()
Return to the List of functions

WSTRCMP()

Description:

This function compares the two strings, **s1** and **s2**, lexicographically. It returns an integer value that indicates the comparison result. In this function, NULL string pointer and empty string are treated the same. Both **s1** and **s2** are wide strings.

Prototype:

```
int WSTRCMP(AECHAR * s1, AECHAR * s2)
```

Parameters:

s1	Pointer to first NULL terminated string
s2	Pointer to second NULL terminated string

Return Value:

0	If s1 == s2
1	If s1 > s2
-1	If s1 < s2

Comments:

None

Side Effects:

None

See Also:

STRCMP()
Return to the List of functions

WSTRCOMPRESS()

Description:

This function compresses the input wide string: if a character is less than or equal to 127, then the function reduces the two bytes to one byte.

Prototype:

```
void WSTRCOMPRESS(const AECHAR * pSrc, int nLen,byte * pDest, int nSize)
```

Parameters:

pSrc	Null terminated source string
nLen	Length of the source string in AECHARs
pDest	Pointer to the destination buffer
nSize	Size of the destination buffer in bytes

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

STRLEN()
Return to the List of functions

WSTRCPY()

Description:

This function copies the **src** string into **dest**. Both **src** and **dest** are wide strings.

Prototype:

```
AECHAR * WSTRCPY(AECHAR * pDest, AECHAR * pSrc)
```

Parameters:

pDest	Pointer to destination buffer
pSrc	Pointer to null-terminated string that must be copied into **pDest**

Return Value:

Pointer	To the destination string, if successful

Comments:

None

Side Effects:

None

See Also:

WSTRNCOPYN()
STRCPY()
Return to the List of functions

WSTRDUP()

Description:

This is a convenience function that can be used make a duplicate copy of an existing wide string. This function first allocates memory (using the CREATEOBJ()) required to store a copy of the incoming string and then copies this string into the newly allocated buffer. This newly allocated buffer is then returned from the function. After use, the buffer must be released using the FREEOBJ().

Prototype:

```
AECHAR * WSTRDUP(AECHAR * pIn)
```

Parameters:

pIn	Valid pointer to a null-terminated wide string

Return Value:

pointer	Containing a copy of the incoming string pIn, if successful. After using this pointer, it must be released using the FREEOBJ().
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

WSTRLEN()

Description:

This function gets the length of the given null terminated wide string.

Prototype:

```
int WStrLen(AECHAR * str)
```

Parameters:

str	Null terminated string whose length is to be determined

Return Value:

Number of AECHAR **characters**	In string excluding the terminal, if successful
0 (zero)	If **str** is NULL or if **str** is empty

Comments:

None

Side Effects:

None

See Also:

STRLEN()
Return to the List of functions

WSTRLOWER()

Description:

This function converts all upper case characters in a wide string to lower case.

Prototype:

```
void WSTRLOWER(AECHAR * pszDest)
```

Parameters:

pszDest	On input, this is a pointer to NULL terminated source string. On return, this buffer contains the converted string.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

WSTRUPPER()
Return to the List of functions

WSTRNCOPYN()

Description:

This function copies specified length of the source string into destination. The destination is guaranteed to be null-terminated. Both **src** and **dest** are wide strings.

Prototype:

```
AECHAR * WSTRNCOPYN(AECHAR * pDest, int cbDest, AECHAR * pSrc, int lenSource)
```

Parameters:

pDest	[out]	Pointer to destination buffer
cbDest	[in]	Size of **pDest** in AECHARs
pSrc	[in]	Pointer to null-terminated string that must be copied into **pDest**
lenSource	[in]	Maximum string length to copy into **pDest**. It does not need to include null character. If it is set to -1, then the entire pSrc is copied to **pDest**

Return Value:

Destination string

Comments:

None

Side Effects:

None

See Also:

WSTRCPY()
Return to the List of functions

WSTRRCHR()

Description:

This function is the wide string counterpart of STRRCHR(). Its behavior is identical to that of STRRCHR(). This function always takes a wide string as input.

Prototype:

```
AECHAR * WSTRRCHR( AECHAR * s1, AECHAR c )
```

Parameters:

s1	Null terminated wide string to search
c	Character to be located

Return Value:

Pointer	To the first occurrence of **c** in **s1**,
NULL	If **c** is not found

Comments:

None

Side Effects:

None

See Also:

STRRCHR()
Return to the List of functions

WSTRSIZE()

Description:

This function returns the size, in bytes, of the wide string.

Prototype:

```
int WSTRSIZE(AECHAR * p)
```

Parameters:

p	Valid pointer to a null-terminated wide string

Return Value:

The size of wide string	In bytes
0 (zero)	If p is NULL

Comments:

None

Side Effects:

None

See Also:

None
Return to the List of functions

WSTRUPPER()

Description:

This function converts all lower case characters in a wide string to upper case

Prototype:

```
void WSTRUPPER(AECHAR * pszDest)
```

Parameters:

pszDest	On input, this is a pointer to NULL terminated source string. On return, this buffer contains the converted string.

Return Value:

None

Comments:

None

Side Effects:

None

See Also:

WSTRLOWER()
Return to the List of functions

WWRITELONGEX()

Description:

This function converts a long into wide string. The string may be padded with wide character '0' and may contain only a portion of the long input value based on the parameters **nPad** and **nRemaining**.

Prototype:

```
AECHAR * WWRITELONGEX(AECHAR * pszBuf, long n, int nPad, int * pnRemaining)
```

Parameters:

pszBuf	[out]	Valid pointer to a null-terminated wide string to which the converted long is written
n	[in]	Input long number
nPad	[in]	Specifies the maximum amount of padding. The value of **nPad** must be less than 12 otherwise it is ignored. The padding is the number of digits that must be present in the final string. For example, if **n** is set to 245 and **nPad** is set to 5, then the output string contains 00245. If this value is greater than or equal to 12 or if it is less than the minimal size needed to represent the number, then it is ignored. For example, if **n** is set to 245 and **nPad** is set to 1, it is ignored. If **npad** is set to 0 or a negative number, it is ignored.
nRemaining	[in/out]	This pointer, on input, points to an integer that specifies the size of **pszBuf**. The size is described in terms of the total number of characters (AECHARs). This pointer, on output, points to an integer that denotes the unused space (in terms of number of AECHARs) left in **pszBuf**. The difference between the output and input values denotes the number of characters written to **pszBuf** (excluding the null character).

Return Value:

Pointer	To the beginning of the unused portion of **pszBuf**. This is the portion in **pszBuf** after the formatted wide string has been written.
NULL	If unsuccessful

Comments:

None

Side Effects:

None

See Also:

WSPRINTF()
Return to the List of functions

Data Structures

This section contains type definitions and descriptions of the data structures used by the BREW API functions. These data structures define the format and content of the data that is passed by applications to the BREW API functions and received by the applications as output from the functions. Type definitions for the BREW data structures are contained in the BREW header files that are shipped with the BREW SDK. Most data structures are specific to a particular BREW interface, and their type definitions are contained in the header file for that interface. Many data structures that are used by more than one interface are found in the files AEE.h and AEEError.h. The descriptions in this section are in alphabetical order by data structure name. The description of each BREW API function contains links to the descriptions of all relevant data structures.

BREW data structures are of three main types:

- **Structures and Unions:** Many BREW functions take pointers to structures as input parameters. To use such a function, you populate an instance of a structure and pass a pointer to the instance when calling the function. For example, the IGraphics shape-drawing functions have structures as input parameters that define the dimensions of the shape to be drawn. Many BREW functions return pointers to structures as output; for example, the IFile Interface and IImage Interface functions that return information about files and images store this information in structures. In this section, each field in each of the BREW structures is described.

- **Enumerated Types:** Many BREW variables and structure members take on values from a finite set defined by the C typedef enum construct. For example, the font types supported by the IDisplay Interface's text-drawing functions are specified with an enumerated-type definition. This section describes each value of each enumerated type.

- **Constant Definitions:** The BREW API functions make use of a number of constants that are defined with the #define construct. One common use of constants is to define a set of bit masks for testing and setting the values of the bits in a bit-vector variable. For example, the BREW menu, time, text and static text controls all have a 32-bit variable used to store control properties, with one bit per property. Each control defines a set of bit-mask constants that are used to test and set the values of each of the control's properties. In this section, each set of related constants used by the BREW API functions is described.

List of data structures

Data structures in this interface include:

AEE Applet Flags

AEE Events

AEE IImage Parameters

AEE IMenuCtl Properties

AEE ITextCtl Properties

AEE ITimeCtl Properties

AEE Privilege Levels

AEE Standard Control Properties

AEEAppInfo

AEEAppStart

AEEArc

AEECallback

AEECircle

AEEClip

AEEClipShape

AEEClrItem

AEEDBField

AEEDBFieldName

AEEDBFieldType

AEEDeviceInfo

AEEDNSResult

AEEEllipse

AEEFrameType

AEEFont

AEEHandlerType

AEEImageInfo

AEEItemStyle

AEEItemType

AEELine

PFNIMAGEINFO

PFNPOSITIONCB

PFNSOUNDPLAYERSTATUS

PFNSOUNDSTATUS

ResType

RGBVAL

SockIOBlock

TChType

Return to the Contents.

AEE Applet Flags

Description:

Applet specific flags.

Definition:

AFLAG_HIDDEN	Applet is hidden
AFLAG_CFG	Applet has a CFG menu
AFLAG_SYSTEM_CFG	Applet has a hidden CFG menu
AFLAG_TOOL	Applet is a tool
AFLAG_GAME	Applet is a game
AFLAG_PIM	Applet is a PIM
AFLAG_WEB	Applet is a Web Applet
AFLAG_STATIC	System use only
AFLAG_DYNAMIC	System use only

Members:

None

Comments:

None

See Also:

None
Return to the List of data structures.

AEE Events

Description:

These are the defined AEE events that can be received by an applet and/or control. For each event the **wParam** and **dwParam** parameters, if any, that are passed to the applet or control are given.

Definition:

Applet Events

EVT_APP_START	Main App started
EVT_APP_STOP	App stopped: no parameters
EVT_APP_SUSPEND	App suspended: no parameters
EVT_APP_RESUME	App resumed: dwParam = (AEEAppStart *)
EVT_APP_CONFIG	Alternate App Start: configuration screen can be shown
EVT_APP_HIDDEN_CONFIG	Alternate App Start: hidden configuration screen
EVT_APP_BROWSE_URL	Called after EVT_APP_START: dwParam = (const AECHAR * pURL)
EVT_APP_BROWSE_FILE	Called after EVT_APP_START
EVT_APP_MESSAGE	Text message: dwParam = ASCIIZ

Key Events

EVT_KEY	App keyup: **wParam** = KEYCODE
EVT_KEY_PRESS	App keydown: **wParam** = KEYCODE
EVT_KEY_RELEASE	App keyheld: **wParam** = KEYCODE
EVT_KEY_HELD	Key held: **wParam** = KEYCODE

Control Events

EVT_COMMAND	App custom control
EVT_CTL_TAB	App TAB event sent by controls: dwParam = control, wParam = 0-left, 1-right
EVT_CTL_SET_TITLE	Message interface to set title: wParam = ID, dwParam = res file (if ID != 0) or text
EVT_CTL_SET_TEXT	Message interface to set text: wParam = ID, dwParam = res file (if ID != 0) or text
EVT_CTL_ADD_ITEM	Message interface to add item: dwParam = CtlAddItem
EVT_CTL_CHANGING	App dwParam = CtlValChange

EVT_CTL_CHANGING	App dwParam = CtlValChange
EVT_CTL_MENU_OPEN	Sent by text controls before their associated softkey menus are activated

Dialog Events

EVT_DIALOG_INIT	Dialog Event: Controls created, pre-init values, flags, and other items
EVT_DIALOG_START	Dialog Event: Dialog opening, **wParam** = ID, **dwParam** = IDialog *
EVT_DIALOG_END	Dialog Event: Dialog completed normally, **wParam** = ID, **dwParam** = IDialog *
EVT_COPYRIGHT_END	Dialog Event: Copyright dialog ended

AEE Shell Events

EVT_ALARM	App **wParam** = uCode
EVT_NOTIFY	dwParam = AEENotify *
EVT_BUSY	0x404

Device Events

EVT_FLIP	wParam = TRUE if open, FALSE if closed
EVT_LOCKED	wParam = TRUE if user interface is locked
EVT_KEYGUARD	wParam = TRUE if keyguard is on

User defined events

EVT_USER	Start of App/User defined Events (Private to apps)

Members:

None

Comments

The user defined events start from EVT_USER

See Also:

None
Return to the List of data structures.

AEE IImage Parameters

Description:

These are the parameters defined for IImage Interface.

Definition:

IPARM_SIZE	Specifies the actual size of the image that needs to be used for displaying purposes
IPARM_OFFSET	Specifies the offset within the entire image that must be used for displaying
IPARM_CXFRAME	Specifies the width of each frame for formats not normally supporting animation (such as Windows .BMP)
IPARM_NFRAMES	Specifies the number of frames
IPARM_RATE	Specifies the animation rate in milliseconds
IPARM_ROP	Specifies the Raster operation to be used while drawing the image
IPARM_OFFSCREEN	Specifies whether to draw the image to the offscreen buffer

Members:

None

Comments:

None

See Also:

IIMAGE_SetParm()
Return to the List of data structures.

AEE IMenuCtl Properties

Description:

These are the properties defined for IMenuCtl Interface.

Definition:

MP_WRAPSCROLL	If set, wrap when scrolling off the end of screen (Only applicable to SoftKey and List controls)
MP_NO_ARROWS	If set, no arrows even if scroll is possible
MP_NO_REDRAW	If set, IMENUCTL_Redraw() function is not internally called in IMENUCTL_SetActive() or when changing selection
MP_UNDERLINE_TITLE	If set, underline title
MP_BI_STATE_IMAGE	If set, 2 state image with no framing (unsel/sel)
MP_TRI_STATE_IMAGE	If set, 3 state image with no framing (unsel/sel/pressed)

Properties valid only for SoftKey list view of the menu control object:

MP_MAXSOFTKEYITEMS	If set, show maximum number of soft key items per screen

Properties valid only for icon list view of the menu control object:

MP_ICON_TEXT_TOP	If set, Icon View: Text at top
MP_ICON_SINGLE_FRAME	If set, Icon View: Single Frame

Properties valid only for calendar event list view of the menu control object:

MP_CALENDAR	If set, menu control object is in calendar event list view
MP_AUTOSCROLLTIME	If set, auto-scroll if in calendar list view

Members:

None

Comments

None

See Also:

IMENUCTL_SetProperties()
IMENUCTL_GetProperties()
Return to the List of data structures.

AEE ITextCtl Properties

Description:

These are the properties defined for ITextCtl Interface.

Definition:

TP_MULTILINE	If set, text control object is multiple line control
TP_FRAME	If set, text control object has a frame
TP_T9_MODE	If set, text control object is in T9 mode

Members:

None

Comments

None

See Also:

ITEXTCTL_SetProperties()
ITEXTCTL_GetProperties()
Return to the List of data structures.

AEE ITimeCtl Properties

Description:

These are the properties defined for ITimeCtl Interface.

Definition:

TP_AUTOREDRAW	if set, redraw on SetActive
TP_NO_SECONDS	if set, do not show the seconds in the COUNTDOWN control
TP_NO_MSECONDS	if set, do not show milliseconds in STOPWATCH
TP_NOEDIT_AMPM	if set, do not allow to edit AM/PM

Members:

None

Comments:

None

See Also:

ITIMECTL_SetProperties(),
ITIMECTL_GetProperties()
Return to the List of data structures.

AEE Privilege Levels

Description:

These flags define the privilege levels that an application can have. An application can have zero or more privilege levels, which are stored internally in a bit vector. These flags can be used to test and set the value of each privilege bit in this vector.

Definition:

PL_FILE	The application has create and write access to files and databases
PL_NETWORK	The application has access to the functions in the INetMgr Interface and ISocket Interface.
PL_TAPI	The application has access to telephony functionality (not supported at present).
PL_DOWNLOAD	The application has access to the IDownload interface, which contains functions for accessing BREW application download servers (this privilege level is only available to carriers and device manufacturers).
PL_SHARED_WRITE	The application has access to the shared application directory, which allows applications to share files.
PL_POS_LOCATION	The application has access to position-location functionality (not supported at present).
PL_SYSTEM	The application has all of the above privilege levels and additional functionality (this privilege level is only available to carriers and device manufacturers).

Members:

None

Comments:

None

See Also:

ISHELL_CheckPrivLevel()
Return to the List of data structures.

AEE Standard Control Properties

Description:

These are the defined Standard Control Properties.

Definition:

CP_BORDER	Control has a border
CP_STATIC	Control is static and SetActive has no effect with this control type
CP_3D_BORDER	3D Border
CP_USE_DEFAULT	Use default properties

Members:

None

Comments

None

See Also:

None
Return to the List of data structures.

AEEAppInfo

Description:

This structure is used for storing information about the applet.

Definition:

```
typedef struct
        {
        AEECLSID cls;
        char * pszMIF;
        uint16 wIDBase;
        uint16 wPad1;
        uint16 wPad2;
        uint16 wPad3;
        uint16 wPad4;
        uint16 wFlags;
        } AEEAppInfo;
```

Members:

cls	Applet ClassID
pszMIF	Applet Resource file
wIDBase	Base ID for locating title, icon, and other items in applet resource file
wPad1	Padding
wPad2	Padding
wPad3	Padding
wPad4	Padding
wFlags	AEE Applet Flags

Comments:

None

See Also:

AEE Applet Flags
Return to the List of data structures.

AEEAppStart

Description:

This structure is sent on EVT_APP_START/EVT_APP_RESUME.

Definition:

```
typedef struct
        {
        int error;
        AEECLSID clsApp;
        IDisplay * pDisplay;
        AEERect rc;
        } AEEAppStart;
```

Members:

error	Filled by app if there is an error
clsApp	Applet ID
pDisplay	Pointer to IDisplay Interface object
rc	Rectangle for the Applet

Comments:

None

See Also:

None
Return to the List of data structures.

AEEArc

Description:

This structure defines the circular arc data type.

Definition:

```
typedef struct _arc
        {
        int16 cx, cy;
        int16 r;
        int16 startAngle;
        int16 arcAngle;
        } AEEArc;
```

Members:

cx	X coordinate of the center of the reference circle
cy	Y coordinate of the center of the reference circle
r	Radius of the reference circle
startAngle	The angle, in degrees, from which the circular arc begins
arcAngle	The angle of the circular arc, in degrees

Comments:

The first 3 fields are identical to that of AEECircle. This makes it convenient to explicitly cast an **AEEArc** structure to AEECircle when needed.

See Also:

AEECircle
IGRAPHICS_DrawArc()
IGRAPHICS_Translate()
Return to the List of data structures.

AEECallback

Description:

This structure specifies the data and functions for a callback registered with the ISHELL_Resume() function.

Definition:

```
typedef struct _AEECallback AEECallback; struct _AEECallback
        {
        AEECallback * pNext;
        void * pmc;
        PFNCBCANCEL pfnCancel;
        void * pCancelData;
        PFNNOTIFY pfnNotify;
        void * pNotifyData;
        void * pReserved;
        };
```

Members:

pNext	Reserved and the caller must not modify this member
pmc	Reserved and the caller must not modify this member
pfnCancel	Pointer to function called by the callback handler if this callback is cancelled. The caller must set this pointer to NULL.
pCancelData	Data passed to **pfnCancel**. The caller must not modify this member.
pfnNotify	This is the callback function that is invoked by AEE. The caller must set this pointer to the function to be called by the AEE callback handler.
pNotifyData	Data to be passed to **pfnNotify**
pfnNotify.	The caller must set this pointer to the data that must be passed to the **pfnNotify** function.
pReserved	Reserved and this member will be used by the callback handler

Comments:

None

See Also:

None
Return to the List of data structures.

AEECircle

Description:

This structure defines the circle data type.

Definition:

```
typedef struct _circle
        {
        int16 cx, cy;
        int16 r;
        } AEECircle;
```

Members:

cx	X coordinate of the circle's center
cy	Y coordinate of the circle's center
r	Radius of the circle, in number of pixels

Comments:

None

See Also:

IGRAPHICS_DrawCircle()
IGRAPHICS_Translate().
Return to the List of data structures.

AEEClip

Description:

This structure defines the dimensions of a clipping shape that is used to restrict the region in which an IGraphics drawing operations takes effect.

Definition:

```
typedef struct _clipshape
        {
        AEEClipShape type;
        union {
        AEERect rect;
        AEECircle circle;
        AEEEllipse ellipse;
        AEEPie pie;
        AEETriangle triangle;
        } shape;
        } AEEClip;
```

Members:

type	Type of clipping shape
shape	Union of clipping shape
shape.rect	Rectangular clipping shape
shape.circle	Circular clipping shape
shape.ellipse	Ellipse as the clipping shape
shape.pie	Circular pie as the clipping shape
shape.triangle	Triangular clipping shape

Comments:

Shape is a UNION type. Only one of the shapes is effective at any moment. The program has to check the type and then access shape for the corresponding clipping shape.

See Also:

AEEClipShape.
Return to the List of data structures.

AEEClipShape

Description:

This **ENUM** specifies shape types for clipping region.

Definition:

```
typedef enum
        {
        CLIPPING_NONE,
        CLIPPING_RECT,
        CLIPPING_CIRCLE,
        CLIPPING_ELLIPSE
        CLIPPING_PIE,
        CLIPPING_TRIANGLE,
        CLIPPING_POLYGON
        } AEEClipShape;
```

Members:

CLIPPING_NONE	No clipping shape is specified, the display window is used as default
CLIPPING_RECT	A rectangular shape
CLIPPING_CIRCLE	A circular shape
CLIPPING_ELLIPSE	A ellipse
CLIPPING_PIE	A circular pie
CLIPPING_TRIANGLE	A triangular shape
CLIPPING_POLYGON	An arbitrary polygon

Comments:

The default clipping shape is **CLIPPING_RECT** and the display window is the default clipping region. The program can change the clipping region by calling IGRAPHICS_SetClip().

See Also:

AEEClip,
AEERect,
AEECircle,

AEEEllipse,
AEEPie,
AEETriangle,
AEEPolygon,
IGRAPHICS_SetClip()
Return to the List of data structures.

AEEClrItem

Description:

This ENUM specifies color types for active drawing as well as system colors.

Definition:

```
typedef enum
        {
        CLR_USER_TEXT=1,
        CLR_USER_BACKGROUND,
        CLR_USER_LINE,
        CLR_SYS_TITLE,
        CLR_SYS_TITLE_TEXT,
        CLR_SYS_ITEM,
        CLR_SYS_ITEM_TEXT,
        CLR_SYS_ITEM_SEL,
        CLR_SYS_ITEM_SEL_TEXT,
        CLR_SYS_WIN,
        CLR_SYS_FRAME_HI,
        CLR_SYS_FRAME_LO,
        CLR_SYS_LT_SHADOW,
        CLR_SYS_DK_SHADOW,
        CLR_SYS_SCROLLBAR,
        CLR_SYS_SCROLLBAR_FILL,
        CLR_SYS_LAST
        } AEEClrItem;
```

Members:

CLR_USER_TEXT	Active Text Color
CLR_USER_BACKGROUND	Active Background color
CLR_USER_LINE	Active Line color (frames, and other items)
CLR_SYS_TITLE	Title Background
CLR_SYS_TITLE_TEXT	Title Text

CLR_SYS_ITEM	ITEM (SoftKey, menu, button) Background
CLR_SYS_ITEM_TEXT	ITEM Text
CLR_SYS_ITEM_SEL	ITEM Selected Background
CLR_SYS_ITEM_SEL_TEXT	ITEM Selected Text
CLR_SYS_WIN	Standard Window Background
CLR_SYS_FRAME_HI	Frame Highlight color (usually white)
CLR_SYS_FRAME_LO	Frame solid color (usually black)
CLR_SYS_LT_SHADOW	Shadow color (usually light grey)
CLR_SYS_DK_SHADOW	Shadow color (usually dark grey)
CLR_SYS_SCROLLBAR	Scroll Bars: Background
CLR_SYS_SCROLLBAR_FILL	Scroll Bars: Filled area

Comments:

Only the CLR_USER_TEXT, CLR_USER_BACKGROUND and CLR_USER_LINE colors can be changed by the user via IDISPLAY_SetColor().

See Also:

IDISPLAY_SetColor()
Return to the List of data structures.

AEEDBField

Description:

This structure defines fields in a record. This structure is used to retrieve and update database fields.

Definition:

```
typedef struct
        {
        AEEDBFieldType fType;
        AEEDBFieldName fName;
        uint16 wDataLen;
        void * pBuffer;
        } AEEDBField;
```

Members:

fType	Database field type (See documentation)
fName	Database field Name (See documentation)
wDataLen	Data Length (excluding header) of the field
pBuffer	Pointer to buffer containing the field data

Comments:

None

See Also:

None
Return to the List of data structures.

AEEDBFieldName

Description:

AEEDBFieldName is used by IDBRECORD_NextField() and IDBRECORD_GetField() to return the field name of the field.

Definition:

```
typedef enum
        {
        AEEDBFIELD_NONE,
        AEEDBFIELD_FULLNAME,
        AEEDBFIELD_LASTNAME,
        AEEDBFIELD_FIRSTNAME,
        AEEDBFIELD_HOME_PHONE,
        AEEDBFIELD_WORK_PHONE,
        AEEDBFIELD_MOBILE_PHONE,
        AEEDBFIELD_FAX,
        AEEDBFIELD_ADDRESS,
        AEEDBFIELD_EMAIL,
        AEEDBFIELD_URL,
        AEEDBFIELD_DATE_TIME,
        AEEDBFIELD_CATEGORY,
        AEEDBFIELD_ALARM,
        AEEDBFIELD_PREF_ID,
        AEEDBFIELD_PREF_VER,
        AEEDBFIELD_PREF_DATA,
        AEEDBFIELD_TITLE,
        AEEDBFIELD_TEXT
        } AEEDBFieldName;
```

Members:

AEEDBFIELD_FULLNAME	Field contains a full name
AEEDBFIELD_LASTNAME	Field contains a last name

AEEDBFIELD_FIRSTNAME	Field contains a first name
AEEDBFIELD_HOME_PHONE	Field contains a home phone number
AEEDBFIELD_WORK_PHONE	Field contains a work phone number
AEEDBFIELD_MOBILE_PHONE	Field contains a mobile phone number
AEEDBFIELD_FAX	Field contains a fax number
AEEDBFIELD_ADDRESS	Field contains a mailing address
AEEDBFIELD_EMAIL	Field contains an email address
AEEDBFIELD_URL	Field contains a URL
AEEDBFIELD_DATE_TIME	Field contains date and time
AEEDBFIELD_CATEGORY	Field contains a category specification
AEEDBFIELD_ALARM	Field contains an alarm
AEEDBFIELD_PREF_ID	Field contains a Preference ID
AEEDBFIELD_PREF_VER	Field contains a Preference Version Number
AEEDBFIELD_PREF_DATA	Field contains Preference Data (such as user preference)
AEEDBFIELD_TITLE	Field contains a Title
AEEDBFIELD_TEXT	Field contains text

Comments:

None

See Also:

None
Return to the List of data structures.

AEEDBFieldType

Description:

AEEDBFieldType is used by IDBRECORD_GetField() to return the field type of the specified field.

Definition:

```
typedef enum
        {
        AEEDB_FT_NONE,
        AEEDB_FT_BYTE,
        AEEDB_FT_WORD,
        AEEDB_FT_DWORD,
        AEEDB_FT_STRING,
        AEEDB_FT_BINARY,
        AEEDB_FT_PHONE,
        AEEDB_FT_BITMAP,
        AEEDB_FT_MAX
        } AEEDBFieldType;
```

Members:

AEEDB_FT_BYTE	Field contains an 8 bit value
AEEDB_FT_WORD	Field contains a 16 bit value
AEEDB_FT_DWORD	Field contains a 32 bit value
AEEDB_FT_STRING	Field contains an `AECHAR` array (0 terminated)
AEEDB_FT_BINARY	Field contains a Binary value
AEEDB_FT_PHONE	Field contains a Phone number
AEEDB_FT_BITMAP	Field contains a bitmap in .BMP format

Comments:

None

See Also:

None

Return to the List of data structures.

AEEDeviceInfo

Description:

This structure contains mobile device information requested in ISHELL_GetDeviceInfo().

Definition:

```
typedef struct
        {
        uint16 cxScreen;
        uint16 cyScreen;
        uint16 cxAltScreen;
        uint16 cyAltScreen;
        uint16 cxScrollBar;
        uint16 wEncoding;
        uint16 unused3;
        uint16 nColorDepth;
        EmptyEnum unused4;
        uint32 unused5;
        uint32 dwRAM;
        int bAltDisplay:1;
        int bFlip:1;
        int bVibrator:1;
        int bExtSpeaker:1;
        int bVR:1;
        int bPosLoc:1;
        int bMIDI:1;
        int bCMX:1;
        uint32 dwPromptProps;
        uint16 wKeyCloseApp;
        uint16 wKeyCloseAllApps;
        uint32 dwLang;
        uint16 wStructSize;
        uint32 dwNetLinger;
```

```
uint32 dwSleepDefer;
uint16 wMaxPath;
} AEEDeviceInfo;
```

Members:

cxScreen	Physical screen size (pixels)
cyScreen	Physical screen size (pixels)
cxAltScreen	Physical screen size of 2nd display
cyAltScreen	Physical screen size of 2nd display
cxScrollBar	Width of standard scroll bars
wEncoding	Character set encoding (UNICODE, S_JIS, KSC5601, and other items.)
nColorDepth	Color Depth (1 = mono, 2 = grey, and so forth)
dwRAM	Total RAM installed (RAM)
bAltDisplay	Device has an alternate display (Pager)
bFlip	Device is a flip-phone
bVibrator	Vibrator installed
bExtSpeaker	External speaker installed
bVR	Voice recognition supported
bPosLoc	Position location supported
bMIDI	MIDI file formats supported
bCMX	CMX audio supported
dwPromptProps	Default prompt properties
wKeyCloseApp	Key to close current app
wKeyCloseAllApps	Key to close all apps (AVK_END is default)
dwLang	ISO defined language ID
dwRAM	Total RAM installed (RAM)
bAltDisplay	Device has an alternate display (Pager)
bFlip	Device is a flip-phone
bVibrator	Vibrator installed
bExtSpeaker	External speaker installed
bVR	Voice recognition supported
bPosLoc	Position location supported
bMIDI	MIDI file formats supported

bCMX	CMX audio supported
dwPromptProps	Default prompt properties
wKeyCloseApp	Key to close current app
wKeyCloseAllApps	Key to close all apps (AVK_END is default)
dwLang	ISO defined language ID
wStructSize	Size of this structure.
dwNetLinger	Network PPP linger time
dwSleepDefer	Active non-sleep
wMaxPath	Max length of file path

Comments:

In order to use dwNetLinger, dwSleepDefer and wMaxPath fields, you MUST fill-in the wStructSize element of the structure before passing this to the GetDeviceInfo call.

See Also:

None
Return to the List of data structures.

AEEDNSResult

Description:

This structure holds the result of an INETMGR_GetHostByName() operation.

Definition:

```
typedef struct
        {
        int nResult;
        INAddr addrs[AEEDNSMAXADDRS];
        } AEENetStats;
```

Members:

nResult (if 1..4) :Number of addresses retrieved
nResult (otherwise) :Error code (see INETMGR_GetHostByName())
addrs[AEEDNSMAXADDRS] :IP addresses

Comments:

None

See Also:

INETMGR_GetHostByName()
Return to the List of data structures.

AEEEllipse

Description:

This structure defines the ellipse data type.

Definition:

```
typedef struct _ellipse
        {
        int16 cx, cy;
        int16 wx;
        int16 wy;
        } AEEEllipse;
```

Members:

cx	X coordinate of the ellipse's center
cy	Y coordinate of the ellipse's center
wx	Semi-axis length of the ellipse along x-axis
wy	Semi-axis length of the ellipse along y-axis

Comments:

All ellipses are axis-aligned. However the major axis can be either aligned with x-axis or y-axis.

See Also:

IGRAPHICS_DrawEllipse()
IGRAPHICS_Translate()
Return to the List of data structures.

AEEFrameType

Description:

This ENUM specifies the various frame types supported by the IDISPLAY_DrawFrame() function. Frame types are also used when specifying menu item styles with IMENUCTL_SetStyle function.

Definition:

```
typedef enum
        {
        AEE_FT_NONE,
        AEE_FT_EMPTY,
        AEE_FT_3D_EMPTY,
        AEE_FT_RAISED,
        AEE_FT_LOWERED,
        AEE_FT_BOX,
        AEE_FT_INDENT,
        AEE_FT_TAB_BOTTOM_SEL,
        AEE_FT_TAB_BOTTOM,
        AEE_FT_TAB_TOP_SEL,
        AEE_FT_TAB_TOP
        } AEEFrameType;
```

Members:

AEE_FT_NONE	No Frame
AEE_FT_EMPTY	1 Pixel Offset: No Frame Drawn
AEE_FT_3D_EMPTY	2 Pixel Offset: No Frame Drawn
AEE_FT_RAISED	3D 2 Pixel Raised Frame
AEE_FT_LOWERED	3D 2 Pixel Lowered Frame
AEE_FT_BOX	1 Pixel Box
AEE_FT_INDENT	3D 1 Pixel Lowered Frame
AEE_FT_TAB_BOTTOM_SEL	3D Bottom Tab (selected)
AEE_FT_TAB_BOTTOM	3D Bottom Tab

AEE_FT_TAB_TOP_SEL	3D Top Tab (selected)
AEE_FT_TAB_TOP	3D Top Tab

Comments:

None

See Also:

AEEItemStyle
Return to the List of data structures.

AEEFont

Description:

This ENUM specifies the logical font type used in IDisplay text drawing operations.

Definition:

```
typedef enum
        {
        AEE_FONT_NORMAL=0x8000,
        AEE_FONT_BOLD,
        AEE_FONT_LARGE,
        AEE_FONT_TOTAL
        } AEEFont;
```

Members:

AEE_FONT_NORMAL	The normal font supported by the mobile device
AEE_FONT_BOLD	Bold type font supported by the mobile device
AEE_FONT_LARGE	Large type font supported by the mobile device
AEE_FONT_TOTAL	Reserved

Comments:

None

See Also:

None
Return to the List of data structures.

AEEHandlerType

Description:

This **ENUM** specifies the handler type in ISHELL_RegisterHandler(), ISHELL_GetHandler(), and so forth

Definition:

```
typedef enum
        { HTYPE_VIEWER,
        HTYPE_SOUND
        } AEEHandlerType;
```

Members:

HTYPE_VIEWER	To specify that the handler provides image viewing services
HTYPE_SOUND	To specify that the handler provides sound services
HTYPE_BROWSE	To specify a handlerapplet that supports either schemes (http, mailto, and so forth) or file extensions (.gif, .htm, and so forth)

Comments:

None

See Also:

None
Return to the List of data structures.

AEEImageInfo

Description:

This structure gets the information about an image

Definition:

```
typedef struct _AEEImageInfo
        {
        uint16 cx;
        uint16 cy;
        uint16 nColors;
        boolean bAnimated;
        uint16 cxFrame;
        } AEEImageInfo;
```

Members:

cx	The width of the image (in pixels)
cy	The height of the image (in pixels)
nColors	The number of colors in the image
bAnimated	TRUE, if the image contains animation
cxFrame	If the image is divided into frames, this member indicates the width of each frame

Comments:

None

See Also:

None
Return to the List of data structures.

AEEItemStyle

Description:

This structure specifies the item style.

Definition:

```
typedef struct
        {
        AEEFrameType ft;
        uint16 xOffset;
        uint16 yOffset;
        AEERasterOp roImage;
        } AEEItemStyle;
```

Members:

ft	Frame type of item
xOffset	X padding inside item (does not include frame size)
yOffset	Y padding inside item (does not include frame size)
roImage	Raster operation for drawing images inside the item

Comments:

None

See Also:

None
Return to the List of data structures.

AEEItemType

Description:

This ENUM specifies the item type whose style is requested using the ISHELL_GetItemStyle().

Definition:

```
typedef enum
        {
        AEE_IT_MENU,
        AEE_IT_SOFTKEY,
        AEE_IT_ICONVIEW
        } AEEItemType;
```

Members:

AEE_IT_MENU	Menu Item type
AEE_IT_SOFTKEY	Soft key item type
AEE_IT_ICONVIEW	Icon View Item type

Comments:

None

See Also:

None
Return to the List of data structures.

AEELine

Description:

This structure defines the line segment data type.

Definition:

```
typedef struct _line
        {
        int16 sx, sy;
        int16 ex, ey;
        } AEELine;
```

Members:

sx	X coordinate of the starting point of the line segment
sy	Y coordinate of the starting point of the line segment
ex	X coordinate of the ending point of the line segment
ey	Y coordinate of the ending point of the line segment

Comments:

Mathematically a line is infinitely long. So **AEELine** defines a line segment from **(sx, sy)** to **(ex, ey),** instead of a mathematical line. The line segment includes both end points.

See Also:

IGRAPHICS_DrawLine()
Return to the List of data structures.

AEEMenuColors

Description:

AEEMenuColors is used to specify overriding color values for various menu control object elements.

Definition:

```
typedef struct
        {
        uint16 wMask;
        RGBVALcBack;
        RGBVALcText;
        RGBVALcSelBack;
        RGBVALcSelText;
        RGBVALcFrame;
        RGBVALcScrollbar;
        RGBVALcScrollbarFill;
        RGBVALcTitle;
        RGBVALcTitleText;
        } AEEMenuColors;
```

Members:

wMask	Mask of bits to pay attention to in this struct
cBack	Background color of unselected items
cText	Text color for unselected items and Arrows
cSelBack	Background color for selected items
cSelText	Text color for selected items
cFrame	Simple frame color
cScrollbar	Scrollbar frame color
cScrollbarFill	Scrollbar fill color
cTitle	Background of title text
cTitleText	Color of title text

Comments:

None

See Also:

IMENUCTL_SetColors()
AEEMenuColorsMask
Return to the List of data structures.

AEEMenuColorsMask

Description:

Set of masks to indicate the item whose color needs to be changed.

Definition:

MC_BACK	Unselected item background
MC_TEXT	Unselected item text
MC_SEL_BACK	Selected item background
MC_SEL_TEXT	Selected item text
MC_FRAME	Simple frame color
MC_SCROLLBAR	Scrollbar frame color
MC_SCROLLBAR_FILL	Scrollbar fill color
MC_TITLE	Title background color. This mask is supported only for list controls.
MC_TITLE_TEXT	Title text color. This mask is supported only for list controls.

Members:

None

Comments:

None.

See Also:

AEEMenuColors
Return to the List of data structures.

AEENetStats

Description:

This structure describes the status of the network connection (the state of the PPP link).

Definition:

```
typedef struct
        {
        uint32 dwOpenTime;
        uint32 dwActiveTime;
        uint32 dwBytes;
        uint32 dwRate;
        uint32 dwTotalOpenTime;
        uint32 dwTotalActiveTime;
        uint32 dwTotalBytes;
        uint32 dwTotalRate;
        } AEENetStats;
```

Members:

dwOpenTime	Time in seconds since PPP connection established
dwActiveTime	Time in seconds the PPP connection was actually active
dwBytes	Total bytes sent on connection
dwRate	Rate of transfer (bytes / sec)
dwTotalOpenTime	Time in seconds for all open PPP sessions
dwTotalActiveTime	Time in seconds for all active PPP sessions
dwTotalBytes	Total bytes sent (all connections)
dwTotalRate	Total rate (all connections)

Comments:

None

See Also:

None
Return to the List of data structures.

AEENotify

Description:

A pointer to this structure is passed as dwParam when EVT_NOTIFY event is sent to an app. An app receives this event as part of the notification(s) that it has registered for.

Definition:

```
typedef struct
        {
        AEECLSID cls;
        INotifier * pNotifier;
        uint32 dwMask;
        void * pData;
        AEENotifyStatus st;
        } AEENotify;
```

Members:

cls	Notifier Class
pNotifier	Notifier Object that issued the Notify
dwMask	Mask of bits that occurred
pData	Notification-specific data
st	Indicates to IShell if the app processed the notificaiot

Comments:

None

See Also:

ISHELL_RegisterNotify()
ISHELL_Notify()
AEENotifyStatus
Return to the List of data structures.

AEENotifyStatus

Description:

This enumerated type defines the notification status values that are returned to the shell by an applet that receives a notification. The applet returns the status of its processing of the notification by setting the st member of the AEENotify structure it is passed along with the EVT_NOTIFY event.

Definition:

```
typedef enum
        {
         NSTAT_PROCESSED,
         NSTAT_IGNORED,
         NSTAT_STOP
        } AEENotifyStatus;
```

Members:

NSTAT_PROCESSED	The applet successfully processed the notification
NSTAT_IGNORED	The applet ignored the notification
NSTAT_STOP	The applet processed the notification, and the notification can not be sent to any other applets that have registered to be notified of this event.

Comments:

None.

See Also:

ISHELL_RegisterNotify()
ISHELL_Notify()
AEENotify
Return to the List of data structures.

AEEPaintMode

Description:

This ENUM specifies the raster operation types for drawing.

Definition:

```
typedef enum
        {
        AEE_PAINT_COPY,
        AEE_PAINT_XOR
        } AEEPaintMode;
```

Members:

AEE_PAINT_COPY	Copy raster operation
AEE_PAINT_XOR	XOR raster operation

Comments:

When the paint mode is set to **AEE_PAINT_COPY**, new content shall overwrite the old content in the display buffer. When the paint mode is set to **AEE_PAINT_XOR**, the "xor-ed" result of the new content and the old is written into the display buffer. IGraphics's paint mode is set to **AEE_PAINT_COPY** as default. The program can change the paint mode by calling IGRAPHICS_SetPaintMode().

See Also:

IGRAPHICS_SetPaintMode()
Return to the List of data structures.

AEEPie

Description:

This structure defines the circular pie data type.

Definition:

```
typedef struct _pie
        {
        int16 cx, cy;
        int16 r;
        int16 startAngle;
        int16 arcAngle;
        } AEEPie;
```

Members:

cx	X coordinate of the center of the reference circle
cy	Y coordinate of the center of the reference circle
r	Radius of the reference circle
startAngle	The angle, in degrees, from which the circular arc begins
arcAngle	The angle of the circular arc, in degrees

Comments:

The parameters in this data structure are identical to that of AEEArc. This makes it convenient to explicitly cast an AEEPie type to AEEArc if necessary.

See Also:

AEEArc
IGRAPHICS_DrawPie(), IGRAPHICS_Translate()
Return to the List of data structures.

AEEPoint

Description:

This structure defines the point data type.

Definition:

```
typedef struct _point
        {
        int16 x, y;
        } AEEPoint;
```

Members:

x	X-coordinate
y	Y-coordinate

Comments:

None

See Also:

IGRAPHICS_DrawPoint()
Return to the List of data structures.

AEEPolygon

Description:

This structure defines the polygon data type.

Definition:

```
typedef struct _polygon
        {
        int16 len;
        AEEPoint * points;
        } AEEPolygon;
```

Members:

len	Number of vertices of the polygon
points	Array of vertices of the polygon

Comments:

None

See Also:

IGRAPHICS_DrawPolygon()
IGRAPHICS_Translate()
Return to the List of data structures.

AEEPolyline

Description:

This structure defines the polyline data type. Polyline is a sequence of connected line segments.

Definition:

```
typedef struct _polyline
        {
        int16 len;
        AEEPoint * points;
        } AEEPolyline;
```

Members:

len	Number of points in the polyline
points	Array of points in the polyline

Comments:

This data type has exactly the same structure as AEEPolygon. This enables convenient casting between **AEEPolyline** and AEEPolygon.

See Also:

IGRAPHICS_DrawPolyline()
IGRAPHICS_Translate()
Return to the List of data structures.

AEEPosAccuracy

Description:

This data structure describes the Position Location Information Accuracy

Definition:

```
typedef enum
        {
        AEE_ACCURACY_LOW,
        AEE_ACCURACY_MED,
        AEE_ACCURACY_HIGH
        } AEEPosAccuracy;
```

Members:

None.

Comments:

The position location information precision is directly related to the time it will take to satisfy the ISHELL_GetPosition() request.

See Also:

ISHELL_GetPosition()
Return to the List of data structures.

AEEPositionInfo

Description:

This data structure describes thePosition Location Information

Definition:

```
typedef struct
        {
        int32 dwLat;
        int32 dwLon;
        uint32dwTimeStamp;
        } AEEPositionInfo;
```

Members:

dwLat	Latitude
dwLon	Longitude
dwTimeStamp	Time Stamp

Comments:

None.

See Also:

ISHELL_GetPosition(),
Return to the List of data structures.

AEEPromptInfo

Description:

This structure specifies the prompt information used by the ISHELL_Prompt() function.

Definition:

```
typedef struct
        {
        const char * pszRes;
        const AECHAR * pTitle;
        const AECHAR * pText;
        uint16 wTitleID;
        uint16 wTextID;
        uint16 wDefBtn;
        const uint16 * pBtnIDs;
        uint32 dwProps;
        AEEFont fntTitle;
        AEEFont fntText;
        uint32 dwTimeout;
        } AEEPromptInfo;
```

Members:

pszRes	Resource file name used for prompt title and text information
pTitle	If **pszRes** is not specified, the pointer to wide string containing the prompt title
pText	If **pszRes** is not specified, the pointer to wide string containing the prompt text
wTitleID	If **pszRes** is specified, the ID of the title string in the resource file
wTextID	If **pszRes** is specified, the ID of the text string in the resource file
wDefBtn	The ID of the default (or selected button)
pBtnIDs	Pointer to array of button IDs
dwProps	Property of the prompt (See the properties for the IStatic Interface)
fntTitle	Title font

fntText	Text font
dwTimeout	Timeout for the prompt. If set to 0 (zero), the prompt has no timeout

Comments:

If **pszRes ,pTitle,** and **pText** are non-NULL, the prompt title and text are read from the resource file.

See Also:

None
Return to the List of data structures.

AEERasterOp

Description:

This ENUM specifies the raster operation for bit-block transfers of bitmaps, and drawing images on the screen with the functions in the IImage Interface.

Definition:

```
typedef enum
        {
        AEE_RO_OR,
        AEE_RO_XOR,
        AEE_RO_COPY,
        AEE_RO_NOT,
        AEE_RO_MASK,
        AEE_RO_MERGENOT,
        AEE_RO_MASKNOT,
        AEE_RO_TRANSPARENT,
        AEE_RO_TOTAL
        } AEERasterOp;
```

Members:

AEE_RO_OR	SRC .OR. DST
AEE_RO_XOR	SRC .XOR. DST
AEE_RO_COPY	DST = SRC
AEE_RO_NOT	DST = (!SRC)
AEE_RO_MASK	Same as **AEE_RO_TRANSPARENT.** In monochrome mode it is equivalent to DST .AND. SRC
AEE_RO_MERGENOT	DST .OR. (!SRC)

AEE_RO_MASKNOT	DST .AND. (!SRC)
AEE_RO_TRANSPARENT	The SRC pixels with a certain color are transparent meaning that the corresponding DST pixels are seen through: For a monochrome device, the color is **RGB_MASK_MONO**, which is white. For a gray -scale devices, the color is **RGB_MASK_GREY** which is white For a color device, the color is **RGB_MASK_COLOR,** which is magenta.

(where SRC is the source bitmap buffer, and DST is the destination bitmap buffer)

Comments:

None

See Also:

None
Return to the List of data structures.

AEERect

Description:

AEERect is used to define a rectangle used by various Display, Graphics, Text Control, and other helper functions.

Definition:

```
typedef struct
        {
        int16 x,y;
        int16 dx, dy;
        } AEERect;
```

Members:

x	The horizontal coordinate for the beginning (top left corner) of the rectangle
y	The vertical coordinate for the beginning (top left corner) of the rectangle
dx	The width of the rectangle (in pixels)
dy	The height of the rectangle (in pixels)

Comments:

None

See Also:

None
Return to the List of data structures.

AEESoundAPath

Description:

AEESoundAPath is used to indicate whether or not ISound Interface can transmit the tone over-the-air in a call

Definition:

```
typedef enum
        {
        AEE_SOUND_APATH_LOCAL,
        AEE_SOUND_APATH_TX,
        AEE_SOUND_APATH_BOTH,
        AEE_SOUND_APATH_MUTE,
        AEE_SOUND_APATH_LAST
        } AEESoundAPath;
```

Members:

AEE_SOUND_APATH_LOCAL	DTMF on local audio
AEE_SOUND_APATH_TX	Transmit DTMFs
AEE_SOUND_APATH_BOTH	Transmit and sound DTMFs locally
AEE_SOUND_APATH_MUTE	Mute DTMFs
AEE_SOUND_APATH_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundCmd

Description:

AEESoundCmd specifies the callback type used by ISound to send events and data to the application

Definition:

```
typedef enum
        {
        AEE_SOUND_STATUS_CB,
        AEE_SOUND_VOLUME_CB
        } AEESoundCmd;
```

Members:

AEE_SOUND_STATUS_CB	ISound status callback
AEE_SOUND_VOLUME_CB	Get volume callback

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundCmdData

Description:

AEESoundCmdData specifies the data sent through callback to application.

Definition:

```
typedef union
        {
        uint16 wVolume;
        uint16 wPlayIndex;
        } AEESoundCmdData;
```

Members:

wVolume	Volume sent through volume callback
wPlayIndex	Current tone to be played. This is sent through status callback for **PlayToneList**

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundDevice

Description:

AEESoundDevice specifies the device selected and used by ISound Interface.

Definition:

```
typedef enum
        {
        AEE_SOUND_DEVICE_UNKNOWN,
        AEE_SOUND_DEVICE_HANDSET,
        AEE_SOUND_DEVICE_HFK,
        AEE_SOUND_DEVICE_HEADSET,
        AEE_SOUND_DEVICE_AHFK,
        AEE_SOUND_DEVICE_SDAC,
        AEE_SOUND_DEVICE_TTY_HFK,
        AEE_SOUND_DEVICE_TTY_HEADSET,
        AEE_SOUND_DEVICE_CURRENT,
        AEE_SOUND_DEVICE_LAST
        } AEESoundDevice;
```

Members:

AEE_SOUND_DEVICE_UNKNOWN	Unknown device
AEE_SOUND_DEVICE_HANDSET	Handset
AEE_SOUND_DEVICE_HFK	Hands Free Kit (HFK)
AEE_SOUND_DEVICE_HEADSET	Headset
AEE_SOUND_DEVICE_AHFK	Analog HFK
AEE_SOUND_DEVICE_SDAC	Stereo DAC
AEE_SOUND_DEVICE_TTY_HFK	TTY HFK
AEE_SOUND_DEVICE_TTY_HEADSET	TTY Headset
AEE_SOUND_DEVICE_CURRENT	Currently selected device
AEE_SOUND_DEVICE_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundInfo

Description:

AEESoundInfo specifies the ISound attributes that are used by ISound for all its operations.

Definition:

```
typedef struct
        {
        AEESoundDevice eDevice;
        AEESoundMethod eMethod;
        AEESoundAPath eAPath;
        AEESoundMuteCtl eEarMuteCtl;
        AEESoundMuteCtl eMicMuteCtl;
        } AEESoundInfo;
```

Members:

eDevice	Device used
eMethod	Method used
eAPath	Over-the-air audio path
eEarMuteCtl	Earpiece mute control
eMicMuteCtl	Microphone mute control

Comments:

None

See Also:

AEESoundDevice, AEESoundMethod,
AEESoundAPath, AEESoundMuteCtl
Return to the List of data structures.

AEESoundMethod

Description:

AEESoundMethod specifies the method used by ISound Interface.

Definition:

```
typedef enum
        {
        AEE_SOUND_METHOD_UNKNOWN,
        AEE_SOUND_METHOD_VOICE,
        AEE_SOUND_METHOD_BEEP,
        AEE_SOUND_METHOD_MESSAGE,
        AEE_SOUND_METHOD_RING,
        AEE_SOUND_METHOD_CLICK,
        AEE_SOUND_METHOD_MIDI,
        AEE_SOUND_METHOD_AUX,
        AEE_SOUND_METHOD_LAST
        } AEESoundMethod;
```

Members:

AEE_SOUND_METHOD_UNKNOWN	Unknown method
AEE_SOUND_METHOD_VOICE	Use the device's voice generator
AEE_SOUND_METHOD_BEEP	Use the device's **keybeep** generator
AEE_SOUND_METHOD_MESSAGE	Use the device's **keybeep** generator
AEE_SOUND_METHOD_RING	Use the device's ring generator
AEE_SOUND_METHOD_CLICK	Use the device's click generator
AEE_SOUND_METHOD_MIDI	Use the device's Midi generator
AEE_SOUND_METHOD_AUX	Use the device's auxiliary generator if avail
AEE_SOUND_METHOD_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundMuteCtl

Description:

AEESoundMuteCtl is used to control ear piece and microphone muting.

Definition:

```
typedef enum
        {
        AEE_SOUND_MUTECTL_UNMUTED,
        AEE_SOUND_MUTECTL_MUTED,
        AEE_SOUND_MUTECTL_LAST
        } AEESoundMuteCtl;
```

Members:

AEE_SOUND_MUTECTL_UNMUTED	Audio path is not muted
AEE_SOUND_MUTECTL_MUTED	Audio path is muted
AEE_SOUND_MUTECTL_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerAudioSpec

Description:

AEESoundPlayerAudioSpec indicates the audio specifications. It is used with the AEESOUNDPLAYER_AUDIO_SPEC playback callback.

Definition:

```
typedef union
        {
        AEESoundPlayerFile fType;
        AEESoundPlayerMIDISpec MIDISpec;
        AEESoundPlayerMP3Spec MP3Spec;
        } AEESoundPlayerAudioSpec;
```

Members:

fType	Audio File Type
MIDISpec	MIDI specifications if file type is MIDI
MP3Spec	MP3 specifications if file type is MP3

Comments:

None

See Also:

AEESoundPlayerFile,
AEESoundPlayerMIDISpec,
AEESoundPlayerMP3Spec
Return to the List of data structures

AEESoundPlayerCmd

Description:

AEESoundPlayerCmd specifies the callback type used by ISoundPlayer to send events and data to application.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_STATUS_CB,
        AEE_SOUNDPLAYER_PLAY_CB,
        AEE_SOUNDPLAYER_TIME_CB,
        AEE_SOUNDPLAYER_SOUND_CB,
        AEE_SOUNDPLAYER_VOLUME_CB
        } AEESoundPlayerCmd;
```

Members:

AEE_SOUNDPLAYER_STATUS_CB	SoundPlayer status callback
AEE_SOUNDPLAYER_PLAY_CB	Playback callback
AEE_SOUNDPLAYER_TIME_CB	Get time callback
AEE_SOUNDPLAYER_SOUND_CB	Sound callback
AEE_SOUNDPLAYER_VOLUME_CB	Get volume callback

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerCmdData

Description:

AEESoundPlayerCmdData specifies the data sent through callback to application.

Definition:

```
typedef union
        {
        uint32 dwElapsedTime;
        uint32 dwTotalTime;
        uint32 dwTempo;
        uint32 dwTune;
        uint32 dwPan;
        uint16 wVolume;
        AEESoundPlayerAudioSpec spSpec;
        } AEESoundPlayerCmdData;
```

Members:

dwElapsedTime	Elapsed time sent through playback callback for Pause, Resume, Rewind, FastForward
dwTotalTime	Total Time sent through get time callback
dwTempo	Tempo factor sent through playback callback for **SetTempo**
dwTune	Tune factor sent through playback callback for **SetTune**
dwPan	Not used
wVolume	Indicates volume of the device, method pair. It ranges from 0 (zero) to **AEE_MAX_VOLUME**
spSpec	Audio specs sent through playback callback for Play

Comments:

None

See Also:

AEESoundPlayerAudioSpec
Return to the List of data structures

AEESoundPlayerFile

Description:

AEESoundPlayerFile indicates the type of file being played.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_FILE_UNKNOWN,
        AEE_SOUNDPLAYER_FILE_MIDI,
        AEE_SOUNDPLAYER_FILE_MP3,
        AEE_SOUNDPLAYER_FILE_LAST
        } AEESoundPlayerFile;
```

Members:

AEE_SOUNDPLAYER_FILE_UNKNOWN	Invalid type
AEE_SOUNDPLAYER_FILE_MIDI	MIDI
AEE_SOUNDPLAYER_FILE_MP3	MP3
AEE_SOUNDPLAYER_FILE_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerInput

Description:

AEESoundPlayerInput indicates the source where ISoundPlayer finds the data or file to be played.

Definition:

```
typedef enum
        {
        SDT_NONE,
        SDT_FILE,
        SDT_BUFFER,
        SDT_VOICEPROMPT
        } AEESoundPlayerInput;
```

Members:

SDT_NONE	Source unknown (reserved)
SDT_FILE	Specified data is file name
SDT_BUFFER	Specified data is raw buffer (reserved)
SDT_VOICEPROMPT	Not used

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMIDISpec

Description:

AEESoundPlayerMIDISpec indicates the audio specifications of the MIDI file type. It is used with the AEESoundPlayerAudioSpec playback callback.

Definition:

```
typedef struct
        {
        AEESoundPlayerFile fType;
        } AEESoundPlayerMIDISpec;
```

Members:

fType	Format of audio file

Comments:

None

See Also:

AEESoundPlayerFile
Return to the List of data structures.

AEESoundPlayerMP3BitRate

Description:

AEESoundPlayerMP3BitRate specifies the MP3 bit rate.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_BITRATE_FREE = 0,
        AEE_SOUNDPLAYER_MP3_BITRATE_8K = 8,
        AEE_SOUNDPLAYER_MP3_BITRATE_16K = 16,
        AEE_SOUNDPLAYER_MP3_BITRATE_24K = 24,
        AEE_SOUNDPLAYER_MP3_BITRATE_32K = 32,
        AEE_SOUNDPLAYER_MP3_BITRATE_40K = 40,
        AEE_SOUNDPLAYER_MP3_BITRATE_48K = 48,
        AEE_SOUNDPLAYER_MP3_BITRATE_56K = 56,
        AEE_SOUNDPLAYER_MP3_BITRATE_64K = 64,
        AEE_SOUNDPLAYER_MP3_BITRATE_80K = 80,
        AEE_SOUNDPLAYER_MP3_BITRATE_96K = 96,
        AEE_SOUNDPLAYER_MP3_BITRATE_112K = 112,
        AEE_SOUNDPLAYER_MP3_BITRATE_128K = 128,
        AEE_SOUNDPLAYER_MP3_BITRATE_144K = 144,
        AEE_SOUNDPLAYER_MP3_BITRATE_160K = 160,
        AEE_SOUNDPLAYER_MP3_BITRATE_176K = 176,
        AEE_SOUNDPLAYER_MP3_BITRATE_192K = 192,
        AEE_SOUNDPLAYER_MP3_BITRATE_224K = 224,
        AEE_SOUNDPLAYER_MP3_BITRATE_256K = 256,
        AEE_SOUNDPLAYER_MP3_BITRATE_288K = 288,
        AEE_SOUNDPLAYER_MP3_BITRATE_320K = 320,
        AEE_SOUNDPLAYER_MP3_BITRATE_352K = 352,
        AEE_SOUNDPLAYER_MP3_BITRATE_384K = 384,
        AEE_SOUNDPLAYER_MP3_BITRATE_416K = 416,
        AEE_SOUNDPLAYER_MP3_BITRATE_448K = 448,
```

```
    AEE_SOUNDPLAYER_MP3_BITRATE_VAR = 500,
    AEE_SOUNDPLAYER_MP3_BITRATE_UNK = 501
    } AEESoundPlayerMP3BitRate;
```

Members:

This is Free bit rate (determined by software)

AEE_SOUNDPLAYER_MP3_BITRATE_FREE

The following are Fixed bit rates

AEE_SOUNDPLAYER_MP3_BITRATE_8K
AEE_SOUNDPLAYER_MP3_BITRATE_16K
AEE_SOUNDPLAYER_MP3_BITRATE_24K
AEE_SOUNDPLAYER_MP3_BITRATE_32K
AEE_SOUNDPLAYER_MP3_BITRATE_40K
AEE_SOUNDPLAYER_MP3_BITRATE_48K
AEE_SOUNDPLAYER_MP3_BITRATE_56K
AEE_SOUNDPLAYER_MP3_BITRATE_64K
AEE_SOUNDPLAYER_MP3_BITRATE_80K
AEE_SOUNDPLAYER_MP3_BITRATE_96K
AEE_SOUNDPLAYER_MP3_BITRATE_112K
AEE_SOUNDPLAYER_MP3_BITRATE_128K
AEE_SOUNDPLAYER_MP3_BITRATE_144K
AEE_SOUNDPLAYER_MP3_BITRATE_160K
AEE_SOUNDPLAYER_MP3_BITRATE_176K
AEE_SOUNDPLAYER_MP3_BITRATE_192K
AEE_SOUNDPLAYER_MP3_BITRATE_224K
AEE_SOUNDPLAYER_MP3_BITRATE_256K
AEE_SOUNDPLAYER_MP3_BITRATE_288K
AEE_SOUNDPLAYER_MP3_BITRATE_320K
AEE_SOUNDPLAYER_MP3_BITRATE_352K
AEE_SOUNDPLAYER_MP3_BITRATE_384K
AEE_SOUNDPLAYER_MP3_BITRATE_416K
AEE_SOUNDPLAYER_MP3_BITRATE_448K

This is Variable bit rate (Changes each frame)

AEE_SOUNDPLAYER_MP3_BITRATE_VAR

Unable to determine bit-rate information

AEE_SOUNDPLAYER_MP3_BITRATE_UNK

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3Channel

Description:

AEESoundPlayerMP3Channel specifies the MP3 channel.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_CHANNEL_STEREO
        AEE_SOUNDPLAYER_MP3_CHANNEL_JOINT_STEREO
        AEE_SOUNDPLAYER_MP3_CHANNEL_DUAL
        AEE_SOUNDPLAYER_MP3_CHANNEL_SINGLE
        } AEESoundPlayerMP3Channel;
```

Members:

AEE_SOUNDPLAYER_MP3_CHANNEL_STEREO	Stereo data
AEE_SOUNDPLAYER_MP3_CHANNEL_JOINT_STEREO	Joint Stereo data
AEE_SOUNDPLAYER_MP3_CHANNEL_DUAL	Dual channel (stereo) data
AEE_SOUNDPLAYER_MP3_CHANNEL_SINGLE	Single channel (mono) data

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3Emphasis

Description:

AEESoundPlayerMP3Emphasis specifies MP3 emphasis flag.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_EMPHASIS_NONE,
        AEE_SOUNDPLAYER_MP3_EMPHASIS_50_15_MS,
        AEE_SOUNDPLAYER_MP3_EMPHASIS_RESERVED,
        AEE_SOUNDPLAYER_MP3_EMPHASIS_CCITT_J17
        } AEESoundPlayerMP3Emphasis;
```

Members:

AEE_SOUNDPLAYER_MP3_EMPHASIS_NONE	Invalid flag
AEE_SOUNDPLAYER_MP3_EMPHASIS_50_15_MS	50_15_MS
AEE_SOUNDPLAYER_MP3_EMPHASIS_RESERVED	Reserved
AEE_SOUNDPLAYER_MP3_EMPHASIS_CCITT_J17	CCITT J17

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3Extension

Description:

AEESoundPlayerMP3Extension specifies MP3 extension for layers 1-3.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_EXT_BAND_4_31 = 0,
        AEE_SOUNDPLAYER_MP3_EXT_BAND_8_31 = 1,
        AEE_SOUNDPLAYER_MP3_EXT_BAND_12_31 = 2,
        AEE_SOUNDPLAYER_MP3_EXT_BAND_16_31 = 3,
        AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_OFF_MS_OFF = 0,
        AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_ON_MS_OFF = 1,
        AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_OFF_MS_ON = 2,
        AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_ON_MS_ON = 3
        } AEESoundPlayerMP3Extension;
```

Members:

// For Layer 1 and 2 files:

AEE_SOUNDPLAYER_MP3_EXT_BAND_4_31	Channel extension info, 4-31
AEE_SOUNDPLAYER_MP3_EXT_BAND_8_31	Channel extension info, 8-31
AEE_SOUNDPLAYER_MP3_EXT_BAND_12_31	Channel extension info, 12-31
AEE_SOUNDPLAYER_MP3_EXT_BAND_16_31	Channel extension info, 16-31

// For Layer 3 files

AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_OFF_MS_OFF	Intensity stereo off, MS off
AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_ON_MS_OFF	Intensify stereo on, MS off

AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_OFF_MS_ON	Intensity stereo off, MS on
AEE_SOUNDPLAYER_MP3_EXT_INTENSITY_ON_MS_ON	Intensity stereo on, MS on

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3Layer

Description:

AEESoundPlayerMP3Layer specifies the MPEG layer information.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_LAYER_RESERVED,
        AEE_SOUNDPLAYER_LAYER_3,
        AEE_SOUNDPLAYER_LAYER_2,
        AEE_SOUNDPLAYER_LAYER_1,
        AEE_SOUNDPLAYER_LAYER_UNKNOWN
        } AEESoundPlayerMP3Layer;
```

Members:

AEE_SOUNDPLAYER_LAYER_RESERVED	Reserved
AEE_SOUNDPLAYER_LAYER_3	MPEG layer 3 compression
AEE_SOUNDPLAYER_LAYER_2	MPEG layer 2 compression
AEE_SOUNDPLAYER_LAYER_1	MPEG layer 1 compression
AEE_SOUNDPLAYER_LAYER_UNKNOWN	Unable to determine layer information

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3SampleRate

Description:

AEESoundPlayerMP3SampleRate specifies MP3 sample rate.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_UNKNOWN,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_8000 = 1,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_11025,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_12000,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_16000,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_22050,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_24000,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_32000,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_44100,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_48000,
        AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_LAST
        } AEESoundPlayerMP3SampleRate;
```

Members:

AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_UNKNOWN	Unknown rate
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_8000 8k	
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_11025	11.025k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_12000	12k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_16000	16k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_22050	22.050k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_24000	24k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_32000	32k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_44100	44.1k

AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_48000	48k
AEE_SOUNDPLAYER_MP3_SAMPLE_RATE_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerMP3Spec

Description:

AEESoundPlayerMP3Spec indicates the audio specifications of MP3 file type. It is used with the AEESOUNDPLAYER_AUDIO_SPEC playback callback.

Definition:

```
typedef struct
        {
        AEESoundPlayerFile fType;
        AEESoundPlayerMP3Version version;
        AEESoundPlayerMP3Layer layer;
        boolean crcFlag;
        AEESoundPlayerMP3BitRate bitrate;
        AEESoundPlayerMP3SampleRate sampleRate;
        AEESoundPlayerMP3Channel channel;
        AEESoundPlayerMP3Extension extension;
        boolean copyrightFlag;
        boolean originalFlag;
        AEESoundPlayerMP3Emphasis emphasis;
        char title[AEE_ID3_TAG_LENGTH];
        char artist[AEE_ID3_TAG_LENGTH];
        char album[AEE_ID3_TAG_LENGTH];
        char year[4];
        char comment[AEE_ID3_TAG_LENGTH];
        uint8 genre;
        } AEESoundPlayerMP3Spec;
```

Members:

Type	Format of audio file
version	MPEG version
layer	MPEG layer description
crcFlag	True if CRC protection

bitrate	Bit rate
sampleRate	Sampling rate
channel	Channel mode
extension	Used Only when channel is AEE_SOUNDPLAYER_MP3_CHANNEL_JOINT_STEREO
copyrightFlag	True if copyrighted
originalFlag	True if original
emphasis	Audio emphasis
title	Song title
artist	Song performer
album	Album with the song
year	Year Album released
comment	Text comment
genre	ID3 genre tag

Comments:

None

See Also:

AEESoundPlayerFile,
AEESoundPlayerMP3Version,
AEESoundPlayerMP3Layer,
AEESoundPlayerMP3BitRate,
AEESoundPlayerMP3SampleRate,
AEESoundPlayerMP3Channel,
AEESoundPlayerMP3Extension,
AEESoundPlayerMP3Emphasis
Return to the List of data structures

AEESoundPlayerMP3Version

Description:

AEESoundPlayerMP3Version specifies the MPEG version types.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_MP3_VERSION_25,
        AEE_SOUNDPLAYER_MP3_VERSION_RESERVED,
        AEE_SOUNDPLAYER_MP3_VERSION_2,
        AEE_SOUNDPLAYER_MP3_VERSION_1,
        AEE_SOUNDPLAYER_MP3_VERSION_UNKNOWN
        } AEESoundPlayerMP3Version;
```

Members:

AEE_SOUNDPLAYER_MP3_VERSION_25	MPEG version 2.5
AEE_SOUNDPLAYER_MP3_VERSION_RESERVED	Reserved
AEE_SOUNDPLAYER_MP3_VERSION_2	MPEG version 2.0
AEE_SOUNDPLAYER_MP3_VERSION_1	MPEG version 1.0
AEE_SOUNDPLAYER_MP3_VERSION_UNKNOWN	Unable to determine version

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerSource

Description:

AEESoundPlayerSource used internally by the ISoundPlayer Interface.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_SOURCE_UNKNOWN, // An invalid type
        AEE_SOUNDPLAYER_SOURCE_MEM, // Designates that data is found in the
        AEE_SOUNDPLAYER_SOURCE_FILE, // Designates that data is found in
        the // Embedded File System (EFS)
        AEE_SOUNDPLAYER_SOURCE_LAST
        } AEESoundPlayerSource;
```

Members:

AEE_SOUNDPLAYER_SOURCE_UNKNOWN	Invalid type
AEE_SOUNDPLAYER_SOURCE_MEM	Data is found in the memory
AEE_SOUNDPLAYER_SOURCE_FILE	Data is found in the Embedded File System (EFS)
AEE_SOUNDPLAYER_SOURCE_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundPlayerStatus

Description:

AEESoundPlayerStatus is returned in callback functions to indicate ISoundPlayer events and return data to application.

Definition:

```
typedef enum
        {
        AEE_SOUNDPLAYER_UNKNOWN,
        AEE_SOUNDPLAYER_SUCCESS,
        AEE_SOUNDPLAYER_REWIND,
        AEE_SOUNDPLAYER_FFORWARD,
        AEE_SOUNDPLAYER_PAUSE,
        AEE_SOUNDPLAYER_RESUME,
        AEE_SOUNDPLAYER_TEMPO,
        AEE_SOUNDPLAYER_TUNE,
        AEE_SOUNDPLAYER_PAN,
        AEE_SOUNDPLAYER_AUDIO_SPEC,
        AEE_SOUNDPLAYER_TICK_UPDATE,
        AEE_SOUNDPLAYER_DATA_ACCESS_DELAY,
        AEE_SOUNDPLAYER_ABORTED,
        AEE_SOUNDPLAYER_REPEAT,
        AEE_SOUNDPLAYER_DONE,
        AEE_SOUNDPLAYER_FAILURE,
        AEE_SOUNDPLAYER_LAST
        } AEESoundPlayerStatus;
```

Members:

AEE_SOUNDPLAYER_UNKNOWN	Unknown status
AEE_SOUNDPLAYER_SUCCESS	Request is accepted
AEE_SOUNDPLAYER_REWIND	Playback is currently rewinding
AEE_SOUNDPLAYER_FFORWARD	Playback is currently fast forwarding

AEE_SOUNDPLAYER_PAUSE	Playback is currently paused
AEE_SOUNDPLAYER_RESUME	Playback has resumed
AEE_SOUNDPLAYER_TEMPO	Playback tempo changed
AEE_SOUNDPLAYER_TUNE	Playback tune changed
AEE_SOUNDPLAYER_PAN	Playback stereo pan changed
AEE_SOUNDPLAYER_AUDIO_SPEC	Audio spec information
AEE_SOUNDPLAYER_TICK_UPDATE	One second update during playback
AEE_SOUNDPLAYER_DATA_ACCESS_DELAY	Playback is being delayed by data access
AEE_SOUNDPLAYER_ABORTED	Command was aborted
AEE_SOUNDPLAYER_REPEAT	Sound repeating
AEE_SOUNDPLAYER_DONE	Command has been carried out
AEE_SOUNDPLAYER_FAILURE	Error occurred with this request
AEE_SOUNDPLAYER_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundStatus

Description:

AEESoundStatus is returned in callback functions to indicate ISound events and return data to the application.

Definition:

```
typedef enum
        {
        AEE_SOUND_UNKNOWN,
        AEE_SOUND_SUCCESS,
        AEE_SOUND_PLAY_DONE,
        AEE_SOUND_FAILURE,
        AEE_SOUND_LAST
        } AEESoundStatus;
```

Members:

AEE_SOUND_UNKNOWN	Unknown status
AEE_SOUND_SUCCESS	Request is accepted
AEE_SOUND_PLAY_DONE	Playback of the tone is either completed or overridden by another tone
AEE_SOUND_FAILURE	Error occurred with this request
AEE_SOUND_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundTone

Description:

AEESoundTone specifies the tone identifiers which are used to play tone by ISound.

Definition:

```
typedef enum
        {
        AEE_TONE_FIRST,
        AEE_TONE_0,
        AEE_TONE_1,
        AEE_TONE_2,
        AEE_TONE_3,
        AEE_TONE_4,
        AEE_TONE_5,
        AEE_TONE_6,
        AEE_TONE_7,
        AEE_TONE_8,
        AEE_TONE_9,
        AEE_TONE_A,
        AEE_TONE_B,
        AEE_TONE_C,
        AEE_TONE_D,
        AEE_TONE_POUND,
        AEE_TONE_STAR,
        AEE_TONE_CTRL,
        AEE_TONE_2ND,
        AEE_TONE_WARN,
        AEE_TONE_ERR,
        AEE_TONE_TIME,
        AEE_TONE_RING_A,
        AEE_TONE_RING_B,
        AEE_TONE_RING_C,
```

```
AEE_TONE_RING_D,
AEE_TONE_RING_A4,
AEE_TONE_RING_AS4,
AEE_TONE_RING_B4,
AEE_TONE_RING_C4,
AEE_TONE_RING_CS4,
AEE_TONE_RING_D4,
AEE_TONE_RING_DS4,
AEE_TONE_RING_E4,
AEE_TONE_RING_F4,
AEE_TONE_RING_FS4,
AEE_TONE_RING_G4,
AEE_TONE_RING_GS4,
AEE_TONE_RING_A5,
AEE_TONE_RING_AS5,
AEE_TONE_RING_B5,
AEE_TONE_RING_C5,
AEE_TONE_RING_CS5,
AEE_TONE_RING_D5,
AEE_TONE_RING_DS5,
AEE_TONE_RING_E5,
AEE_TONE_RING_F5,
AEE_TONE_RING_FS5,
AEE_TONE_RING_G5,
AEE_TONE_RING_GS5,
AEE_TONE_RING_A6,
AEE_TONE_RING_AS6,
AEE_TONE_RING_B6,
AEE_TONE_RING_C6,
AEE_TONE_RING_CS6,
AEE_TONE_RING_D6,
AEE_TONE_RING_DS6,
AEE_TONE_RING_E6,
AEE_TONE_RING_F6,
```

```
AEE_TONE_RING_FS6,
AEE_TONE_RING_G6,
AEE_TONE_RING_GS6,
AEE_TONE_RING_A7,
AEE_TONE_RBACK,
AEE_TONE_BUSY,
AEE_TONE_INTERCEPT_A,
AEE_TONE_INTERCEPT_B,
AEE_TONE_REORDER_TONE,
AEE_TONE_PWRUP,
AEE_TONE_OFF_HOOK_TONE,
AEE_TONE_CALL_WT_TONE,
AEE_TONE_DIAL_TONE_TONE,
AEE_TONE_ANSWER_TONE,
AEE_TONE_HIGH_PITCH_A,
AEE_TONE_HIGH_PITCH_B,
AEE_TONE_MED_PITCH_A,
AEE_TONE_MED_PITCH_B,
AEE_TONE_LOW_PITCH_A,
AEE_TONE_LOW_PITCH_B,
AEE_TONE_TEST_ON,
AEE_TONE_MSG_WAITING,
AEE_TONE_PIP_TONE_TONE,
AEE_TONE_SPC_DT_INDIA,
AEE_TONE_SIGNAL_INDIA,
AEE_TONE_DT_TONE_INDIA,
AEE_TONE_DT_TONE_BRAZIL,
AEE_TONE_DT_DTACO_TONE,
AEE_TONE_HFK_TONE1,
AEE_TONE_HFK_TONE2,
AEE_TONE_LAST
} AEESoundTone;
```

Members:

AEE_TONE_FIRST	Reserved
AEE_TONE_0	DTMF for 0 key
AEE_TONE_1	DTMF for 1 key
AEE_TONE_2	DTMF for 2 key
AEE_TONE_3	DTMF for 3 key
AEE_TONE_4	DTMF for 4 key
AEE_TONE_5	DTMF for 5 key
AEE_TONE_6	DTMF for 6 key
AEE_TONE_7	DTMF for 7 key
AEE_TONE_8	DTMF for 8 key
AEE_TONE_9	DTMF for 9 key
AEE_TONE_A	DTMF for A key
AEE_TONE_B	DTMF for B key
AEE_TONE_C	DTMF for C key
AEE_TONE_D	DTMF for D key
AEE_TONE_POUND	DTMF for # key
AEE_TONE_STAR	DTMF for * key
AEE_TONE_CTRL	Tone for a control key
AEE_TONE_2ND	Tone for secondary function on a key
AEE_TONE_WARN	Warning tone (for example, overwriting user phone# slot)
AEE_TONE_ERR	Tone to indicate an error
AEE_TONE_TIME	Time marker tone
AEE_TONE_RING_A	1st Ringer tone
AEE_TONE_RING_B	2nd Ringer tone
AEE_TONE_RING_C	3rd Ringer tone
AEE_TONE_RING_D	4th Ringer tone
AEE_TONE_RING_A4	440.0 Hz -Piano Notes-
AEE_TONE_RING_AS4	466.1 Hz
AEE_TONE_RING_B4	493.8 Hz
AEE_TONE_RING_C4	523.2 Hz

AEE_TONE_RING_CS4	554.3 Hz
AEE_TONE_RING_D4	587.3 Hz
AEE_TONE_RING_DS4	622.2 Hz
AEE_TONE_RING_E4	659.2 Hz
AEE_TONE_RING_F4	698.5 Hz
AEE_TONE_RING_FS4	739.9 Hz
AEE_TONE_RING_G4	784.0 Hz
AEE_TONE_RING_GS4	830.6 Hz
AEE_TONE_RING_A5	880.0 Hz
AEE_TONE_RING_AS5	932.2 Hz
AEE_TONE_RING_B5	987.7 Hz
AEE_TONE_RING_C5	1046.5 Hz
AEE_TONE_RING_CS5	1108.7 Hz
AEE_TONE_RING_D5	1174.6 Hz
AEE_TONE_RING_DS5	1244.3 Hz
AEE_TONE_RING_E5	1318.5 Hz
AEE_TONE_RING_F5	1397.0 Hz
AEE_TONE_RING_FS5	1479.9 Hz
AEE_TONE_RING_G5	1568.0 Hz
AEE_TONE_RING_GS5	1661.2 Hz
AEE_TONE_RING_A6	1760.0 Hz
AEE_TONE_RING_AS6	1864.7 Hz
AEE_TONE_RING_B6	1975.5 Hz
AEE_TONE_RING_C6	2093.1 Hz
AEE_TONE_RING_CS6	2217.4 Hz
AEE_TONE_RING_D6	2349.3 Hz
AEE_TONE_RING_DS6	2489.1 Hz
AEE_TONE_RING_E6	2637.0 Hz
AEE_TONE_RING_F6	2793.7 Hz
AEE_TONE_RING_FS6	2959.9 Hz
AEE_TONE_RING_G6	3135.9 Hz
AEE_TONE_RING_GS6	3322.4 Hz

AEE_TONE_RING_A7	3520.0 Hz
AEE_TONE_RBACK	Ring back (audible ring)
AEE_TONE_BUSY	Busy tone
AEE_TONE_INTERCEPT_A	First tone of an intercept
AEE_TONE_INTERCEPT_B	Second tone of an intercept
AEE_TONE_REORDER_TONE	Reorder
AEE_TONE_PWRUP	Power-up tone
AEE_TONE_OFF_HOOK_TONE	Off-hook tone, IS-95 (CAI 7.7.5.5)
AEE_TONE_CALL_WT_TONE	Call-waiting tone
AEE_TONE_DIAL_TONE_TONE	Dial tone
AEE_TONE_ANSWER_TONE	Answer tone
AEE_TONE_HIGH_PITCH_A	1st High pitch for IS-54B alerting
AEE_TONE_HIGH_PITCH_B	2nd High pitch for IS-54B alerting
AEE_TONE_MED_PITCH_A	1st Medium pitch for IS-54B alerting
AEE_TONE_MED_PITCH_B	2nd Medium pitch for IS-54B alerting
AEE_TONE_LOW_PITCH_A	1st Low pitch for IS-54B alerting
AEE_TONE_LOW_PITCH_B	2nd Low pitch for IS-54B alerting
AEE_TONE_TEST_ON	Test tone on
AEE_TONE_MSG_WAITING	Message Waiting Tone
AEE_TONE_PIP_TONE_TONE	Used for Pip-Pip-Pip-Pip (Vocoder) Tone
AEE_TONE_SPC_DT_INDIA	Used for India's Special Dial Tone
AEE_TONE_SIGNAL_INDIA	Used in Various India Signalling Tones
AEE_TONE_DT_TONE_INDIA	Used for India's Normal Dial Tone (and others)
AEE_TONE_DT_TONE_BRAZIL	Used for Brazil's Dial Tone
AEE_TONE_DT_DTACO_TONE	Used for DTACO's single tone (350Hz, 350Hz)
AEE_TONE_HFK_TONE1	These two tones used for Voice Activation and
AEE_TONE_HFK_TONE2	Incoming Call Answer in phone VR-HFK
AEE_TONE_LAST	Reserved

Comments:

None

See Also:

None
Return to the List of data structures

AEESoundToneData

Description:

AEESoundToneData specifies tone id and duration to play the tone using the ISOUND_PlayTone() and ISOUND_PlayToneList().

Definition:

```
typedef struct
        {
        AEESoundTone eTone;
        uint16 wDuration;
        } AEESoundToneData ;
```

Members:

eTone	Tone ID
wDuration	Duration in milliseconds

Comments:

None

See Also:

AEESoundTone
Return to the List of data structures.

AEESymbol

Description:

This ENUM specifies the special symbol whose corresponding AECHAR value is returned by IDISPLAY_GetSymbol(). Some mobile devices have AECHAR values defined for these symbols, and an IDISPLAY_DrawText() call with the AECHAR string corresponding to the symbol draws that symbol on the display.

Definition:

```
typedef enum
        {
        AEE_SYM_ELLIPSES,
        AEE_SYM_AM,
        AEE_SYM_PM,
        AEE_SYM_KEY_LEFT,
        AEE_SYM_KEY_RIGHT,
        AEE_SYM_KEY_CLR,
        AEE_SYM_KEY_1,
        AEE_SYM_KEY_2,
        AEE_SYM_KEY_3,
        AEE_SYM_KEY_4,
        AEE_SYM_KEY_5,
        AEE_SYM_KEY_6,
        AEE_SYM_KEY_7,
        AEE_SYM_KEY_8,
        AEE_SYM_KEY_9,
        AEE_SYM_KEY_0,
        AEE_SYM_KEY_STAR,
        AEE_SYM_KEY_POUND,
        AEE_SYM_KEY_SEND,
        AEE_SYM_KEY_END,
        AEE_SYM_KEY_SELECT
        } AEESymbol;
```

Members:

AEE_SYM_ELLIPSES	Ellipses symbol
AEE_SYM_AM	Symbol indicating AM for 12H time format
AEE_SYM_PM	Symbol indicating PM for 12H time format
AEE_SYM_KEY_LEFT	Left Arrow Key
AEE_SYM_KEY_RIGHT	Right Arrow Key
AEE_SYM_KEY_CLR	Clear Key
AEE_SYM_KEY_1	The "1" Key
AEE_SYM_KEY_2	The "2" Key
AEE_SYM_KEY_3	The "3" Key
AEE_SYM_KEY_4	The "4" Key
AEE_SYM_KEY_5	The "5" Key
AEE_SYM_KEY_6	The "6" Key
AEE_SYM_KEY_7	The "7" Key
AEE_SYM_KEY_8	The "8" Key
AEE_SYM_KEY_9	The "9" Key
AEE_SYM_KEY_0	The "0" Key
AEE_SYM_KEY_STAR	The " * " Key
AEE_SYM_KEY_POUND	The "#" Key
AEE_SYM_KEY_SEND	The "Send", or "Call" Key
AEE_SYM_KEY_END	The "End" Key
AEE_SYM_KEY_SELECT	The Key to select, or "Enter" Key

Comments:

None

See Also:

None
Return to the List of data structures

AEETextInputMode

Description:

This enumerated type specifies the text-entry modes that can be used to enter text into a text control. The function ITEXTCTL_SetInputMode() is used to select the input mode that is used for a particular text control instance.

Definition:

```
typedef enum
        {
        AEE_TM_NONE,
        AEE_TM_CURRENT,
        AEE_TM_SYMBOLS,
        AEE_TM_LETTERS,
        AEE_TM_RAPID,
        AEE_TM_NUMBERS
        } AEETextInputMode;
```

Members:

AEE_TM_NONE	No input mode is currently specified.
AEE_TM_CURRENT	Designates the currently active input mode.
AEE_TM_SYMBOLS	Key presses will enter the special symbol (if any) associated with each key.
AEE_TM_LETTERS	Key presses will enter the letter of the alphabet associated with each key.
AEE_TM_RAPID	Rapid (T9) mode will be used.
AEE_TM_NUMBERS	Key presses will enter the number associated with each key.

Comments:

The available text entry modes differ with each BREW-enabled device.

See Also:

ITEXTCTL_SetInputMode()
Return to the List of data structures.

AEETriangle

Description:

This structure defines the triangle data type.

Definition:

```
typedef struct _triangle {
        int16 x0, y0;
        int16 x1, y1;
        int16 x2, y2;
        } AEETriangle;
```

Members:

x0	X coordinate of the first vertex of the triangle.
y0	Y coordinate of the first vertex.
x1	X coordinate of the second vertex.
y1	Y coordinate of the second vertex.
x2	X coordinate of the third vertex.
y2	Y coordinate of the third vertex.

Comments:

None

See Also:

IGRAPHICS_DrawTriangle()
IGRAPHICS_Translate()
Return to the List of data structures.

AEEVoicePrompt

Description:

AEEVoicePrompt is not implemented.

Definition:

```
typedef struct
        {
        byte * pbData;
        uint16 wNumFrames;
        } AEEVoicePrompt;
```

Members:

pbData	Not implemented
wNumFrames	Not implemented

Comments:

None

See Also:

None
Return to the List of data structures

BeepType

Description:

This ENUM specifies the beep type in ISHELL_Beep() function.

Definition:

```
typedef enum {
        BEEP_OFF,
        BEEP_ALERT,
        BEEP_REMINDER,
        BEEP_MSG,
        BEEP_ERROR,
        BEEP_VIBRATE_ALERT,
        BEEP_VIBRATE_REMIND
        } BeepType;
```

Members:

BEEP_OFF	Beep Off
BEEP_ALERT	Beep to alert mobile user
BEEP_REMINDER	Reminder beep
BEEP_MSG	Beep to indicate arrival of SMS message
BEEP_ERROR	Beep to indicate error
BEEP_VIBRATE_ALERT	Silent mode vibration to alert mobile user
BEEP_VIBRATE_REMIND	Silent mode vibration to remind mobile user

Comments:

On a target device, BeepType is dependent on device manufacturers. In the BREW Emulator, device manufacturers or application developers can define their own BeepType tones. For example, to create your own BEEP_ALERT tone, save your .WAV file as "BEEP_ALERT.wav" in \Brew\bin\DataFiles directory. Sample tones for all beep types are provided and their usage is illustrated in the sample sound app.

See Also:

None
Return to the List of data structures

CtlAddItem

Description:

An encapsulation for a control item added to the control.

Definition:

```
typedef struct _CtlAddItem
        {
        const AECHAR * pText;
        IImage * pImage;
        const char * pszResImage;
        const char * pszResText;
        uint16 wText;
        uint16 wFont;
        uint16 wImage;
        uint16 wItemID;
        uint32 dwData;
        } CtlAddItem;
```

Members:

pText	Text in the item
pImage	Image in the item
pszResImage	Name of the resource file
pszResText	Name of the resource file
wText	Resource ID of the text string
wFont	0 (zero): Default
wImage	Resource ID of the Image
wItemID	Control item ID
dwData	Data value associated with menu item

Comments:

pText and **pImage** are used by default. If they are not set (NULL), the **pszResImage** and **pszResText** are used with **wText** and **wImage** to load the text and/or image respectively.

See Also:

None
Return to the List of data structures.

DialogInfo

Description:

This ENUM specifies the resource type to identify resources in a resource file.

Definition:

```
typedef struct
        {
        DialogInfoHead h;
        DialogItem controls[1];
        } DialogInfo;
```

Members:

h	Dialog Information header
controls	Array of dialog items

Comments:

None

See Also:

None
Return to the List of data structures.

DialogInfoHead

Description:

This structure specifies the dialog information header.

Definition:

```
typedef struct
        {
        uint16 wID;
        uint16 nControls;
        AEERect rc;
        uint32 dwProps;
        uint16 wTitle;
        uint16 wFocusID;
        } DialogInfoHead;
```

Members:

wID	Dialog ID (Resource ID of the Dialog)
nControls	Number of controls
rc	Dialog rectangle ((-1, -1, -1, -1) is default)
dwProps	Property of the dialog (Dialogs currently supports only the CP_BORDER, CP_3D_BORDER control parameters.)
wTitle	Dialog Title. This field is not supported currently
wFocusID	ID of the control that is active when the dialog is started

Comments:

None

See Also:

AEE Standard Control Properties
Return to the List of data structures.

DialogItem

Description:

This structure contains a dialog item.

Definition:

```
typedef struct
        {
        DialogItemHead h;
        DListItem items[1];
        } DialogItem;
```

Members:

h	Dialog Item header
items	Array of dialog list items

Comments:

None

See Also:

None
Return to the List of data structures.

DialogItemHead

Description:

This structure defines the dialog item header.

Definition:

```
typedef struct
        {
        AEECLSID cls;
        uint16 wID;
        uint16 nItems;
        uint32 dwProps;
        uint16 wTextID;
        uint16 wTitleID;
        AEERect rc;
        } DialogItemHead;
```

Members:

cls	ClassID of the control.
wID	Control ID (Resource ID of the control).
nItems	Number of dialog list items.
dwProps	Property of the dialog control (See CP_BORDER, CP_STATIC, etc)
wTextID	Resource ID of text string.
wTitleID	Resource ID of title string.
rc	Rectangle relative to the dialog ((-1, -1, -1, -1) is default).

Comments:

None

See Also:

AEERect
Return to the List of data structures.

DListItem

Description:

This structure defines a dialog list item.

Definition:

```
typedef struct _DListItem
        {
        uint16 wID;
        uint16 wTextID; // Text
        uint16 wIconID; // Icon ID
        uint16 pad;
        uint32 dwData; // 32-bit data
        } DListItem;
```

Members:

wID	ID of the dialog list item.
wTextID	Resource ID of the text string.
wIconID	Resource ID of the Icon.
pad	Reserved
dwData	Data that is sent with the event when the item is selected.

Comments:

None

See Also:

None
Return to the List of data structures.

FileAttrib

Description:

FileAttrib specifies the type of a file.

Definition:

```
typedef enum
        {
        _FA_NORMAL=0,
        _FA_HIDDEN=0x0001,
        _FA_DIR=0x0002,
        _FA_READONLY=0x0004,
        _FA_SYSTEM=0x0008
        } FileAttrib;
```

Members:

_FA_NORMAL	File is normal file
_FA_HIDDEN	File is a hidden file (reserved)
_FA_DIR	File is directory (reserved)
_FA_READONLY	File is read only file
_FA_SYSTEM	File is system file

Comments:

None

See Also:

None
Return to the List of data structures.

FileInfo

Description:

FileInfo is used to contain information associated with a file.

Definition:

```
typedef struct _FileInfo
        {
        FileAttrib attrib;
        uint32 dwCreationDate;
        uint32 dwSize;
        char szName[MAX_FILE_NAME];
        } FileInfo;
```

Members:

attrib	File attributes specified by FileAttrib.
dwCreationDate	File creation date. The reference time is Jan 6. 1980. The unit of time measured is seconds,
dwSize	File size
szName	File name

Comments:

None

See Also:

FileAttrib
Return to the List of data structures.

FileSeekType

Description:

FileSeekType is used to specify the origin of seek operation when setting the current file pointer position using the IFILE_Seek().

Definition:

```
typedef enum
        {
        _SEEK_START,
        _SEEK_END,
        _SEEK_CURRENT
        } FileSeekType;
```

Members:

_SEEK_START	Start seek from the start of the file
_SEEK_END	Start seek from the end of the file
_SEEK_CURRENT	Start seek from current file pointer position

Comments:

None

See Also:

None
Return to the List of data structures.

IDISPLAY Flags

Description:

These constants are used to build the flags (dwFlags parameter) in the IDISPLAY_DrawRect() and IDISPLAY_DrawText() functions.

Definition:

Flags for the Rectangle

NOTE: IDF_RECT_INVERT overrides the other IDF_RECT flags.

IDF_RECT_NONE	No Rectangle, used in IDISPLAY_DrawText()
IDF_RECT_FRAME	Draw rectangular frame.
IDF_RECT_FILL	Draw filled rectangle.
IDF_RECT_INVERT	Invert the text and background colors.
IDF_RECT_MASK	Mask to isolate IDF_RECT flags.

Flags for Horizontal alignment of text

NOTE: These flags are mutually exclusive:

IDF_ALIGN_NONE	No alignment specified.
IDF_ALIGN_LEFT	Left justified.
IDF_ALIGN_CENTER	Centered.
IDF_ALIGN_RIGHT	Right justified.
IDF_ALIGN_FILL	Fill the rectangle width (specified with IDISPLAY_DrawText()) with the text.
IDF_ALIGNHORZ_MASK	Mask to isolate ALIGN(horizontal) flags Vertical alignment flags for text
IDF_ALIGN_TOP	Align text with top of rectangle specified.
IDF_ALIGN_MIDDLE	Place text in the middle of the specified rectangle.
IDF_ALIGN_BOTTOM	Align text with the bottom of the rectangle specified.
IDF_ALIGN_SPREAD	Spread the text over the height of the rectangle.

IDF_ALIGNVERT_MASK	Mask to isolate ALIGN(vertical) flags.

IDF_ALIGN_MASK (IDF_ALIGNHORZ_MASK|IDF_ALIGNVERT_MASK)

Flags for Text Formats

IDF_TEXT_UNDERLINE	Underline text
IDF_TEXT_INVERTED	Inverted text (highlights)
IDF_TEXT_TRANSPARENT	Background is preserved
IDF_TEXT_FORMAT_OEM	Text is OEM_RAW_TEXT format (byte *)

Members:

None

Comments:

None

See Also:

None
Return to the List of data structures.

IGRAPHICS Flags

Description:

The bitmap flags to control the behavior of a number of IGraphics functions.

Definition:

AEE_GRAPHICS_NONE	Default to no flags
AEE_GRAPHICS_FRAME	if this flag is set, the frame of the geometric primitive is drawn
AEE_GRAPHICS_CLEAR	If this flag is set, the interior is cleared to the background color
AEE_GRAPHICS_FILL	If this flag is set, the interior is filled.

Members:

None

Comments:

None

See Also:

IGRAPHICS_SetClip().
Return to the List of data structures.

ITField

Description:

Field in the time control. There are three fields in each time control, shown as following.

Definition:

```
typedef enum {ITF_HOUR, ITF_MIN, ITF_SEC} ITField;
```

Members:

ITF_HOUR	The hour field of the time control
ITF_MIN	The minute field of the time control
ITF_SEC	The second field of the time control

Comments:

None

See Also:

None
Return to the List of data structures.

JulianType

Description:

This structure contains Julian date information.

Definition:

```
typedef struct
        {
        uint16 wYear;
        uint16 wMonth;
        uint16 wDay;
        uint16 wHour;
        uint16 wMinute;
        uint16 wSecond;
        uint16 wWeekDay;
        } JulianType;
```

Members:

wYear	4-digit year
wMonth	Month 0-11(January=0, December=11)
wDay	Day 1-31
wHour	Hour 0-23
wMinute	Minute 0-59
wSecond	Seconds 0-59
wWeekDay	Day of the week 0-6 (0=Sunday, 6=Saturday)

Comments:

None

See Also:

None
Return to the List of data structures

NetSocket

Description:

NetSocket is an enumeration of the types of sockets that can be created with INetMgr Interface. A NetSocket value is passed to INETMGR_OpenSocket().

Definition:

```
typedef enum
        {
        AEE_SOCK_STREAM=0,
        AEE_SOCK_DGRAM
        } NetSocket;
```

Members:

AEE_SOCK_STREAM	TCP: streaming socket
AEE_SOCK_DGRAM	UDP: datagram socket

Comments:

None

See Also:

None
Return to the List of data structures

NetState

Description:

NetState is an enumeration of the states of the handset's PPP connection to the Internet. A NetState value is returned by the INETMGR_NetStatus() call.

Definition:

```
typedef enum
        {
        NET_INVALID_STATE,
        NET_PPP_OPENING,
        NET_PPP_OPEN,
        NET_PPP_CLOSING,
        NET_PPP_CLOSED
        } NetState;
```

Members:

NET_INVALID_STATE	Not an actual state; this value will not be returned by INETMGR_NetStatus()
NET_PPP_OPENING	The PPP connection is being established
NET_PPP_OPEN	The PPP connection is active
NET_PPP_CLOSING	The PPP connection is closing
NET_PPP_CLOSED	The PPP connection is inactive

Comments:

None

See Also:

None
Return to the List of data structures.

OpenFileMode

Description:

OpenFileMode is used to specify file opening mode when opening a file using the IFILEMGR_OpenFile().

Definition:

```
typedef enum
        {
        _OFM_READ=0x0001,
        _OFM_READWRITE=0x0002,
        _OFM_CREATE=0x0004,
        _OFM_APPEND=0x0008,
        _OFM_NO_BUFFER=0x8000
        } OpenFileMode;
```

Members:

_OFM_READ	Open file in read only mode
_OFM_READWRITE	Open file in read/write mode
_OFM_CREATE	Open file in create mode
_OFM_APPEND	Open file in append mode
_OFM_NO_BUFFER	Disable data buffering

Comments:

Option _OFM_NO_BUFFER doesn't do anything on the BREW Emulator.

See Also:

None
List of data structures.

PFNAEEEVENT

Description:

PFNAEEEVENT specifies the type of the event callback passed to IDIALOG_SetEventHandler().

Definition:

```
typedef boolean ( * PFNAEEEVENT)
        (
        void * pUser,
        AEEEvent evt,
        uint16 w,
        uint32 dw
        );
```

Members:

pUser	User Data
evt	Event code
w	16-bit event-specific parameter
dw	32-bit event-specific parameter

Comments:

None.

See Also:

IDIALOG_SetEventHandler()
Return to the List of data structures

PFNCONNECTCB

Description:

PFNCONNECTCB specifies the type of the callback function passed to ISOCKET_Connect().

Definition:

```
typedef void (*PFNCONNECTCB)
        (
           void * pUser,
           int nError
        );
```

Members:

pUser	User Data
nError	Error Code

Comments:

None.

See Also:

ISOCKET_Connect()
Return to the List of data structures

PFNIMAGEINFO

Description:

PFNIMAGEINFO specifies the type of the callback function passed to IIMAGE_Notify().

Definition:

```
typedef void ( * PFNIMAGEINFO)
        (
        void * pUser,
        IImage * pIImage,
        AEEImageInfo * pi,
        int nErr
        );
```

Members:

pUser	User Data
pIImage	Pointer to IIMage interface object
pi	Pointer to image information
nErr	Error Code

Comments:

None.

See Also:

IIMAGE_Notify()
AEEImageInfo
Return to the List of data structures

PFNPOSITIONCB

Description:

PFNPOSITIONCB specifies the type of the callback function passed to ISHELL_GetPosition().

Definition:

```
typedef void ( * PFNPOSITIONCB)
        (
        void * pUser,
        AEEPositionInfo * pli,
        int nErr
        );
```

Members:

pUser	User Data
pli	Position Location Information
nErr	Error Code

Comments:

None.

See Also:

ISHELL_GetPosition()
Return to the List of data structures.

PFNSOUNDPLAYERSTATUS

Description:

PFNSOUNDPLAYERSTATUS is the type specification for the ISoundPlayer callback function that application must register with ISoundPlayer. ISoundPlayer sends all the events and data to application via the registered callback function.

Definition:

```
typedef void ( * PFNSOUNDPLAYERSTATUS)
        (
        void * pUser,
        AEESoundPlayerCmd eCBType,
        AEESoundPlayerStatus eSPStatus,
        uint32 dwParam
        );
```

Members:

pUser	Application specified data pointer
eCBType	Type of callback
eSPStatus	Status within the callback type
dwParam	Pointer to AEESoundPlayerCmdData, if any. NULL otherwise.

Comments:

None

See Also:

AEESoundPlayerCmd,
AEESoundPlayerStatus
Return to the List of data structures.

PFNSOUNDSTATUS

Description:

PFNSOUNDSTATUS is the type definition of the **ISound** callback function that applications must register with **ISound**. **ISound** sends all the events and data to the applications via the registered callback function.

Definition:

```
typedef void ( * PFNSOUNDSTATUS)
        (
        void * pUser,
        AEESoundCmd eCBType,
        AEESoundStatus eSPStatus,
        uint32 dwParam
        );
```

Members:

pUser	Application specified data pointer
eCBType	Type of callback
eSPStatus	Status within the callback type
dwParam	Pointer to AEESoundCmdData, if any. NULL otherwise.

Comments:

None

See Also:

AEESoundCmd,
AEESoundPlayerStatus
AEESoundCmdData
Return to the List of data structures.

ResType

Description:

This ENUM specifies the resource type to identify resources in a Resource file.

Definition:

```
typedef enum
        {
        RESTYPE_STRING=1,
        RESTYPE_IMAGE=6,
        RESTYPE_DIALOG=0x2000,
        RESTYPE_CONTROL=0x2001,
        RESTYPE_LISTITEM=0x2002,
        RESTYPE_BINARY=0x5000
        } ResType;
```

Members:

RESTYPE_STRING	String resource, made of UNICODE or ISOLATIN strings
RESTYPE_IMAGE	Image resource, made of .BMP, PNG, or GIF images
RESTYPE_DIALOG	Dialog resource, made up of one or more dialog controls
RESTYPE_CONTROL	Control resource
RESTYPE_LISTITEM	List Item Type resource
RESTYPE_BINARY	Binary data resource

Comments:

None

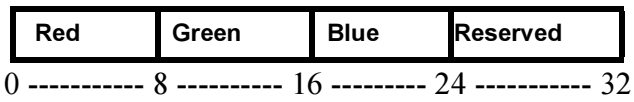
See Also:

None
Return to the List of data structures.

RGBVAL

Description:

The RGB value for a color is defined using this data type. The eight-bit values for red, green, blue are stored in 32-bits as follows:



Definition:

```
typedef uint32 RGBVAL
```

Members:

None

Comments:

The user can create their own colors using the MAKE_RGB macro with their values for red, green and blue to get the corresponding **RGBVAL**.

See Also:

None
Return to the List of data structures.

SockIOBlock

Description:

A single structure describes an individual block of memory from which data is read or to which data is written.

Arrays of SockIOBlock structures are used in calls to ISOCKET_ReadV() and ISOCKET_WriteV() to describe data that can be sent/received as a continuous stream even when, in memory, it is scattered among several blocks.

Definition:

```
typedef struct
        {
        byte * pbBuffer;
        uint16 wLen;
        } SockIOBlock;
```

Members:

pbBuffer	Data Buffer
wLen	Length of Buffer

Comments:

None

See Also:

None
Return to the List of data structures.

TChType

Description:

TChType is an enumeration used to return the type of the wide character by the GETCHTYPE() function.

Definition:

```
typedef enum
        {
        SC_UNKNOWN,
        SC_ALPHA,
        SC_DIGIT,
        SC_WHITESPACE
        } TChType;
```

Members:

SC_UNKNOWN	Unknown type
SC_ALPHA	Alphabet type
SC_DIGIT	Numeric type (0-9)
SC_WHITESPACE	White Space

Comments:

None

See Also:

None
Return to the List of data structures.