QUALCOMM®

Binary Runtime Environment for Wireless™

BREW™ 2.1 OEM API Reference for MSM™ Platforms

brew™

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

BREW™ 2.1 OEM API Reference for MSM™ Platforms

May 12, 2003

Contents

Data Types 728

Introducing the BREW OEM API Reference for MSM™ Platform

This document provides manufacturers with information about the Binary Runtime Environment for Wireless™ (BREW™) OEM functions. With these functions, OEM devices become BREW-enabled.

In this reference

This remainder of the this reference manual contains the following sections:

BREW API Interfaces	Describes the BREW interfaces and functions that are used by the OEM APIs.
Data Types	Describes the data structures that are used by the OEM APIs.
Functions and Data types	Provides an alphabetic listing of all functions and data types discussed in this reference.

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 to be input and the items returning.
Return Values:	The items returning from the function call, including 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 that may not be normally considered when using a function call. This heading only appears when there is a side effect.
See Also:	Cross-references to any related function or data structure.

BREW SDK™ documentation set

The BREW documentation set contains the following documents:

BREW SDK User's Guide	Introduces the components of the BREW Software Development Kit (BREW SDK™) and their relationship to one another. The document also contains general instructions for developing BREW applications and for using the BREW Emulator.
BREW API Reference	Provides information about BREW functions and data structures needed to develop applications for BREW-enabled mobile platforms.
BREW Device Configurator Guide	Describes how to use the BREW Device Configurator to create effective wireless devices for emulation by the BREW Emulator.
BREW Resource Editor Guide	Describes how to use the BREW Resource Editor to create the text strings, images, and dialogs for BREW applications.
BREW MIF Editor Guide	Describes how to use the BREW 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.
BREW SDK™ Utilities Guide	Describes how to use the utilities, such as the PureVoice Converter, included with the BREW SDK.
BREW Compressed Image Authoring Guide	Describes how to use the BREW Compressed Image Authoring Tool to create files for displaying and animating images in your applications.

BREW OEM documentation set

The BREW OEM documentation set includes the following documents:

BREW OEM Porting Guide	Describes the interfaces required from the OEM that allow the BREW AEE to provide various applications services.
BREWapi OEM API Reference	Describes the OEM mobile interface layer (MIL) and chip interface layer (ChIL) details.
BREW OEM Application Test (OAT) Guide	Describes which tests to run and how to run them.
BREW Core Application Guide	Provides MobileShop/Application Manager application requirements and reference UI specs for an OEM that is developing a device for a carrier.
BREWStone™ User's Guide	Describes the Brewstone benchmarking tool designed to test the efficiency of BREW devices against the base device QCP3035.

BREW OEM acronyms

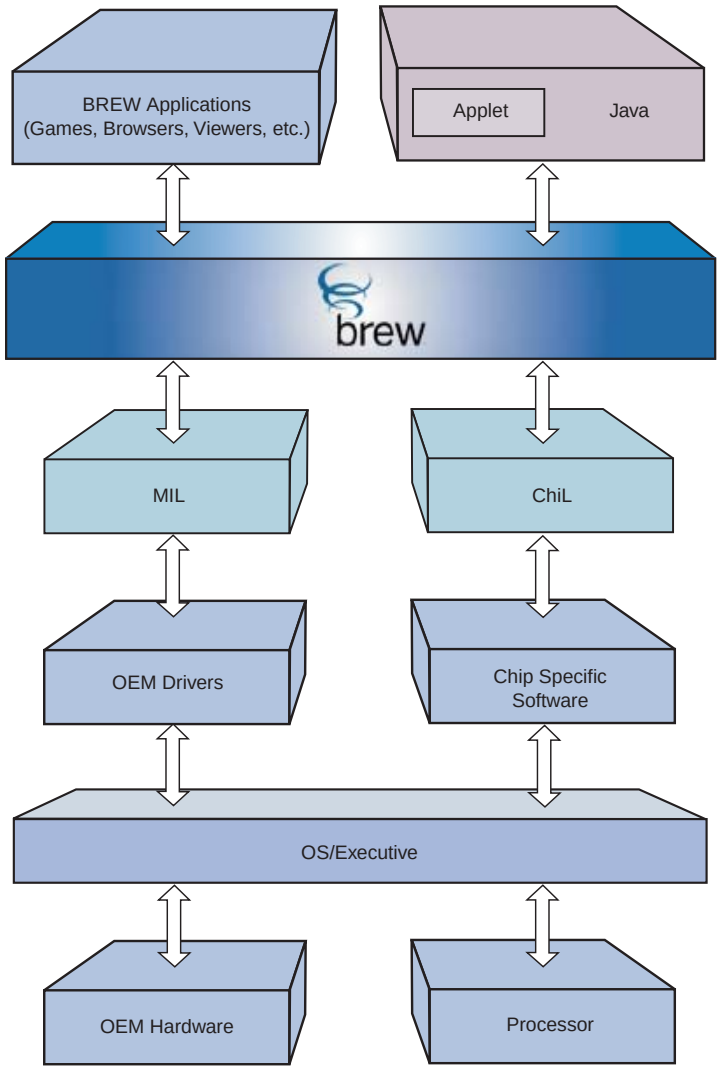
ADS	Application Download Server
AEE	Application Execution Environment
APCS	ARM Procedure Call Standard
API	Application Programming Interface
ARM	Asynchronous Response Mode
ASIC	Application-specific integrated circuit
BREW	Binary Runtime Environment for Wireless.
ChIL	Chip Interface Layer
ClassID	32-bit IDs for identifying BREW classes and applets. These IDs are assigned at the site www.qualcomm.com/brew/sdk/classid. BREW ClassIDs are available to authenticated developers only.
DC	Display Context
DMSS	Dual Mode Subscriber Software
Interface	An abstract class providing a set of methods for a specific service. For example, the IDisplay interface provides a set of methods for basic display services. Each interface has a unique class identifier (AEECLSID), and the name of each interface begins with the letter “I.” In BREW, all the interfaces are derived from a base level class interface called IBase. IBase consists of two standard methods for incrementing and decrementing the reference count of an object. This reference count mechanism allows an object to be shared by multiple users.
IP	Internet Protocol
ISOD	Interface Specification and Operational Description
MIDI	Musical Instrument Digital Interface
MIF	Module Information File. The MIF Editor generates this binary file, which contains information regarding the list of classes and applets supported by the modules.
MIL	Mobile Interface Layer
MIME	Multipurpose Internet Mail Extensions
NVRAM	Non-volatile random access memory
OAT	BREW OEM Acceptance Test
OEM	Original equipment manufacturer The phrase 'called by the OEM” means the function is called from the device code.
PNG	Portable Network Graphics
RAM	Random access memory
SDK	Software development kit
SDT	Software development toolkit
SMS	Short message service

TCP	Transmission Control Protocol
UI	OEM device user interface (non-BREW)
UTF8	Unicode Transformation format 8 bit encoding form. UTF8 serializes a Unicode scalar value (code point) as a sequence of one to four bytes.
Wide string	A character string composed of 16-bit characters. Wide strings are used for character encoding, such as Unicode that require more than 8 bits per character.

BREW architecture

The following figure illustrates BREW architecture and its interaction with other software components of the device.

BREW architecture and interactions



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.

AEEBitFont Interface

The Interface provides a way to create an IFont interface from a Brew Bit Font (BBF).

List of Header files to be included

The following header file is required:

AEEBitFont.h

List of functions

Functions in this interface include:

AEEBitFont_NewFromBBF()

The remainder of this section provides details for each function.

AEEBitFont_NewFromBBF()

Description:

This function may be called by the OEM layer to create an IFont interface from a Brew Bit Font (BBF). A BBF is created using the BBFGEN.EXE tool.

Prototype:

```
int AEEBitFont_NewFromBBF
      (
      const byte *pbyMem,
      int cbMem,
      void *pvStorage,
      PFNREALLOC pfnRealloc,
      IFont **ppIFont
      )
```

Parameters:

pbyMem	[in]	Source data in BBF format.
cbMem	[in]	Size in bytes of the BBF data in **pbyMem**.
pvStorage	[in]	Controls how the memory in **pbyMem**[] is treated. It will hold one of the following values: 1. pointer to memory block => the object can retain pointers into **pbyMem**[] instead of copying out the data. If the object is successfully created, it will free this pointer using **pfnRealloc** at some point. If creation fails, this pointer will not be freed. 2. AEEBITFONT_COPYDATA => This function must copy any data it needs from **pbyMem**[] before returning. 3. AEEBITFONT_STATICDATA => The IFont will retain pointers into **pbyMem**[]. It does not need to free anything.
pfnRealloc	[in]	Pointer to realloc() function to be used for all memory allocation and deallocation. This may be NULL, in which case the BREW heap will be used for all memory operations.
ppIFont	[out]	New instance of IFont interface created from the BBF.

Return Value:

SUCCESS if successful.

EBADPARM if a parameter is invalid.

EINVALIDFORMAT if **pbyMem** is not in BBF format.

ENOMEMORY if not enough memory.

Comments:

None.

See Also:

None.

Return to the List of functions

AEE Media Interface

AEEMedia is the base class of all IMedia-based classes. The Interface provides applications the ability to

- Register the MIME type of the class
- Make sure a command can be executed in a particular state
- Manage the IMedia state machine
- Handle IMedia callbacks and makes callbacks into application
- Stop IMedia when IMedia is released

List of Header files to be included

The following header file is required:

AEEMedia.h

List of functions

Functions in this interface include:

AEEMedia_AddRef()
AEEMedia_CallbackNotify()
AEEMedia_Delete()
AEEMedia_GetMediaParm()
AEEMedia_GetState()
AEEMedia_GetTotalTime()
AEEMedia_Init()
AEEMedia_New()
AEEMedia_Pause()
AEEMedia_Play()
AEEMedia_QueryInterface()
AEEMedia_Record()
AEEMedia_RegisterNotify()
AEEMedia_Release()
AEEMedia_Resume
AEEMedia_Seek()
AEEMedia_SetMediaParm()
AEEMedia_Stop()

The remainder of this section provides details for each function.

AEEMedia_AddRef()

Description:

Increments the reference count of IMedia object.

Prototype:

```
uint32 AEEMedia_AddRef(IMedia * po);
```

Parameters:

po: Pointer to IMedia

Return Value:

Incremented ref count

Comments:

None

See Also:

AEEMedia_Release()

Return to List of functions

AEEMedia_CallbackNotify()

Description:

This function is registered with CMediaMMLayer for callbacks from the multimedia layer. It performs state management and calls the user-registered callback function.

Prototype:

```
void AEEMedia_CallbackNotify(AEEMedia * pme, AEEMediaCallback * pmcb);
```

Parameters:

pme: Pointer to AEEMedia

pmcb: Pointer to media callback structure

Return Value:

None

Comments:

This function is called in BREW context.

See Also:

AEEMediaCallback

AEEMedia_Play()

AEEMedia_Record()

AEEMedia_Stop()

Return to List of functions

AEEMedia_Delete()

Description:

This function is the base class destructor.

Prototype:

```
void AEEMedia_Delete(IMedia * po);
```

Parameters:

po: Pointer to IMedia

Return Value:

None

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_GetMediaParm()

Description:

This function handles MM_PARM_MEDIA_DATA and MM_PARM_CLSID.

Prototype:

```
int AEEMedia_GetMediaParm
      (
      IMedia * po,
      int nParamID,
      int32 * pP1,
      int32 * pP2
      );
```

Parameters:

po: Pointer to the IMedia Interface object

nParmID: MM_PARM_XXX

pP1: Depends on parm

pP2: Depends on parm

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot get parm in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_GetState()

Description:

This function returns the current state of IMedia and also indicates the IMedia object is currently in state transition.

Prototype:

```
int AEEMEDIA_GetState(IMedia * po, boolean * pbStateChanging);
```

Parameters:

po: Pointer to the IMedia Interface object

pbStateChanging: TRUE means IMedia is currently busy transitioning the state.

Return Value:

MM_STATE_XXX

Comments:

None

Side Effects:

If IMedia is currently is in state transition, then most of IMedia APIs fail and return EBADSTATE.

See Also:

None

Return to List of functions

AEEMedia_GetTotalTime()

Description:

This function checks if get total time command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_GetTotalTime(IMedia * po);
```

Parameters:

po: Pointer to IMedia

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_Init()

Description:

This function registers the MIME type in the Shell registry.

Prototype:

```
void AEEMedia_Init(IShell * ps, char * szMIME, AEECLSID clsHandler);
```

Parameters:

ps: Pointer to IShell

szMIME: MIME string

clsHandler: Class ID of the IMedia-based class

Return Value:

None

Comments:

This function is called only once during BREW initialization.

See Also:

None

Return to List of functions

AEEMedia_New()

Description:

This function is the base class constructor.

Prototype:

```
int AEEMedia_New(IMedia * po, IShell * ps, AEECLSID cls);
```

Parameters:

po: Pointer to IMedia

ps: Pointer to IShell

cls: Class ID of the IMedia-based class

Return Value:

SUCCESS: Successful

Comments:

None

See Also:

AEEMedia_Delete()

AEEMedia_Init()

Return to List of functions

AEEMedia_Pause()

Description:

This function checks if pause command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Pause(IMedia * po);
```

Parameters:

po: Pointer to IMedia

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_Play()

Description:

This function checks if playback command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Play(IMedia * po);
```

Parameters:

po: Pointer to IMedia

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_QueryInterface()

Description:

This function can be used to

(1) get a pointer to an interface or data based on the input class ID

(2) query an extended version of the IMedia-derived class

(3) support version compatibility

Prototype:

```
int AEEMEDIA_QueryInterface(IMedia * po, AEECLSID clsReq, void ** ppo);
```

Parameters:

po	[in]	Pointer to IMedia interface.
clsReq	[in]	A globally unique id to identify the entity (interface or data) that we are trying to query.
ppo	[out]	Pointer to the interface or data that we want to retrieve. If the value passed back is NULL, the interface or data that we query are not available.

Return Value:

Return SUCCESS on success, otherwise returns error code.

Comments:

If **ppo** is back a NULL pointer, the interface or data that we query is not available.

Side Effects:

If an interface is retrieved, then this function increments its reference count.

If a data structure is retrieved, then a pointer to the internal structure is given and user should not free it.

See Also:

None

Return to List of functions

AEEMedia_Record()

Description:

This function checks if record command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Record(IMedia * po);
```

Parameters:

po	Pointer to AEE Media Interface object.

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

MM_EBADMEDIADATA: Bad media data. Possibly ISource is used for recording.

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_RegisterNotify()

Description:

This function registers a callback notification function with IMedia object.

IMedia reports asynchronous events thorough this callback.

Prototype:

```
int AEEMEDIA_RegisterNotify
      (
      IMedia * po,
      PFNMEDIANOTIFY pfnNotify,
      void * pUser
      );
```

Parameters:

po	Pointer to AEE Media Interface object.
pfnNotify	User callback function pointer.
pUser	User data to be used when calling **pfnNotify**().

Return Value:

SUCCESS: Successful.

EBADSTATE: Error - IMedia is not in Ready state.

Comments:

None

See Also:

PFNMEDIANOTIFY

Return to List of functions

AEEMedia_Release()

Description:

Decrements the reference count of IMedia object. If reference count goes down to zero, then it deregisters the user registered callback and stops the IMedia playback/recording, if any.

Prototype:

```
uint32 AEEMedia_Release(IMedia * po);
```

Parameters:

po	Pointer to AEE Media Interface object.

Return Value:

Updated reference count. Zero if object is released.

Comments:

None

See Also:

AEEMedia_AddRef()

Return to List of functions

AEEMedia_Resume

Description:

This function checks if resume command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Resume(IMedia * po);
```

Parameters:

po	Pointer to AEE Media Interface object.

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_Seek()

Description:

This function checks if seek command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Seek(IMedia * po, AEEMediaSeek eSeek, int32 lTimeMS);
```

Parameters:

po	Pointer to AEE Media Interface object.
eSeek	The seek reference
lTimeMS	The seek time

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

AEEMediaSeek

Return to List of functions

AEEMedia_SetMediaParm()

Description:

This function handles MM_PARM_MEDIA_DATA in the Idle state.

Prototype:

```
int AEEMedia_SetMediaParm
      (
      IMedia * po,
      int nParamID,
      int32 p1,
      int32 p2
      );
```

Parameters:

po	Pointer to the AEE Media Interface object
nParmID	MM_PARM_XXX
p1	Depends on parm
p2	Depends on parm

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot set parm in the current state

Comments:

None

See Also:

None

Return to List of functions

AEEMedia_Stop()

Description:

This function checks if stop command is valid in the current IMedia state.

Prototype:

```
int AEEMedia_Stop(IMedia * po);
```

Parameters:

po	Pointer to AEE Media Interface object.

Return Value:

SUCCESS: Successful

EBADSTATE: Cannot issue command in the current state

Comments:

None

See Also:

None

Return to List of functions

OEMMedia_DetectType()

Description:

Given data in a buffer or the name of an object, this function detects the MIME type of the media. This function is typically used to get the handler associated with the data type. For example, if the data represents standard MIDI format, then this function returns the MIME “audio/mid”. Using the MIME type, you can query Shell registry to obtain the handler (Class ID) of type AEECLSID_MEDIA.

Prototype:

```
int OEMMedia_DetectType
     (
     const void * cpBuf,
     uint32 * pdwSize,
     const char * cpszName,
     const char ** pcpszMIME
     );
```

Parameters:

cpBuf	[in]	Buffer containing the data whose type needs to be determined
pdwSize	[in/out]	On input - Size of data in **pBuf**, unless **pBuf** is NULL, then ignored On output - number of additional data bytes needed to perform type detection
cpszName	[in]	Name of the object whose type needs to be determined (may be null, if unknown).
pcpszMIME	[out]	MIME string returned to caller, on return, filled with a pointer to a constant string (do not free)

Return Value:

SUCCESS: Data type is detected and MIME is returned

ENOTYPE: There is no type associated with this data

EBADPARM: Wrong input data (parameter(s))

ENEEDMORE: Need more data to perform type detection. ***pdwSize** contains the required number of additional bytes.

EUNSUPPORTED: Type detection for the specified input is not supported

Comments:

pBuf takes precedence over **pszName**. If both of them are specified, then first **pBuf** is used for type detection followed by **pszName**.

If the function returns ENEEDMORE, then ***pdwSize** is filled with the required additional bytes to carry out the operation. Call this function again with (original **dwSize** + ***pdwSize**) bytes.

To determine the maximum number of bytes required to enable type detection, you can call

```
if (ENEEDMORE == ISHELL_DetectType(ps, NULL, &dwReqSize, NULL,
NULL))
{
// dwReqSize contains the max bytes needed for type detection
}
```

See Also:

OEMRegistry_DetectType()

ISHELL_DetectType()

ISHELL_GetHandler()

ISHELL_CreateInstance().

Return to List of functions

AEE Object Manager Interface

Object Manager provides an interface to

- Manage the contexts of BREW objects
  (created in application context and of finite lifetime) that participate in asynchronous operations of infinite timeout
- Facilitate the validation of the objects in asynchronous callbacks

Why is ObjectMgr needed?

To illustrate the usage of ObjectMgr, consider the following situation:

1. An application creates IMedia-based object and calls IMEDIA_Play()
2. IMEDIA_Play() in OEM layer implementation calls lower-layer device multimedia API that takes a callback function and user data to correlate the transaction. Assume we set user data to IMedia object pointer
3. The callback function gets fired when multimedia task sends events to BREW in that task's context. BREW correlates the transaction, identifies the IMedia object, saves the event info and requests for a context switch.
4. When BREW gets scheduled, BREW processes the event info corresponding to IMedia object and delivers the event to application If the application is unloaded just before step (3) occurs, then IMedia object pointer returned in callback event in is invalid. Also, step (3) could occur anytime.

ObjectMgr solves the problem

When the IMedia object is created, the OEM layer implementation must register IMedia object with ObjectMgr. ObjectMgr saves object info and returns an opaque object context. This context must be used as user data in asynchronous operations. In the callbacks, first query ObjectMgr with object context to obtain the object pointer. If the pointer is NULL, then the object has been freed either by application or due to application unloading. In this case, drop the callback.

ObjectMgr API:

AEEObjectMgr_Register()	Registers the BREW object and returns the AEEObjectHandle
AEEObjectMgr_Unregister()	Unregisters the BREW object and the object handle is no more valid
AEEObjectMgr_GetObject()	Given object handle, returns the object pointer

NOTES:

- ObjectMgr is never released. It is automatically released by BREW when BREW exits.

List of Header files to be included

The following header file is required:

OEMObjectMgr.h

List of functions

Functions in this interface include:

AEEObjectMgr_GetObject()
AEEObjectMgr_Register()
AEEObjectMgr_Unregister()

The remainder of this section provides details for each function.

AEEObjectMgr_GetObject()

Description:

Given object handle, this function retrieves the object.

Prototype:

```
void * AEEObjectMgr_GetObject(AEEObjectHandle hObj);
```

Parameters:

hObj	Handle of the object

Return Value:

NULL: Object does not exist

Otherwise valid object pointer

Comments:

None.

See Also:

None.

Return to List of functions

AEEObjectMgr_Register()

Description:

This function registers a BREW object with the ObjectMgr. ObjectMgr returns an opaque context to the caller.

Prototype:

```
int AEEObjectMgr_Register
      (
      void * pObject,
      AEEObjectHandle * phObject
      );
```

Parameters:

pObject	Object to be registered
phObject	Handle to the object

Return Value:

SUCCESS: ObjectMgr cannot allocate handle.

Otherwise error.

See Also:

`AEEObjectHandle`.

Return to List of functions

AEEObjectMgr_Unregister()

Description:

This function unregisters the object and calls the caller-registered function, if any.

Prototype:

```
int AEEObjectMgr_Unregister(AEEObjectHandle hObj);
```

Parameters:

hObj	Handle of the object

Return Value:

SUCCESS: Unregister succeeded

Otherwise Unregister failed

Comments:

The handle should not be used after unregistration.

See Also:

None.

Return to List of functions

I3D Interface

This interface provides definitions for the 3D graphics engine. Application developers need to be aware of possible multiplication overflow when the range of individual triangles becomes too large in the rendering process.

The 3D engine is a fixed-point implementation for rendering three dimensional triangles. The graphics model coordinates are in 16.16 format. List of Header files to be included

The following header file is required:

AEE3D.h

List of functions

Functions in this interface include:

I3D_AddRef()
I3D_ApplyModelViewTransform()
I3D_CalcVertexArrayNormal()
I3D_CalcVertexArrayColor()
I3D_ClearFrameBuf()
I3D_GetClipRect()
I3D_GetCoordTransformMode()
I3D_GetCullingMode()
I3D_GetDestination()
I3D_GetFocalLength()
I3D_GetLight()
I3D_GetLightingMode()
I3D_GetMaterial()
I3D_GetModelViewTransform()
I3D_GetRenderMode()
I3D_GetScreenMapping()
I3D_GetTexture()
I3D_GetViewDepth()
I3D_PopMatrix()
I3D_PushMatrix()
I3D_QueryInterface()
I3D_RegisterEventNotify()
I3D_Release()
I3D_RenderTriangleFan()
I3D_RenderTriangles()

I3D_RenderTriangleStrip()
I3D_ResetZBuf()
I3D_SetClipRect()
I3D_SetCoordTransformMode()
I3D_SetCullingMode()
I3D_SetDestination()
I3D_SetFocalLength()
I3D_SetLight()
I3D_SetLightingMode()
I3D_SetMaterial()
I3D_SetModelViewTransform()
I3D_SetRenderMode()
I3D_SetScreenMapping()
I3D_SetTexture()
I3D_SetViewDepth()
I3D_StartFrame()

The remainder of this section provides details for each function.

I3D_AddRef()

Description:

This function increments the reference count for the 3D graphics class.

Prototype:

uint32 I3D_AddRef(I3D* pI3D);

Parameters:

pI3D	Pointer to a I3D interface whose reference count need to be incremented.

Return Value:

The updated reference count.

Comments:

None

See Also:

I3D_Release()

Return to the List of functions

I3D_ApplyModelViewTransform()

Description:

General API for doing model-view transformation given a pointer to a sequence of vertex coordinate vectors and number of vertices to be transformed. The Type of transformation is determined by the transformation mode. The output will update the location vectors in the vertex structure.

Prototype:

```
int I3D_ApplyModelViewTransform(
      I3D* pI3D,
      AEE3DTLVertex* pVertex,
      AEE3DPoint* pVertexBuf,
      uint32 n);
```

Parameters:

pI3D	:[in]	Pointer to I3D interface.
pVertex	:[out]	Vertex structure for 3D rendering.
pVertBuf	:[in]	Pointer to a sequence of coordinates (x,y,z).
n	:[in]	Number of vertices.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

AEE3DTLVertex

Return to the List of functions

I3D_CalcVertexArrayNormal()

Description:

Calculates the normal components (nx, ny, nz) in the vertex normal array.

Prototype:

```
int I3D_CalcVertexArrayNormal
     (
       I3D* pme,
       AEE3DPoint16* pVertexNormalArray,
       uint16* pVertexIndexArray,
       uint32 num_triangles,
       AEE3DPoint* pVertexArray,
       AEE3DPrimitiveType Prim_type
     )
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pVertexNormalArray	[in/out]	Pointer to vertex normal array
pVertexIndexArray	[in]	Pointer to the index vertex array
num_triangles	[in]	Number of triangles
pVertexArray	[in]	Pointer to array of vertices
Prim_type	[in]	Type of primitive the Vertex Index Array is referring to. (that is, triangles, triangle fan, and so on)

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

AEE3DPoint

AEE3DPoint16

AEE3DPrimitiveType

AEE3DTLVertex

Return to the List of functions

I3D_CalcVertexArrayColor()

Description:

Calculates the color (rgb) values in the AEETLVertex array.

Prototype:

```
int I3D_CalcVertexArrayColor
      (
      I3D* pI3D, AEE3DTLVertex* pVertexArray,
      uint16* pVertexIndexArray,
      uint32 num_vertices,
      AEE3DPoint16* pVertexNormalArray,
      AEE3DPrimitiveType Prim_type
      )
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pVertexArray	[in/out]	Pointer to a list of AEETLVertex
pVertexIndexArray	[in]	The index list of vertices
num_vertices	[in]	Number of vertices
pVertexNormalArray	[in]	The list of vertex normals
Prim_type	[in]	Type of primitive the pVertexList is referring to. (that is, Triangles, Triangle fan, and so on)

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

AEE3DPoint

AEE3DPoint16

AEE3DPrimitiveType

AEE3DTLVertex

Return to the List of functions

I3D_ClearFrameBuf()

Description:

Clear 3D frame buffer with background color

Prototype:

```
void I3D_ClearFrameBuf(I3D* pI3D);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

I3D_GetClipRect()

Description:

This function gets the clipping rectangle. All output parameters will be in terms of number of pixels.

Prototype:

```
int I3D_GetClipRect(I3D* pI3D, AEERect* pRect);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pRect	[out]	Pointer to a AEERect.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

pRect->x: The horizontal coordinate for the top left corner of the rectangle.

pRect->y: The vertical coordinate for the top left corner of the rectangle.

pRect->dx: The width of the clipping rectangle.

pRect->dy: The height of the clipping rectangle.

See Also:

I3D_SetClipRect()

AEERect

Return to the List of functions

I3D_GetCoordTransformMode()

Description:

NOTE: This function is currently not supported.

This function gets the coordinate transformation type from the graphics context. This will indicate which coordinate transformation will be applied before triangles are rendered.

Prototype:

```
int I3D_GetCoordTransformMode
      (
      I3D* pI3D,
      AEE3DCoordinateTransformType* pType
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pType	[out]	Pointer to a Coordinate Transformation type.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_SetCoordTransformMode()

AEE3DCoordinateTransformType.

Return to the List of functions

I3D_GetCullingMode()

Description:

This function gets the culling type. This will indicate which triangles should be discarded before they are mapped to the screen. By default, triangles with vertices arranged in counterclockwise rotation will be visible. A counterclockwise orientation indicates front-facing. A clockwise orientation is considered back facing.

Prototype:

```
int I3D_GetCullingMode(I3D* pI3D, AEE3DCullingType* pFacing);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pFacing	[out]	Pointer to culling type.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_SetCullingMode()

AEE3DCullingType

Return to the List of functions

I3D_GetDestination()

Description:

This function will get the Frame Buffer from I3D.

Prototype:

```
int I3D_GetDestination(I3D* pI3D,IBitmap** pFrameBuffer);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pFrameBuffer	[out]	IBitmap* pointer for 3D graphics framebuffer.

Return Value:

Return SUCCESS on success,

Otherwise returns error code.

Comments:

Currently, I3D only accepts 16 bpp device dependant bitmap (DDB). If pBitmap is not 16bpp DDB, it will return EUNSUPPORTED.

See Also:

I3D_SetDestination()

I3D_ClearFrameBuf()

I3D_ResetZBuf()

IBitmap Interface

Return to the List of functions

I3D_GetFocalLength()

Description:

This function gets the focal length. The output range will be within the depth of z-buffer (1-32767).

Prototype:

```
int I3D_GetFocalLength(I3D* pI3D,uint16* pFocalLength);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pFocalLength	[out]	Pointer to a Focal Length

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

The z-buffer is 16 bits in this release.

See Also:

I3D_SetFocalLength()

I3D_GetViewDepth()

Return to the List of functions

I3D_GetLight()

Description:

This function will get the lighting properties for the specified light type.

Prototype:

```
int I3D_GetLight(I3D* pI3D,AEE3DLightType type, AEE3DLight* light);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
type	[in]	The lighting type
light	[out]	The light values

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

AEE3DLight

AEE3DLightType

I3D_GetLightingMode()

I3D_SetLight()

Return to the List of functions

I3D_GetLightingMode()

Description:

This function will get the lighting mode values. This mode will indicate what lighting is enabled for rendering.

Prototype:

```
int I3D_GetLightingMode (I3D* pI3D, AEE3DLightingMode* pMode);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pMode	[out]	Pointer to current lighting mode

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

I3D_SetLight()

I3D_SetLightingMode()

AEE3DLight

AEE3DLightingMode

AEE3DLightType

Return to the List of functions

I3D_GetMaterial()

Description:

This function gets the current material.

Prototype:

```
int I3D_GetMaterial(I3D* pI3D, AEE3DMaterial* pMaterial);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pMaterial	[out]	Pointer to the material

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

AEE3DMaterial

I3D_SetMaterial()

Return to the List of functions

I3D_GetModelViewTransform()

Description:

This function gets the fixed point transformation matrix. It is copied from the ModelViewTransform member of the graphics context structure.

Prototype:

```
int I3D_GetModelViewTransform
      (
      I3D* pI3D,
      AEE3DTransformMatrix* pMatrix
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pMatrix	[out]	Pointer to a transformation matrix.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_SetModelViewTransform()

AEE3DTransformMatrix

Return to the List of functions

I3D_GetRenderMode()

Description:

This function gets the rendering type. It indicates how the triangle will be filled based on the surface color, texel and shading mode.

Prototype:

```
int I3D_GetRenderMode(I3D* pI3D, AEE3DRenderType* pType);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pType	[out]	Pointer to a Render type

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_SetRenderMode()

AEE3DRenderType

Return to the List of functions

I3D_GetScreenMapping()

Description:

This function gets the fixed-point screen mapping. The scaling part of the output is in Q12 format. The translation or shift part is in pixel units.

Prototype:

```
int I3D_GetScreenMapping
      (
      I3D* pI3D,
      int32* sx,
      int32* sy,
      int32* shftx,
      int32* shfty
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
sx	[out]	Pointer that will contain the X-scaling in Q12 format.
sy	[out]	Pointer that will contain the Y-scaling in Q12 format.
shftx	[out]	Pointer that will contain the X-shift in number of pixels.
shfty	[out]	Pointer that will contain the Y-shift in number of pixels.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

1 in Q12 = 4096

Also see:

Q12 Fixed Point Format

I3D_SetScreenMapping()

Return to the List of functions

I3D_GetTexture()

Description:

This function gets the texture of a specific type that is used in 3D rendering. A NULL output pointer indicates that no texture image of the given type is being used.

Prototype:

```
int I3D_GetTexture
      (
      I3D* pI3D,
      AEE3DTexture* pTexture
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pTexture	[out]	Pointer that contains the texture from the graphics context.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_SetTexture()

AEE3DTexture

AEE3DTextureType

Return to the List of functions

I3D_GetViewDepth()

Description:

This function gets the view depth in graphics context. Objects outside the view depth will not be rendered.

Prototype:

```
int I3D_GetViewDepth(I3D* pI3D, uint16* pZ0, uint16* pZ1);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pZ0	[out]	Pointer will contain value of the near view plane (1 <= z0 < z1)
pZ1	[out]	Pointer will contain value of the far view plane (z0 < z1 <=32767)

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

The z-buffer is 16 bit in this release.

See Also:

I3D_SetViewDepth()

Return to the List of functions

I3D_PopMatrix()

Description:

This function will pop the current Matrix off the stack.

Prototype:

```
int I3D_PopMatrix(I3D* pI3D)
```

Parameters:

pI3D	[in]	Pointer to I3D interface

Return Value:

Returns SUCCESS on success, otherwise returns error code.

Comments:

None

See Also:

I3D_PushMatrix()

AEE3DMatrixMode

Return to the List of functions

I3D_PushMatrix()

Description:

This function will push the current Matrix onto the stack.

Prototype:

int I3D_PushMatrix(I3D* pI3D).

Parameters:

pI3D	[in]	Pointer to I3D interface

Return Value:

Returns SUCCESS on success, otherwise returns error code.

Comments:

The default max stack size is 32

See Also:

I3D_PopMatrix()

AEE3DMatrixMode

Return to the List of functions

I3D_QueryInterface()

Description:

This function retrieves a pointer to what you query, according to the input class ID. This function can be used to query an extended version of I3D. This supports version compatibility.

Prototype:

```
int I3D_QueryInterface(I3D* pI3D, AEECLSID id, void** p);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
id	[in]	A globally unique id to identify the entity (interface or data) that we are trying to query.
p	[out]	Pointer to the data or interface that we want to retrieve. If the value passed back is NULL, the interface or data that we query are not available.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

If “p” passes back a NULL pointer, the data or interface that we query are not available.

See Also:

None

Return to the List of functions

I3D_RegisterEventNotify()

Description:

Registers an event callback function to be invoked whenever there is some new event or information to report about an asynchronous I3D operation. The application will not block on 3D rendering functions. The application developer needs to register a callback function to be notified when it is okay to start the next frame and to update the display. See the 3D events section to understand what each event means.

Prototype:

```
int I3D_RegisterEventNotify(I3D* pI3D, PFNEVENTNOTIFY pfn, void*
pUser);
```

Parameters:

pI3D	[in]	Pointer to I3D interface object
pfn	[in]	Pointer to the callback function to invoke to notify the application of events (NULL to deregister)
pUser	[in]	User data to be passed to callback when it is invoked

Return Value:

Always returns SUCCESS.

Comments:

An event callback function MUST be registered to properly use the 3D engine. It could be possible to update the display in the middle of rendering a frame if the user does not wait for the approriate event. Check the notified event to make sure it is okay to modify data or update the display. YOU MUST USE THIS FUNCTION IN EVERY APPLICATION.

Side Effects:

If not used the 3D application will not know when to update the display correctly.

See Also:

3D Events

AEE3DEventNotify

I3D_StartFrame()

Return to the List of functions

I3D_Release()

Description:

This function decrements the reference count for the 3D graphics object and does appropriate cleanup if the reference count reaches zero.

Prototype:

```
uint32 I3D_Release(I3D* pI3D);
```

Parameters:

pI3D	Pointer to the I3D interface whose reference count needs to be decremented.

Return Value:

The updated reference count

Comments:

None

See Also:

None

Return to the List of functions

I3D_RenderTriangleFan()

Description:

Render a triangle fan where each additional vertex defines a new triangle after the initial two vertices are defined. The first vertex is part of every triangle.

Prototype:

```
int I3D_RenderTriangleFan
      (
      I3D* pI3D,
      AEE3DTLVertex* pVertexArray,
      const uint16* pVertexIndexArray,
      uint32 num_of_triangles,
      uint32 num_of_vertices
      );
```

Parameters:

pI3D	Pointer to I3D interface
pVertexArray	List of vertices
pVertexIndexArray	Index list to the vertex list
num_of_triangles	Number of triangles
num_of_vertices	Number of vertices

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_RenderTriangles()

I3D_RenderTriangleStrip()

AEE3DTLVertex

Return to the List of functions

I3D_RenderTriangles()

Description:

Render one or more triangles.

Prototype:

```
int I3D_RenderTriangles
      (
      I3D* pI3D,
      AEE3DTLVertex* pVertexArray,
      const uin16* pVertexIndexArray,
      uint32 num_of_triangles,
      uint32 num_of_vertices
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pVertexArray	[in]	List of vertices
`pVertexIndexArray`	[in]	List of indicies into the vertex list. Every three of them in sequence define a triangle
`num_of_triangles`	[in]	Number of triangles
`num_of_vertices`	[in]	Number of vertices

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

I3D_RenderTriangleStrip()

I3D_RenderTriangleFan()

AEE3DTLVertex

Return to the List of functions

I3D_RenderTriangleStrip()

Description:

Renders a triangle strip, where each additional vertex defines a new triangle after the initial two vertices are defined.

Prototype:

```
int I3D_RenderTriangleStrip
      (
      I3D* pI3D,
      AEE3DTLVertex* pVertexArray,
      const uin16* pVertexIndexArray,
      uint32 num_of_triangles,
      uint32 num_of_vertices
```

);

Parameters:

pI3D	[in]	Pointer to I3D interface
pVertexArray	[in]	List of vertices
pVertexIndexArray	[in]	List of indices into the vertex list. Every three of them in sequence define a triangle
num_of_triangles	[in]	Number of triangles
num_of_vertices	[in]	Number of vertices

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

I3D_RenderTriangles()

I3D_RenderTriangleFan()

AEE3DTLVertex

Return to the List of functions

I3D_ResetZBuf()

Description:

This function resets the Z buffer to the highest depth value (65535 for 16 bit z-buffer).

Prototype:

```
void I3D_ResetZBuf(I3D* pI3D);
```

Parameters:

pme	[in]	Pointer to I3D interface.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

I3D_SetClipRect()

Description:

This function sets the clipping rectangle. All input parameters are in pixel units. Objects outside the clipping rectangle will not be rendered. The clipping rectangle is the full display area by default.

Prototype:

```
int I3D_SetClipRect(I3D* pI3D, const AEERect* pRect);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pRect	[in]	Pointer to a Clipping Rectangle.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

pRect->x: The horizontal coordinate for the top left corner of the clipping rectangle.

pRect->y: The vertical coordinate for the top left corner of the clipping rectangle.

pRect->dx: The width of the clipping rectangle.

pRect->dy: The height of the clipping rectangle.

See Also:

I3D_GetClipRect()

AEERect

Return to the List of functions

I3D_SetCoordTransformMode()

Description:

NOTE: This function is currently not supported.

Set coordinate transformation type. This will indicate which coordinate transformation will be applied before triangles are rendered. Model view transformation, projection, and screen mapping, all will be applied by default.

Prototype:

```
int I3D_SetCoordTransformMode
      (
      I3D* pI3D,
      AEE3DCoordinateTransformType type
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
type	[in]	Coordinate Transformation type.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_GetCoordTransformMode()

AEE3DCoordinateTransformType

Return to the List of functions

I3D_SetCullingMode()

Description:

Set FRONT or BACK face culling. This will indicate which triangles should be discarded before they are rendered. By default, triangles with vertices arranged in counter-clock wise rotation will be visible. A counter-clock wise rotation indicates front-facing. A clock-wise rotation is considered back-facing.

Prototype:

```
int I3D_SetCullingMode(I3D* pI3D, AEE3DCullingType facing);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
facing	[in]	either FRONT or BACK facing polygons will be culled.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

Default value is BACK_FACING

See Also:

I3D_GetCullingMode()

AEE3DCullingType

Return to the List of functions

I3D_SetDestination()

Description:

This function will set the Frame Buffer for I3D.

Prototype:

```
int I3D_SetDestination(I3D* pI3D, IBitmap* pFrameBuffer);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pFrameBuffer	[in]	IBitmap pointer for 3D graphics frame buffer

Return Value:

SUCCESS on success

Error code otherwise

Comments:

Currently, I3D only accepts 16 bpp device dependant bitmap (DDB). If pBitmap is not 16bpp DDB, it will return EUNSUPPORTED.

See Also:

I3D_GetDestination()

I3D_ClearFrameBuf()

I3D_ResetZBuf(),

IBitmap Interface,

Return to the List of functions

I3D_SetFocalLength()

Description:

Set focal length. The input range should be within the depth of z-buffer. Perspective division is not performed when focal length=0.

Prototype:

```
int I3D_SetFocalLength(I3D* pI3D,uint16 f);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
f	[in]	Focal length (1<= f<= 32767)

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_GetFocalLength()

Return to the List of functions

I3D_SetRenderMode()

Description:

This function sets the rendering type. It determines how the triangle will be filled based on the surface color, texel, and shading mode. A triangle will be shaded with a single color (flat-shading) by default.

Prototype:

```
int I3D_SetRenderMode(I3D* pI3D, AEE3DRenderType type);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
type	[in]	Render type

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_GetRenderMode()

AEE3DRenderType

Return to the List of functions

I3D_SetScreenMapping()

Description:

Set fixed-point screen mapping matrix. The scaling part of input is in Q12 Fixed Point Format. The translation part is in number of pixels. Input range is not checked

Prototype:

```
int I3D_SetScreenMapping
      (
      I3D* pI3D,
      int32 sx,
      int32 sy,
      int32 shftx,
      int32 shfty
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
sx	[in]	X-scaling in Q12 format (unit scaling = 4096)
sy	[in]	Y-scaling in Q12 format (unit scaling = 4096)
shftx	[in]	X-shift (in pixel).
shfty	[in]	Y-shift (in pixel).

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

Q12 Fixed Point Format

I3D_GetScreenMapping()

Return to the List of functions

I3D_SetTexture()

Description:

Set texture to be used in 3D rendering. A NULL input pointer indicates that the corresponding texture image will not be used. Texture is initialized to NULL by default.

Prototype:

```
int I3D_SetTexture
      (
      I3D* pI3D,
      AEE3DTexture* pTexture
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pTexture	[in]	Pointer to a texture object.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_GetTexture()

AEE3DTexture

AEE3DTextureType

Return to the List of functions

I3D_SetViewDepth()

Description:

Set view depth to be used in graphics context. Objects outside the view depth will not be rendered. The default view depth is the entire range of the z-buffer (0-65535).

Prototype:

```
int I3D_SetViewDepth(I3D* pI3D,uint16 z0,uint16 z1);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
z0	[in]	The near view plane (1 <= z0 < z1)
z1	[in]	The far view plane (z0 < z1 < 32767

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

The z-buffer is 16 bit. The default view depth is set to (1,2048).

See Also:

I3D_GetViewDepth()

Return to the List of functions

I3D_SetLight()

Description:

This function will set the lighting properties for the specified light type.

Prototype:

```
int I3D_SetLight(I3D* pI3D,AEE3DLight plight);
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
plight	[in]	Pointer to a light.

Return Value:

Return SUCCESS on success.

Otherwise returns error code.

Comments:

None

See Also:

AEE3DLight

I3D_SetLightingMode()

I3D_GetLight()

Return to the List of functions

I3D_SetLightingMode()

Description:

This function will set the lighting mode. This mode will indicate what lighting is enabled for rendering.

Prototype:

```
int I3D_SetLightingMode(I3D* pI3D,AEE3DLightingMode mode);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
mode	[in]	The lighting mode

Return Value:

Return SUCCESS on success

Otherwise returns error code

Comments:

None

See Also:

AEE3DLight

I3D_SetLight()()

I3D_GetLightingMode()

Return to the List of functions

I3D_SetMaterial()

Description:

This function sets the current material properties.

Prototype:

```
int I3D_SetMaterial(I3D* pI3D,AEE3DMaterial* pMaterial);
```

Parameters:

pI3D	[in]	Pointer to I3D interface
pMaterial	[in]	Pointer to the material

Return Value:

SUCCESS on success

Error code otherwise

Comments:

None

See Also:

AEE3DMaterial

I3D_GetMaterial()

Return to the List of functions

I3D_SetModelViewTransform()

Description:

Set fixed point transformation matrix. The input matrix is assumed to have the correct Q-factor. It is copied to the Model View Transform Matrix of the graphics context structure. Range of input is not checked.

Prototype:

```
int I3D_SetModelViewTransform
      (
      I3D* pI3D,
      const AEE3DTransformMatrix* pMatrix
      );
```

Parameters:

pI3D	[in]	Pointer to I3D interface.
pMatrix	[in]	Pointer to a Model-view transformation matrix.

Return Value:

SUCCESS on success.

Error code otherwise.

Comments:

None

See Also:

I3D_GetModelViewTransform()

AEE3DTransformMatrix

Return to the List of functions

I3D_StartFrame()

Description:

This function will not block but will run asynchronous. It will tell the 3D graphics engine to start processing and rendering the current frame.

Prototype:

```
int I3D_StartFrame(I3D* po);
```

Parameters:

po	[in]	Pointer to I3D interface.

Return Value:

SUCCESS: Command accepted.

EFAILED: General failure.

EBADPARM: Bad parm is passed.

ENOMEMORY: Not enough memory.

Comments:

MUST register an event callback function to properly use the 3D engine. It could be possible to update the display in the middle of rendering a frame if the user does not wait for the approriate event. Check the notified event to makes sure it is okay to modify data or update the display. YOU MUST USE THIS FUNCTION IN EVERY APPLICATION.

Side Effects:

If NOT called the 3D graphics engine will not know when to start rendering the current frame. THIS FUNCTION MUST BE CALLED TO START THE RENDERING ENGINE.

See Also:

I3D_RegisterEventNotify()

3D Event types

Return to the List of functions

I3DUtil Interface

This interface provides definitions for the 3D graphics Utility made available by the AEE to the application developers. This is a standard header file that must be included by all applications using I3DUtil interfaces.

These utility functions provide the developer with simplified operations to obtain the model-view transformation matrices and unit vectors.

NOTE: ALL I3DUtil functions are in Q12 format. Parameters are expected to be in Q12 fixed point format.

List of Header files to be included

The following header file is required:

AEE3DUtil.h

List of functions

Functions in this interface include:

I3DUtil_AddRef()
I3DUtil_cos()
I3DUtil_GetRotateMatrix()
I3DUtil_GetRotateVMatrix()
I3DUtil_GetViewTransformMatrix()
I3DUtil_GetUnitVector()
I3DUtil_MatrixMultiply()
I3DUtil_QueryInterface()
I3DUtil_Release()
I3DUtil_SetIdentityMatrix()
I3DUtil_SetTranslationMatrix()
I3DUtil_sin()
I3DUtil_sqrt()

The remainder of this section provides details for each function.

I3DUtil_AddRef()

Description:

This function increments the reference count of the I3DUtil Interface object, allowing the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 I3DUtil_AddRef(I3DUtil* pI3DUtil)
```

Parameters:

pI3DUtil	Pointer to the I3DUtil Interface object.

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

See Also:

I3DUtil_Release()

Return to the List of functions

I3DUtil_cos()

Description:

This function computes the cosine.

Prototype:

```
int32 I3DUtil_cos(I3DUtil* pI3DUtil, int32 angle)
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
angle	[in]	Q12 format (PI=2048).

Return Value:

Returns the cosine of the angle.

Comments:

None

See Also:

Q12 Fixed Point Format

Return to the List of functions

I3DUtil_GetRotateMatrix()

Description:

This function computes the transformation matrix for a rotation about x, y, or z-axis.

Prototype:

```
int I3DUtil_GetRotateMatrix
      (
      I3DUtil* pI3DUtil,
      int32 angle,
      AEE3DTransformMatrix* pMatrixOut,
      AEE3DRotateType axis
      )
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
angle	[in]	Rotation angle in Q12 format.
pMatrixOut	[out]	Pointer to the resulting transformation matrix.
axis	[in]	Axis to do rotation around.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

In respect to the angle (PI = 2048).

See Also:

Q12 Fixed Point Format

AEE3DTransformMatrix

AEE3DRotateType

Return to the List of functions

I3DUtil_GetRotateVMatrix()

Description:

This function computes the transformation matrix for a rotation about a given vector from origin.

Prototype:

```
int I3DUtil_GetRotateVMatrix
      (
      I3DUtil* pI3DUtil,
      const AEE3DPoint* pVector,
      int32 angle,
      AEE3DTransformMatrix* pMatrixOut
      );
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pVector	[in]	Pointer to vector originated from origin for the rotation in Q12 format.
angle	[in]	Rotation angle in Q12 format (PI=2048).
pMatrixOut	[out]	Pointer to the resulting transformation matrix.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q12 Fixed Point Format

AEE3DPoint

AEE3DTransformMatrix

Return to the List of functions

I3DUtil_GetViewTransformMatrix()

Description:

This function computes the fixed point transformation matrix for a given position, look-at-direction, and up-direction. Each directional vector is given as a 3D point or vector in Q12 format.

Prototype:

```
int I3DUtil_GetViewTransformMatrix
      (
      I3DUtil* pI3DUtil,
      const AEE3DPoint* pPosition,
      const AEE3DPoint* pLook,
      const AEE3DPoint* pUp,
      AEE3DTransformMatrix* pMatrixOut
      );
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pPosition	[in]	Pointer to positional vector of the viewer.
pLook	[in]	Pointer to directional vector of the viewing direction.
pUp	[in]	Pointer to directional vector for the up-direction.
pMatrixOut	[out]	Pointer to the resulting transformation matrix.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q12 Fixed Point Format

AEE3DPoint

AEE3DTransformMatrix

Return to the List of functions

I3DUtil_GetUnitVector()

Description:

This function computes the unit vector (dst) of a source vector (src). The resulting vector is Q12.

Prototype:

```
int I3DUtil_GetUnitVector
      (I3DUtil* pI3DUtil,
      const AEE3DPoint* pSrc,
      AEE3DPoint* pDst
      )
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pSrc	[in]	Pointer to source vector.
pDst	[out]	Pointer to resulting unit vector.

Return Value:

Return SUCCESS on success.

Otherwise returns error code.

Comments:

None

See Also:

AEE3DPoint

Q12 Fixed Point Format

Return to the List of functions

I3DUtil_MatrixMultiply()

Description:

This function multiplies two fixed point matrices. The multiplication is made using the equation:

```
MaxtrixOut = MatrixOut * MatrixIn
```

Prototype:

```
int I3DUtil_MatrixMultiply
      (
      I3DUtil* pI3DUtil,
      AEE3DTransformMatrix* pMatrixOut,
      const AEE3DTransformMatrix* pMatrixIn
      );
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pMatrixOut	[in/out]	Left multiplicant and the resulting matrix.
pMatrixIn	[in]	Right mutiplicant.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q12 Fixed Point Format

AEE3DTransformMatrix

Return to the List of functions

I3DUtil_QueryInterface()

Description:

This function asks an object for another API contract from the object in question.

Prototype:

```
int I3DUtil_QueryInterface
      (
      I3DUtil * pI3DUtil,
      AEECLSID idReq,
      void * * ppo
      )
```

Parameters:

pI3DUtil	[in]	Pointer to the I3DUtil Interface object.
idReq	[in]	Requested ClassID exposed by the object.
ppo	[in/out]	Returned object. Filled by this function.

Return Value:

SUCCESS, interface found.

ENOMEMORY, insufficient memory.

ECLASSNOTSUPPORT, requested interface is unsupported.

Comments:

- If *ppo is an interface pointer, then the pointer in ***ppo** is set to the new interface (with **refcount** incremented), or NULL if the ClassID is not supported by the object.
- If *ppo is a data structure pointer, then *ppo is set to the internal data represented by the classID or set to NULL if classID is not supported by the object.

See Also:

None

Return to the List of functions

I3DUtil_Release()

Description:

This function decrements the reference count of the I3DUtil Interface object. The object is freed from memory and is no longer valid when the reference count reaches 0 (zero).

Prototype:

```
uint32 I3DUtil_Release(I3DUtil * pI3DUtil)
```

Parameters:

pI3DUtil	Pointer to the I3DUtil Interface object.

Return Value:

Decremented reference count for the object.

0 (zero) if the object has been freed and is no longer valid.

Comments:

None

See Also:

I3DUtil_AddRef()

Return to the List of functions

I3DUtil_SetIdentityMatrix()

Description:

Set the rotation part (3x3) of the transformation matrix to the identity matrix in Q12 format.

The transformation matrix will set to:

{ 4096, 0, 0, 0,
0, 4096, 0, 0,
0, 0, 4096, 0 }

Prototype:

```
int I3DUtil_SetIdentityMatrix
      (
      I3DUtil* pI3DUtil,
      AEE3DTransformMatrix* pMatrixOut
      )
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pMatrixOut	[out]	Pointer to the resulting matrix.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q3D File Format

AEE3DTransformMatrix

Return to the List of functions

I3DUtil_SetTranslationMatrix()

Description:

This function sets the translation part of the transformation matrix for a given translation vector in Q12 Fixed Point Format.

Prototype:

```
int I3DUtil_SetTranslationMatrix
      (
      I3DUtil* pI3DUtil,
      AEE3DPoint* pVector,
      AEE3DTransformMatrix* pMatrixOut
      );
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
pVector	[in]	Pointer to the translation vector in Q12 Fixed Point Format.
pMatrixOut	[in&out]	Pointer to the resulting matrix.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q12 Fixed Point Format

AEE3DPoint

AEE3DTransformMatrix

Return to the List of functions

I3DUtil_sin()

Description:

This function computes the sine.

Prototype:

```
int32 I3DUtil_sin(I3DUtil* pI3DUtil, int32 angle)
```

Parameters:

pI3DUtil	[in]	Pointer to I3DUtil interface.
angle	[in]	Q12 format (PI=2048).

Return Value:

Returns the sine of the angle.

Comments:

None

See Also:

Q12 Fixed Point Format

Return to the List of functions

I3DUtil_sqrt()

Description:

This function computes the square root of the input parameter number.

Prototype:

```
uint32 I3DUtil_sqrt(I3DUtil* pI3DUtil, uint32 number);
```

Parameters:

pI3DUtil	Pointer to I3DUtil interface.
number	input parameter

Return Value:

Returns the square root of the number.

Comments:

None

See Also:

None

Return to the List of functions

I3DModel Interface

This interface provides definitions for 3D graphics models made available to the application developers. A standard header file must be included by all applications using the I3DModel interfaces.

The I3DModel provides high-level APIs for users to draw a structured group of triangles.

The reserved unique ClassID for I3DModel is defined to be AEECLSID_3DMODEL.

List of Header files to be included

The following header files are required for I3DModel

AEE3DModel.h

List of functions

Functions in this interface include:

I3DModel_AddRef()
I3DModel_Draw()
I3DModel_GetModelData()
I3DModel_GetModelVertexList()
I3DModel_Load()
I3DModel_QueryInterface()
I3DModel_Release()
I3DModel_SetTextureTbl()
I3DModel_SetSegmentMVT()

The remainder of this section provides details for each function.

I3DModel_AddRef()

Description:

This function increments the reference count of the I3DModel Interface object, allowing the object to be shared by multiple callers. The object is freed when the reference count reaches 0 (zero).

Prototype:

```
uint32 I3DModel_AddRef(I3DModel* pI3DModel)
```

Parameters:

pI3DModel	Pointer to the I3DModel Interface object.

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count.

See Also:

I3DModel_Release()

Return to the List of functions

I3DModel_Draw()

Description:

This function will draw a 3D model. The 3D model must be in the 3D model structure.

Prototype:

```
int I3DModel_Draw(I3DModel* pI3DModel, I3D* pI3D);
```

Parameters:

pI3DModel	[in]	Pointer to I3DModel interface.
pI3D	[in]	Pointer to I3D interface.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

The lower-level function called in this routine is I3D_RenderTriangles. If lighting mode is set to NONE then you can replace the AEE3DVertex list contained in the model structure with the AEE3DTLVertex list, in which color has been defined per vertex.

See Also:

Q3D File Format

AEE3DVertex

AEE3DModelData

Return to the List of functions

I3DModel_GetModelData()

Description:

This function will get the model information for an I3DModel instance.

Prototype:

```
int I3DModel_GetModelData
      (
      I3DModel* pI3DModel,
      AEE3DModelData** pModel_out
      )
```

Parameters:

pI3DModel	[in]	Pointer to I3DModel interface.
pModel_out	[out]	Address of a pointer to a model structure.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

AEE3DModelData

Q3D File Format

Return to the List of functions

I3DModel_GetModelVertexList()

Description:

This function will get the vertex list stored in an I3DModel instance.

Prototype:

```
int I3DModel_GetModelVertexList(I3DModel* pI3DModel, AEE3DVertex**
pVertexList_out);
```

Parameters:

pI3DModel	[in]	Pointer to I3DModel interface.
pVertexList_out	[out]	Address of a pointer to a vertex list.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

None

See Also:

Q3D File Format

AEE3DVertex

Return to the List of functions

I3DModel_Load()

Description:

This function will load a 3D model. The 3D model must be in the Q3D file format.

Prototype:

```
int I3DModel_Load(I3DModel* pI3DModel,const char* pFilename);
```

Parameters:

pI3DModel	[in]	Pointer to I3DModel interface.
pFilename	[in]	File name string.

Return Value:

SUCCESS on success.

Error code, otherwise.

Comments:

The internal model is only allocated when I3DModel_Load is called.

See Also:

Q3D File Format

AEE3DModelData

AEE3DVertex

I3DModel_GetModelData()

I3DModel_GetModelVertexList()

Return to the List of functions

I3DModel_QueryInterface()

Description:

This function asks an object for another API contract from the object in question.

Prototype:

```
int I3DModel_QueryInterface
      (
      I3DModel* pI3DModel,
      AEECLSID idReq,
      void** ppo
      )
```

Parameters:

pI3DModel	[in]	Pointer to the I3DModel Interface object.
idReq	[in]	Requested ClassID exposed by the object.
ppo	[in/out]	Returned object. Filled by this function.

Return Value:

SUCCESS, interface found.

ENOMEMORY, insufficient memory.

ECLASSNOTSUPPORT, requested interface is unsupported.

Comments:

- If *ppo is an interface pointer, then the pointer in ***ppo** is set to the new interface (with **refcount** incremented), or NULL if the ClassID is not supported by the object.
- If *ppo is a data structure pointer, then *ppo is set to the internal data represented by the classID or set to NULL if classID is not supported by the object.

See Also:

None

Return to the List of functions

I3DModel_Release()

Description:

This function decrements the reference count of the I3DModel Interface object. The object is freed from memory and is no longer valid when the reference count reaches 0 (zero).

Prototype:

```
uint32 I3DModel_Release(I3DModel* pI3DModel)
```

Parameters:

pI3DModel	Pointer to the I3DModel Interface object.

Return Value:

Decremented reference count for the object.

0 (zero), If the object has been freed and is no longer valid.

Comments:

None

See Also:

I3DModel_AddRef()

Return to the List of functions

I3DModel_SetTextureTbl()

Description:

This function will set the texture table in a model.

Prototype:

```
int I3DModel_SetTextureTbl
      (
      13DModel* p13DModel,
      AEE3DTexture* pTexture,
      unit16 index
      )
```

Parameters:

p13DModel	[in]	Pointer to I3DModel instance
pTexture	[in]	Pointer to a texture
index	[in]	Index for the model's texture table

Return Value:

SUCCESS, EFAILED.

Comments:

None

See Also:

AEE3DTexture

Return to the List of functions

I3DModel_SetSegmentMVT()

Description:

This function will set the texture table in a model.

Prototype:

```
int I3DModel_SetSegmentMVT(I3DModel* p13DModel, AEE3DTransformMatrix*
trans, int16 index)
```

Parameters:

p13DModel	[in]	Pointer to I3DModel instance
trans	[in]	Pointer to a transformation matrix
index	[in]	Segment index -1 will set all segments using this transformation

Return Value:

SUCCESS, EFAILED.

Comments:

None

See Also:

AEE3DModelData

AEE3DTransformMatrix

Return to the List of functions

IBitmap Interface

This interface manipulate bitmaps. Each IBitmap instance represents a bitmap. IBitmap is an interface with multiple implementations. Device-independent bitmaps (DIB) created with IDISPLAY_CreateDIBitmap() are one class, and bitmaps that represent the handset's display are another class. While both classes implement the IBitmap interface, each has different capabilities. Both DIBs and display bitmaps can be used in blit operations. Display bitmaps support all drawing operations, but DIBs do not generally support drawing, and return EUNSUPPORTED from most functions. All functions that return an error code can potentially return EUNSUPPORTED. Users should be prepared for all types of error codes.

List of Header files to be included

The following header file is required:

AEEBitmap.h

List of functions

Functions in this interface include:

IBITMAP_AddRef()
IBITMAP_BltIn()
IBITMAP_BltOut()
IBITMAP_CreateCompatibleBitmap()
IBITMAP_DrawHScanline()
IBITMAP_DrawPixel()
IBITMAP_FillRect()
IBITMAP_GetInfo()
IBITMAP_GetPixel()
IBITMAP_GetTransparencyColor()
IBITMAP_NativeToRGB()
IBITMAP_QueryInterface()
IBITMAP_Release()
IBITMAP_RGBToNative()
IBITMAP_SetPixels()
IBITMAP_SetTransparencyColor()

The remainder of this section provides details for each function.

IBITMAP_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IBITMAP_Release()

Return to the List of functions.

IBITMAP_BltIn()

Description:

This function performs a bit-block transfer of the data corresponding to a rectangle of pixels from the specified source bitmap into this bitmap. Each pixel in the source is associated with a corresponding pixel in the destination. A logical operation is performed on each pair of source and destination pixels, and the result is written over the destination pixel.

Prototype:

```
int IBITMAP_BltIn
      (
      IBitmap * pIBitmap,
      int xDst,
      int yDst,
      int dx,
      int dy,
      IBitmap *pSrc,
      int xSrc,
      int ySrc,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap interface object into which the bit-block transfer needs to be done.
xDst	[in]	Specifies the x-coordinate of the upper left corner of the destination rectangular area.
yDst	[in]	Specifies the y-coordinate of the upper left corner of the destination rectangular area.
dx	[in]	Specifies the width of the destination and source rectangles. Negative values are treated as zero.
dy	[in]	Specifies the height of the destination and source rectangles. Negative values are treated as zero.
pSrc	[in]	Pointer to another IBitmap interface that represents the source bitmap.
xSrc	[in]	Specifies the x-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin.
ySrc	[in]	Specifies the y-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin.
rop	[in]	Specifies the raster operation that is used while doing the bit-block transfer.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EUNSUPPORTED, if a non-supported raster operation is specified.

Other error code, if the operation is not supported. This might be due to the format of the source bitmap or the type of raster operation that was requested or a combination of the two.

Comments:

When either **dx** or **dy** is negative, nothing is written to the destination bitmap. The rectangles are treated as empty.

It is legal for all or part of the source or destination rectangles that fall outside the corresponding bitmap bounds, to include negative coordinates.

When parts of the source or destination rectangles exceed a bitmap's bounds, they are clipped. Clipping will not affect the mapping from source to destination of any unclipped portions, and will not result in an error code, even when everything is clipped.

When the width and height of the source bitmap are not known, to blit the entire bitmap, very large values can be supplied for **dx** and **dy**, and clipping will limit the rectangle to the size of the source.

The source bitmap may or may not be the same format as the destination bitmap, but not all source formats are necessarily supported.

See Also:

AEERasterOp

IBITMAP_BltOut()

Return to the List of functions

IBITMAP_BltOut()

Description:

Perform a bit-block transfer from this bitmap into a specified destination bitmap. Users would not normally call this function directly. Instead, the destination bitmap's IBITMAP_BltIn() member function should be called, because that will succeed in more cases.

Prototype:

```
int IBITMAP_BltOut
      (
      IBitmap * pIBitmap,
      int xDst,
      int yDst,
      int dx,
      int dy,
      IBitmap *pDst,
      int xSrc,
      int ySrc,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface object from which the bit-block transfer needs to be done.
xDst	[in]	Specifies the x-coordinate of the upper left corner of the destination rectangular area.
yDst	[in]	Specifies the y-coordinate of the upper left corner of the destination rectangular area.
dx	[in]	Specifies the width of the destination rectangle.
dy	[in]	Specifies the height of the destination rectangle.
pDst	[in]	Pointer to another IBitmap interface that represents the destination bitmap.
xSrc	[in]	Specifies the x-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin.
ySrc	[in]	Specifies the y-coordinate of the upper left corner of the source bitmap from where the bit-block transfer must begin.
rop	[in]	Specifies the raster operation that is used while doing the bit-block transfer.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EUNSUPPORTED, if a non-supported raster operation.

Other implementation-specific error codes

Comments:

There is no need to call this function except from the implementation of IBITMAP_BltIn(). IBITMAP_BltOut() exists to help IBITMAP_BltIn(). IBITMAP_BltIn() will delegate to the source bitmap's IBITMAP_BltOut() when the destination does not support the operation and the bitmaps are of different classes. In this manner, both classes have the opportunity to perform the operation, and it will succeed as long as either class supports it. Note that IBITMAP_BltOut() cannot delegate likewise to IBITMAP_BltIn(), because that would lead to infinite recursion.

See Also:

AEERasterOp

IBITMAP_BltIn()

Return to the List of functions

IBITMAP_CreateCompatibleBitmap()

Description:

This function creates a new bitmap compatible with this bitmap interface (the first parameter). Compatible means having equivalent pixel sizes and the same mapping between pixel values (native colors) and RGB values. Width and height do not affect compatibility. A blit operation involving two compatible bitmaps is reasonably fast, because it does not need to perform complex translations of pixel data. Being a common case, this type of operation is typically highly optimized. Also, copies between compatible bitmaps do not result in the loss of any color information. A compatible bitmap will generally support all the same drawing operations that the original bitmap supports.

Prototype:

```
int IBITMAP_CreateCompatibleBitmap
      (
      IBITMAP *pIBitmap,
      IBitmap **ppIBitmap,
      uint16 w,
      uint16 h
      )
```

Parameters:

pIBitmap	[in]	Pointer to the current bitmap interface.
ppIBitmap	[out]	Pointer to the interface of new bitmap that has the same format of the current bitmap.
w	[in]	Width of the new bitmap.
h	[in]	Height of the new bitmap.

Return Value:

SUCCESS, if the function executed correctly.

Error code, if otherwise.

ENOMEMORY, if there was not enough memory for the operation.

Other implementation-specific error codes.

Comments:

The created bitmap inherits the transparency color from the bitmap it was created from.

See Also:

None

Return to the List of functions

IBITMAP_DrawHScanline()

Description:

This function draws a horizontal line.

Prototype:

```
int IBITMAP_DrawHScanline
      (
      IBitmap *pIBitmap,
      unsigned y,
      unsigned xMin,
      unsigned xMax,
      NativeColor color,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface object to be used to draw the horizontal line.
y	[in]	Y-coordinate of the line.
xMin	[in]	X-coordinate of the left end of the line.
xMax	[in]	X-coordinate of the right end of the line.
color	[in]	Specifies the color to be used to draw the line. This is a native color, which is obtained using IBITMAP_RGBToNative().
rop	[in]	Specifies the raster operation that is used to draw the line. Only AEE_RO_COPY and AEE_RO_XOR are valid.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EBADPARM, if raster operation is invalid.

Other implementation-specific error codes

Comments:

None

See Also:

AEERasterOp

Return to the List of functions

IBITMAP_DrawPixel()

Description:

This function draws a single pixel in the bitmap.

Prototype:

```
int IBITMAP_DrawPixel
      (
      IBitmap *pIBitmap,
      unsigned x,
      unsigned y,
      NativeColor color,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface to which the pixel will be drawn.
x	[in]	X-coordinate of the pixel.
y	[in]	Y-coordinate of the pixel.
color	[in]	Specifies the color to be used to draw the pixel. This is a native color, which is obtained using IBITMAP_RGBToNative().
rop	[in]	Specifies the raster operation that is used to draw the pixel. Only AEE_RO_COPY and AEE_RO_XOR are valid.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EBADPARM, if raster operation is invalid.

Other implementation-specific error codes

Comments:

If x or y is outside the bounds of the bitmap, nothing is drawn, and SUCCESS is returned.

See Also:

AEERasterOp

IBITMAP_GetPixel()

IBITMAP_SetPixels()

Return to the List of functions

IBITMAP_FillRect()

Description:

This function draws a solid rectangle of the specified color.

Prototype:

```
int IBITMAP_FillRect
      (
      IBitmap *pIBitmap,
      const AEERect *prc,
      NativeColor color,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface object to be used to fill the rectangle.
prc	[in]	A valid pointer to a rectangle that needs to be filled with the specified color.
color	[in]	Specifies the color to be used to fill the rectangle. This is a native color, which is obtained using IBITMAP_RGBToNative().
rop	[in]	Specifies the raster operation that is used while drawing the rectangle. Only AEE_RO_COPY and AEE_RO_XOR are valid.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EBADPARM, if raster operation is invalid.

Other implementation-specific error codes

Comments:

The **prc** parameter must be a valid pointer. Any portions of the rectangle that fall outside the bitmap's bounds are silently ignored (no error is generated).

See Also:

AEERasterOp

Return to the List of functions

IBITMAP_GetInfo()

Description:

This function retrieves the dimension of the bitmap.

Prototype:

```
int IBITMAP_GetInfo
      (
      IBitmap * pIBitmap,
      AEEBitmapInfo * pinfo,
      int nSize
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface.
pinfo	[out]	Pointer to AEEBitmapInfo, which contains the width, height, color depth, and so on.
nSize	[in]	Set to the sizeof(AEEBitmapInfo) in the current version. The AEEBitmapInfo structure may grow over time. This field allows backward compatibility.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EUNSUPPORTED, if the size is not recognized by the bitmap

Comments:

This function should always succeed when **nSize** is equal to sizeof(AEEBitmapInfo), and when **pinfo** is a valid pointer.

See Also:

AEEBitmapInfo

Return to the List of functions

IBITMAP_GetPixel()

Description:

This function retrieves the value of the specified pixel.

Prototype:

```
int IBITMAP_GetPixel
      (
      IBitmap *pIBitmap,
      unsigned x,
      unsigned y,
      NativeColor *pColor
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap interface from which the pixel value is retrieved.
x	[in]	X-coordinate of the pixel.
y	[in]	Y-coordinate of the pixel.
pColor	[out]	Color of the specified pixel.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EBADPARM, if the coordinate is out of range.

EUNSUPPORTED, if the operation is not supported.

Other implementation specific-error codes.

Comments:

None

See Also:

IBITMAP_SetPixels()

Return to the List of functions

IBITMAP_GetTransparencyColor()

Description:

This function gets the current transparency color of the bitmap. This is used when this bitmap is the source bitmap of a transparent bit blit operation.

Prototype:

```
int IBITMAP_GetTranparencyColor
      (
      IBITMAP *pMe,
      NativeColor *pColor
      )
```

Parameters:

pMe	[in]	Pointer to the current bitmap interface.
pColor	[out]	Transparency color.

Return Value:

SUCCESS, if the function executed correctly.

Error code, if otherwise.

EBADPARM, if **pColor** is NULL.

EUNSUPPORTED, if the operation is not supported.

Other implementation-specific error codes

Comments:

The transparency color is a NativeColor value, not an RGBVAL.

See Also:

IBITMAP_BltIn()

IBITMAP_BltOut()

IBITMAP_SetTransparencyColor()

Return to the List of functions

IBITMAP_NativeToRGB()

Description:

This function obtains the RGB definition of a NativeColor value, in RGBVAL format.

Each valid NativeColor corresponds to an RGB value. The mapping of NativeColor values is a property of the bitmap.

Prototype:

```
RGBVAL IBITMAP_NativeToRGB(IBitmap *pIBitmap, NativeColor clr)
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface.
clr	[in]	Native color value.

Return Value:

The corresponding RGBVAL.

Comments:

If the NativeColor provided is not associated with a specific RGB value, the return value from this function is undefined.

See Also:

NativeColor

RGBVAL

IBITMAP_RGBToNative()

Return to the List of functions

IBITMAP_QueryInterface()

Description:

This function retrieves a pointer to an interface conforming to the definition of the specified ClassID. This can be used to query for extended functionality, like future versions or proprietary features. Upon a successful query, the interface is returned. The caller is responsible for calling Release() at some point in the future. One exception is when the pointer returned is not an interface pointer. In that case, the memory will share the lifetime of the object being queried, and the returned pointer will not be used to free or release the object.

Prototype:

```
int IBITMAP_QueryInterface(IBitmap *pIBitmap, AEECLSID id, void **p);
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap interface.
id	[in]	A globally unique id to identify the entity (interface or data) being queried.
p	[out]	Pointer to the data or interface to be retrieved. If the value passed back is NULL, the queried interface or data is not available.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

Comments:

Many bitmaps support the IDIB interface (class id: AEECLSID_DIB), which contains public data that allows the caller to directly access the bitmap data. Only bitmaps with internal formats that conform to one of the documented IDIB formats can support IDIB. As with other interfaces, the IDIB must be Released when the user finishes with it.

On failure, ***p** should be set to NULL, but it is good form to explicitly set ***p** to NULL before calling IBITMAP_QueryInterface().

See Also:

None

Return to the List of functions

IBITMAP_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IBITMAP_AddRef()

Return to the List of functions.

IBITMAP_RGBToNative()

Description:

This function converts an RGBVAL value into a native color value (pixel value). Native color values are the values stored in the pixel array; the mapping between native and RGB values is a property of the bitmap. If no NativeColor corresponds exactly to the specified RGBVAL, a close match is returned. This function is not required to return the closest match, and for performance reasons a close match (but not the closest) may be returned.

Prototype:

```
NativeColor IBITMAP_RGBToNative(IBitmap *pIBitmap, RGBVAL RGBColor)
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface.
RGBColor	[in]	RGB value to be converted from. Only true RGB values are supported. Color table indices are not supported.

Return Value:

The corresponding native color value.

Comments:

If the bitmaps lack palette information, the result is undefined.

See Also:

RGBVAL

IBITMAP_NativeToRGB()

Return to the List of functions

IBITMAP_SetPixels()

Description:

This function draws multiple pixels with the same color.

Prototype:

```
int IBITMAP_SetPixels
      (
      IBitmap *pIBitmap,
      unsigned cnt,
      AEEPoint *pPoint,
      NativeColor color,
      AEERasterOp rop
      )
```

Parameters:

pIBitmap	[in]	Pointer to the IBitmap Interface object to which the pixels are drawn.
cnt	[in]	Number of pixels.
pPoint	[in]	Array of 2D points.
color	[in]	Specifies the color to be used to draw the pixels. This is a native color, which is obtained using IBITMAP_RGBToNative().
rop	[in]	Specifies the raster operation that is used to draw the pixel.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

EBADPARM, if raster operation is invalid.

Other implementation-specific error codes

Comments:

Only AEE_RO_COPY and AEE_RO_XOR are valid raster operations.

Any other value passed as a parameter is treated like AEE_RO_COPY. In the current version, this function returns EBADPARM.

See Also:

AEERasterOp

IBITMAP_GetPixel()

Return to the List of functions

IBITMAP_SetTransparencyColor()

Description:

This function sets the transparency color of the bitmap. This is used when this bitmap is the source bitmap of a transparent bit blit operation. For pixels that contain this NativeColor, the corresponding destination pixel is unaffected.

Prototype:

```
int IBITMAP_SetTranparencyColor(IBITMAP *pMe, NativeColor color);
```

Parameters:

pMe	[in]	Pointer to the current bitmap interface.
color	[in]	Color to make transparent.

Return Value:

SUCCESS, if the function executed correctly.

Error code, if otherwise.

Or other implementation-specific error codes

Comments:

None

See Also:

NativeColor

IBITMAP_BltIn()

IBITMAP_BltOut()

IBITMAP_GetTransparencyColor()

Return to the List of functions

IBitmapCtl Interface

The IBitmapCtl interface is used to enable and restrict access to the device bitmap. It is used internally by BREW and is not available to applications.

The purpose is to allow BREW to control which application is allowed to write to the screen. IBitmapCtl is an extension to the IBitmap interface, and is obtained through IBITMAP_QueryInterface() with a class ID of AEECLSID_BITMAPCTL.

It is a requirement of the OEM that this interface be implemented for the device bitmap. This interface does not need to be implemented for any other bitmaps.

List of Header files to be included

The following header file is required:

OEMDisp.h

List of functions

Functions in this interface include:

IBITMAPCTL_AddRef()
IBITMAPCTL_Enable()
IBITMAPCTL_NotifyRelease()
IBITMAPCTL_QueryInterface()
IBITMAPCTL_Release()
IBITMAPCTL_Restrict()

The remainder of this section provides details for each function.

IBITMAPCTL_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IBITMAPCTL_Release()

Return to the List of functions

IBITMAPCTL_Enable()

Description:

This function is used to enable or disable drawing operations to the entire bitmap.

Prototype:

```
int IBITMAPCTL_Enable(IBitmapCtl *pIBitmapCtl, boolean bOn);
```

Parameters:

pIBitmapCtl	[in]	Pointer to the IBitmapCtl interface.
bOn	[in]	TRUE if drawing will be allowed, otherwise FALSE.

Return Value:

SUCCESS, or error code.

Comments:

None.

See Also:

IBITMAPCTL_Restrict()

Return to the List of functions

IBITMAPCTL_NotifyRelease()

Description:

This function is used to register for notification of release of the last reference to this bitmap.

Prototype:

```
int IBITMAPCTL_NotifyRelease(IBitmapCtl *po, AEECallback *pcb);
```

Parameters:

po	Pointer to IBitmapCtl interface.
pcb	Pointer to callback stucture. This callback will be triggered when the last reference to this bitmap is released.

Return Value:

SUCCESS is returned if function executed correctly.

EBADPARM if pcb is NULL.

Other values may be returned if other errors occur.

Comments:

None

See Also:

None

Return to the List of functions

IBITMAPCTL_QueryInterface()

Description:

This function is inherited from IQI_QueryInterface().

See Also:

Return to the List of functions

IBITMAPCTL_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IBITMAPCTL_AddRef()

Return to the List of functions

IBITMAPCTL_Restrict()

Description:

This function restricts drawing operations to a portion of the bitmap.

Prototype:

```
int IBITMAPCTL_Restrict(IBitmapCtl *pIBitmapCtl, AEERect *prc);
```

Parameters:

pIBitmapCtl	[in]	Pointer to IBitmapCtl interface.
prc	[in]	Pointer to rectangle that that specifies where drawing is allowed.

Return Value:

SUCCESS is returned if function executed correctly.

EBADPARM if prc is NULL.

EUNSUPPORTED if function is not implemented.

Comments:

None.

See Also:

IBITMAPCTL_Enable()

Return to the List of functions

ICallHistory Interface

Description:

The ICallHistory interface provides applications with the ability to add an entry to the Call History list, remove the entries, enumerate the entries in the list as well as get the number of entries in the list and the maximum list size.

Predefined Field Types:

An entry into the CallHistory is made up of one or more fields. Each field is composed of a ClassID and FieldID which together indicate the type of data contained by the field. BREW has several pre-defined field types that represent information commonly associated with voice calls. For all of the predefined fields, the ClsId parameter should be set to 0. User applications should not attempt to overload the predefined field types, but rather make an extended type assocaited with their own ClassID and possibly approximate the value with the predefined equivalent), so that applications that are not aware of the extended data type can continue to operate correctly.

Predefined Fields:

AEECALLHISTORY_FIELD_CALL_TYPE (uint16):

The call type must be one of the following values:

- AEECALLHISTORY_CALL_TYPE_TO
- AEECALLHISTORY_CALL_TYPE_FROM
- AEECALLHISTORY_CALL_TYPE_MISSED

AEECALLHISTORY_FIELD_NUMBER_TYPE (uint16):

The number type must be one of the following values:

- AEECALLHISTORY_NUMBER_TYPE_INTERNATIONAL
- AEECALLHISTORY_NUMBER_TYPE_NATIONAL
- AEECALLHISTORY_NUMBER_TYPE_NETWORK
- AEECALLHISTORY_NUMBER_TYPE_SUBSCRIBER
- AEECALLHISTORY_NUMBER_TYPE_ABREVIATED
- AEECALLHISTORY_NUMBER_TYPE_QCHAT

AEECALLHISTORY_FIELD_NUMBER_PLAN (uint16):

The number plan must be one of the following values:

- AEECALLHISTORY_NUMBER_PLAN_TELEPHONY
- AEECALLHISTORY_NUMBER_PLAN_DATA
- AEECALLHISTORY_NUMBER_PLAN_TELEX
- AEECALLHISTORY_NUMBER_PLAN_PRIVATE

AEECALLHISTORY_FIELD_DATE_TIME (uint32):

The date and time of the origination of the phone call expressed as the number of seconds since Jan 6, 1980 00:00:00 GMT. This is the same format as the GETTIMESECONDS() helper function.

AEECALLHISTORY_FIELD_CALL_DURATION (uint32):

The duration of the call, in seconds

AEECALLHISTORY_FIELD_NUMBER (ASCII unterminated string):

This is the number dialed. Valid characters for the string include the ASCII digits '0' - '9', '#', '*', and ','. The comma represents and OEM-defined soft pause. OEMs are free to truncate the number field as needed by their implementations.

AEECALLHISTORY_FIELD_NAME (Unicode unterminated string):

This is the text description of the call history entry. For custom call types like entries without a AEECALLHISTORY_FIELD_NUMBER, this value will be displayed to describe the call.

NOTE: OEMs may require certain pre-defined fields in an entry before that entry can be added or updated to the Call History. Check OEM documentation to determine which, if any, fields are required.

NOTE: OEMs may not store all field types. If an entry to be added/updated contains field types that the OEM cannot store, the function will return SUCCESS, and only those fields that are supported by the OEM will be available to be retrieved via the enumeration functions. .

List of Header files to be included

The following header file is required:

AEECallHistory.h

List of functions

Functions in this interface include:

ICALLHISTORY_Clear()
ICALLHISTORY_AddEntry()
ICALLHISTORY_EnumInit()
ICALLHISTORY_EnumNext()
ICALLHISTORY_UpdateEntry()

The remainder of this section provides details for each function.

ICALLHISTORY_Clear()

Description:

This deletes all entries from the Call History list in the current storage.

Prototype:

```
int ICALLHISTORY_Clear(ICallHistory *pich)
```

Parameters:

pich	Pointer to an ICallHistory interface object

Return Value:

SUCCESS if successful

EFAILED or other applicable BREW error if failed to delete entries

Comments:

None

See Also:

None

Return to the List of functions

ICALLHISTORY_AddEntry()

Description:

This function adds a new entry to the Call History list in the current set storage. If the maximum number of entries is reached, the will delete the oldest entry in the list, and add the new entry at the top of the list.

Prototype:

```
int ICALLHISTORY_AddEntry
      (
      ICallHistory *pich,
      const AEECallHistoryEntry *pche
      );
```

Parameters:

pich	Pointer to an ICallHistory interface object
pche	new entry

Return Value:

SUCCESS if successful

EBADPARM if field data is not valid for field type or invalid AEECallHistory entry structure.

EFAILED or other applicable BREW error if failed to delete entries

Comments:

The memory specified in pche is copied by the ICALLHISTORY implementation, and need not be maintained by the caller after the call to AddEntry()

See Also:

AEECallHistoryEntry

Return to the List of functions

ICALLHISTORY_EnumInit()

Description:

This initializes/resets the enumeration in the Call History list.

Prototype:

```
int ICALLHISTORY_EnumInit(ICallHistory *pich)
```

Parameters:

pich	Pointer to an ICallHistory interface object

Return Value:

SUCCESS if successful

EFAILED or another BREW error if an error occurs (ENOMEMORY, etc.)

Comments:

None

See Also:

ICALLHISTORY_EnumNext()

Return to the List of functions

ICALLHISTORY_EnumNext()

Description:

This retrieves the next entry in the Call History list from current storage. ICALLHISTORY_EnumInit() must be called before any successive calls to this function. Call History entries are returned in reverse chronilogical order of addition to the system (i.e. Newest record first).

Prototype:

```
const AEECallHistoryEntry ICALLHISTORY_EnumNext
      (
      ICallHistory *pche,
      int *pnErr
      );
```

Parameters:

pich	[in]	Pointer to an ICallHistory interface object
pnErr	[out]	Pointer to an integer to hold any error value, set to SUCCESS if successful or at end of enumeration EFAILED or another BREW error if an error occurs

Return Value:

The "next" callhistory entry, if applicable

NULL if we're at the last entry or an error occurs

Comments:

The memory pointed to by the returned AEECallHistoryEntry is owned by the ICALLHISTORY object. Its contents will stay valid until the next call to ICALLHISTORY_EnumNext(), ICALLHISTORY_EnumInit(), or ICALLHISTORY_Release(). The contents of the returned pointer must not be modified by the caller.

See Also:

AEECallHistoryEntry

Return to the List of functions

ICALLHISTORY_UpdateEntry()

Description:

This replaces the current call history entry with the one provided. The current entry is defined as the entry that was returned during the most recent call to ICALLHISTORY_EnumNext().

Prototype:

```
int ICALLHISTORY_UpdateEntry
      (
      ICallHistory *pich,
      const AEECallHistoryEntry *pche
      );
```

Parameters:

pich	Pointer to an ICallHistory interface object
pche	New data to replace existing entry

Return Value:

SUCCESS if successful

EFAILED or some other BREW error if failed to update entries

EBADPARM if field data is not valid for field type or invalid AEECallHistory entry structure.

Comments:

None

See Also:

AEECallHistoryEntry

Return to the List of functions

ICamera Interface

ICamera interface provides a generic way to BREW applications to control device camera and to record snapshots and movies in various formats like JPEG, MPEG4, and others.

Event Notification:

ICamera asynchronously notifies all the events to client app via the callback function. App must register a callback notification function using ICAMERA_RegisterNotify().

Display:

ICamera dispatches the captured frames via user registered callback function in preview, snapshot and movie modes. The frame is deliverd via CAM_STATUS_FRAME callback. In the callback, user needs to call ICAMERA_GetFrame() to get the frame represented by IBitmap.

It is app's responsibility to display these frames on to the screen or other destination. ICamera DOES NOT perform any display operations.

Preview Mode:

Before you start camera in preview mode, you need to perform the following operations:

(1) ICAMERA_SetDisplaySize() to set the frame display size

(2) [Optional] ICAMERA_SetFramesPerSecond() to set the FPS of the camera

ICAMERA_Preview() starts the camera in preview mode. CAM_STATUS_START callback will be sent to app. Preview frames are delivered via CAM_STATUS_FRAME callback. Use ICAMERA_GetFrame() to retrieve the frame.

ICAMERA_Pause() stops the frame callbacks. ICAMERA_Resume() resumes the frame callbacks.

ICAMERA_Stop() stops the preview operation and puts the camera in ready mode. CAM_STATUS_DONE callback will be sent to app.

Snapshot Mode:

Before you do snapshot operation, you need to perform the following operations:

(1) ICAMERA_SetMediaData()

(2) ICAMERA_SetSize()

(3) [Optional] ICAMERA_SetVideoEncode()

(4) [Optional] ICAMERA_SetQuality()

(5) [Optional] ICAMERA_SetFramesPerSecond()

ICAMERA_RecordSnapshot() starts the snapshot recording operation. CAM_STATUS_START callback will be sent to app followed by CAM_STATUS_DONE when recording is complete. When the encoding is completed, {CAM_CMD_ENCODESNAPSHOT, CAM_STATUS_DONE} callback will be sent.

ICamera can be configured to defer the snapshot encoding as follows. This is known as DeferEncode feature. Note that, by default, DeferEncode is disabled.

(1) ICAMERA_SetMediaData()

(2)ICAMERA_DeferEncode(). Use ICAMERA_DeferEncode(TRUE) to Indicate that encoding must be defered

ICAMERA_EncodeSnapshot() starts the snapshot recording operation. CAM_STATUS_START callback will be sent to app followed by CAM_STATUS_DONE when recording is complete. Now, only raw frame is recorded and it is not yet encoded. App can access the raw frame using ICAMERA_GetFrame() in the callback. ICAMERA_EncodeSnapshot() encodes the frame and sends {CAM_CMD_ENCODESNAPSHOT, CAM_STATUS_DONE} callback when encoding is done. Do ICAMERA_SetMediaData() before calling ICAMERA_EncodeSnapshot().

Movie Mode:

Before you do start recording movie, you need to perform the following operations:

(1) ICAMERA_SetMediaData()

(2) ICAMERA_SetSize()

(3) [Optional] ICAMERA_SetVideoEncode() and ICAMERA_SetAudioEncode()

(4) [Optional] ICAMERA_SetQuality()

(5) [Optional] ICAMERA_SetFramesPerSecond()

Recorded frames are delivered via CAM_STATUS_FRAME callback. Use:

ICAMERA_GetFrame() to retrieve the frame.

ICAMERA_Pause() pauses recording as well as stops the frame callbacks.

ICAMERA_Resume() resumes the recording and the frame callbacks.

ICAMERA_Stop() stops the record operation and puts the camera in ready mode. CAM_STATUS_DONE callback will be sent to app.

App Suspend/Resume:

When app gets EVT_APP_SUSPEND, it is recommended that app stop the camera and release ICamera interface.

When app gets EVT_APP_RESUME, it can create ICamera interface and resume its operation.

Sample Code:

The following code snippet starts the camera in preview mode and displays frames to the device screen.

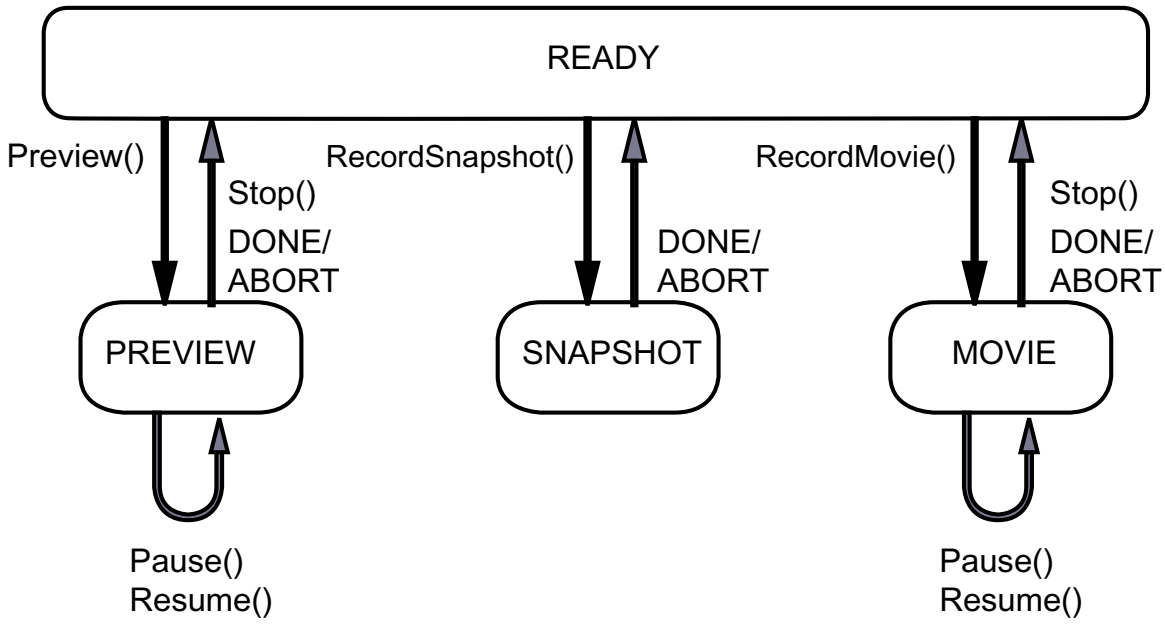
```
static int CApp_StartCameraInPreviewMode(CApp * pme)
     {
         int   nErr;
         // Create ICamera instance.
       nErr = ISHELL_CreateInstance(pme->a.m_pIShell, AEECLSID_CAMERA,
(void **)&pme->m_pICamera);
         if (nErr)
            return nErr;
         // Register callback notification function.
         nRet = ICAMERA_RegisterNotify(pme->m_pICamera,
CApp_CameraNotify, pme);
         if (nErr)
            return nErr;
        ICAMERA_SetDisplaySize(pme->m_pICamera, &pme->m_sizePreview);
         nErr = ICAMERA_Preview(pme->m_pICamera);
         if (nErr)
            return nErr;
         return SUCCESS;
      }
static void CApp_CameraNotify(void * pUser, AEECameraNotify * pn)
     {
         CApp *   pme = (CApp *)pUser;
if (!pme || !pn)
            return;
switch (pn->nStatus)
       {
            case CAM_STATUS_START:
      // Preview has begun...
      break;
case CAM_STATUS_FRAME:
          {
      IBitmap *      pFrame;
      AEEBitmapInfo  bi;

      //
      // IMPORTANT NOTE: You need to do IBITMAP_Release(pFrame) after
      you're done with pFrame.
      //
      if (SUCCESS == ICAMERA_GetFrame(pme->m_pICamera, &pFrame))
      return;
// Get the bitmap info...this can be saved in app global structure.
      IBITMAP_GetInfo(pFrame, &bi, sizeof(bi));

      // Display the frame at (0, 0) location of the screen
      IDISPLAY_BitBlt(pme, 0, 0, bi.cx, bi.cy, pFrame, 0, 0,
      AEE_RO_COPY);
      IBITMAP_Release(pFrame);
      break;
           }
case CAM_STATUS_DONE:
```

```
            // ICAMERA_Stop() was called and preview operation stopped.
            break;
      case CAM_STATUS_ABORT:
            // Preview got aborted.
            break;
               }
            }
```

Camera State Machine:



ICamera Call Flow for Preview Mode

BREW App | **[ICamera State]** | **BREW ICamera** | **Example OEM Camera Layer**

ICAMERA_SetDisplaySize() >

[READY] ICAMERA_Preview() >

camera_start_preview() >

[PREVIEW] < SUCCESS

< CAM_CMD_START/CAM_STATUS_START

< FRAME

< CAM_CMD_START/CAM_STATUS_FRAME

ICAMERA_GetFrame() >

[returns PreviewFrame IBitmap] >

< FRAME

< CAM_CMD_START/CAM_STATUS_FRAME

ICAMERA_GetFrame() >

[returns PreviewFrame IBitmap] >

…

ICAMERA_Stop() >

camera_stop_preview() >

[READY] < DONE

ICamera Call Flow for Picture Taking Mode

Record SnapShot (Immediate Encoding)

BREW App [ICamera State]	BREW ICamera	Example OEM Camera Layer
	ICAMERA_SetMediaData() >	
	ICAMERA_SetVideoEncode() >	
	ICAMERA_SetSize() >	
	[Optional]ICAMERA_SetQuality() >	
[READY]	ICAMERA_RecordSnapshot() >	
		camera_take_picture() >
		< SUCCESS
[SNAPSHOT]	< CAM_STATUS_START	
		< DONE(SnapshotFrame)
[READY]	< CAM_CMD_START/ CAM_STATUS_DONE	
		camera_encode(SnapshotFrame) >
		< SUCCESS
		< DONE
[READY]	< CAM_CMD_ENCODESNAPSHOT/ CAM_STATUS_DONE	

ICamera Call Flow for Recording SnapShot (deferred encoding) Mode

BREW App

[ICamera State]	BREW ICamera	Example OEM Camera Layer
	ICAMERA_DeferEncode() >	
	ICAMERA_SetMediaData() >	
	ICAMERA_SetVideoEncode() >	
	ICAMERA_SetSize() >	
	ICAMERA_SetQuality() >	
[READY]	ICAMERA_RecordSnapshot() >	
		camera_take_picture() >
		< SUCCESS
[SNAPSHOT]	< CAM_STATUS_START	
		< DONE(SnapshotFrame)
[READY]	< CAM_CMD_START/CAM_STATUS_DONE	
	ICAMERA_GetFrame() >	
		[returns SnapshotFrame IBitmap] >
	ICAMERA_EncodeSnapshot(SnapshotFrame) >	
		camera_encode(SnapshotFrame) >
		< SUCCESS
		< DONE
[READY]	< CAM_CMD_ENCCODESNAPSHOT/ CAM_STATUS_DONE	

ICamera Call Flow for Recording Movie Mode

BREW App

[ICamera State]	BREW ICamera	Example OEM Camera Layer
	ICAMERA_SetMediaData() >	
	ICAMERA_SetVideoEncode() >	
	ICAMERA_SetAudioEncode() >	
	ICAMERA_SetSize() >	
	[Optional]ICAMERA_SetQuality() >	
	[Optional]ICAMERA_SetFramesPerSecond() >	
[READY]	ICAMERA_RecordMovie() >	

```
                                                                camera_start_record() >
                                                        < SUCCESS
[MOVIE]     < CAM_CMD_START/CAM_STATUS_START
                                                        < FRAME
            < CAM_CMD_START/CAM_STATUS_FRAME
                                        ICAMERA_GetFrame() >
                                                    [returns MovieFrame IBitmap] >
                                                        < FRAME
            < CAM_CMD_START/CAM_STATUS_FRAME
                                        ICAMERA_GetFrame() >
                                                    [returns MovieFrame IBitmap] >
                                    …
                                            ICAMERA_Pause() >
                                                                camera_pause_record() >
                                                        < SUCCESS
            < CAM_CMD_START/CAM_STATUS_PAUSE
                                        ICAMERA_Resume() >
                                                                camera_resume_record() >
                                                        < SUCCESS
            < CAM_CMD_START/CAM_STATUS_RESUME
                                    …
                                            ICAMERA_Stop() >
                                                                camera_stop_record() >
                                                        < DONE
[READY]     < CAM_CMD_START/CAM_STATUS_DONE
```

List of Header files to be included

The following header file is required:

AEECamera.h

List of functions

Functions in this interface include:

ICAMERA_AddOverlay()
ICAMERA_AddRef()
ICAMERA_ClearOverlay()
ICAMERA_DeferEncode()
ICAMERA_EncodeSnapshot()
ICAMERA_GetDisplaySizeList()
ICAMERA_GetFrame()
ICAMERA_GetMode()
ICAMERA_GetParm()
ICAMERA_GetSizeList()
ICAMERA_IsBrightness()
ICAMERA_IsContrast()
ICAMERA_IsMovie()
ICAMERA_IsSharpness()
ICAMERA_IsSupport()
ICAMERA_IsZoom()
ICAMERA_Pause()
ICAMERA_Preview()
ICAMERA_QueryInterface()
ICAMERA_RecordMovie()
ICAMERA_RecordSnapshot()
ICAMERA_RegisterNotify()
ICAMERA_Release()
ICAMERA_Resume()
ICAMERA_RotateEncode()
ICAMERA_RotatePreview()
ICAMERA_SetAudioEncode()
ICAMERA_SetBrightness()
ICAMERA_SetContrast()
ICAMERA_SetDisplaySize()
ICAMERA_SetFramesPerSecond()
ICAMERA_SetMediaData()
ICAMERA_SetParm()
ICAMERA_SetQuality()
ICAMERA_SetSharpness()
ICAMERA_SetSize()
ICAMERA_SetVideoEncode()
ICAMERA_SetZoom()
ICAMERA_Start()
ICAMERA_Stop()

The remainder of this section provides details for each function.

ICAMERA_AddOverlay()

Description:

This function sets the overlay image that will be part of the recorded picture. This operation is done any camera mode.

Prototype:

```
int ICAMERA_AddOverlay
      (
      ICamera * pICamera,
      IBitmap * pb
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
pb	Pointer to IBitmap representing the overlay

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

You can add overlays on top of another image by calling this function repeatedly with different images. To clear ALL overlays, call ICAMERA_ClearOverlay().

See Also:

ICAMERA_SetParm()
ICAMERA_ClearOverlay()
Return to the List of functions

ICAMERA_AddRef()

Description:

This function increments the reference count of the ICamera 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 ICAMERA_AddRef(ICamera * pICamera)
```

Parameters:

pICamera: Pointer to the ICamera Interface object

Return Value:

Incremented reference count for the object

Comments:

A valid object returns a positive reference count.

See Also:

ICAMERA_Release()

Return to the List of functions

ICAMERA_ClearOverlay()

Description:

This function clears all the overlaid images.

Prototype:

```
int ICAMERA_ClearOverlay(ICamera * pICamera);
```

Parameters:

pICamera Pointer to ICamera Interface.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_GetParm()

ICAMERA_ClearOverlay()

Return to the List of functions

ICAMERA_DeferEncode()

Description:

This function defers the encoding of the snapshot done by ICAMERA_RecordSnapshot() API. You need to explicitly call ICAMERA_EncodeSnapshot() to encode the snapshot.

Prototype:

```
int ICAMERA_DeferEncode
      (
      ICamera * pICamera,
      boolean bDefer
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
bDefer	TRUE implies the encoding will be done by user.

Return Value:

SUCCESS if successful.

Error code if failure.

Comments:

None

See Also:

ICAMERA_Start()

ICAMERA_RecordSnapshot()

ICAMERA_EncodeSnapshot()

Return to the List of functions

ICAMERA_EncodeSnapshot()

Description:

This function, typically, encodes the recorded snapshot.

Prototype:

```
int ICAMERA_EncodeSnapshot
      (
      ICamera * pICamera,
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.

Return Value:

SUCCESS if successful.

Error code if failure.

Comments:

This API is typically called in response to CAM_STATUS_DONE when ICAMERA_RecordSnapshot(). You can use ICAMERA_GetFrame()() to get the latest raw snapshot frame.

This function results in {CAM_CMD_ENCODESNAPSHOT, CAM_STATUS_DONE} callback after the snapshot is encoded.

This API can be called in any mode. It may abort the current active operation like preview, movie, encode snapshot, etc. It may also be used to encode any frame, if supported.

See Also:

ICAMERA_Start()

ICAMERA_RecordSnapshot()

ICAMERA_GetFrame()

Return to the List of functions

ICAMERA_GetDisplaySizeList()

Description:

This function retrieves the list of discrete display sizes supported for specified mode or continuous range (e.g. any size between 10x10 to 100x150).

Prototype:

```
int ICAMERA_GetDisplaySizeList
      (
      ICamera * pICamera,
      AEESize ** ppList,
      boolean * pbRange
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
ppList	[in]	*ppList contains CAM_MODE_PREVIEW/CAM_MODE_MOVIE
	[out]	Pointer to NULL- terminated size list
pbRange	[out]	Pointer to boolean when TRUE indicates the passed list is a NULL-terminated paired list (i.e. multiple of 2) of ranges

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

The list should be copied and should not be freed.

See Also:

ICAMERA_GetParm()

AEESize

Return to the List of functions

ICAMERA_GetFrame()

Description:

This function returns the current frame captured by the camera.

Prototype:

```
IBitmap * ICAMERA_GetFrame(ICamera * pICamera, IBitmap ** ppFrame);
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
ppFrame	[out]	Frame IBitmap returned .

Return Value:

SUCCESS, if successful

Error code, if failure.

Comments:

This function is typically called in response to CAM_STATUS_FRAME callback. It is caller's responsibility to release the IBitmap object after calling this function.

The caller can QueryInterface, on the returned IBitmap, for IDIB which, if supported, allows access to frame data.

See Also:

ICAMERA_Start()

ICAMERA_Preview()

ICAMERA_RecordMovie()

Return to the List of functions

ICAMERA_GetMode()

Description:

This function returns the current camera mode.

Prototype:

```
int ICAMERA_GetMode
      (
      ICamera * pICamera,
      int16 * pnMode,
      boolean * pbPaused
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pnMode	[out]	Pointer to mode. CAM_MODE_XXX
pbPaused	[out]	TRUE/FALSE Paused/Resumed

Return Value:

SUCCESS: Successful. Operation completed.

EBADPARM: Bad parm

Comments:

None

See Also:

ICAMERA_Start()

Return to the List of functions

ICAMERA_GetParm()

Description:

This function gets the camera control parameters.

Prototype:

```
int ICAMERA_GetParm
      (
      ICamera * pICamera,
      int16 nParmID,
      int32 * pP1,
      int32 * pP2
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nParmID	CAM_PARM_XXX
pP1	Depends on the nParmID parameter
pP2	Depends on the nParmID parameter

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

See Camera Control Parameters for parameter details.

See Also:

ICAMERA_SetParm()

Camera Control Parameters

Return to the List of functions

ICAMERA_GetSizeList()

Description:

This function retrieves the list of discrete sizes supported for specified mode or continuos range (e.g. any size between 10x10 to 100x150).

Prototype:

```
int ICAMERA_GetSizeList
      (
      ICamera * pICamera,
      AEESize ** ppList,
      boolean * pbRange
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
ppList	[in]	ppList contains CAM_MODE_SNAPSHOT/CAM_MODE_MOVIE
	[out]	Pointer to NULL- terminated size list
pbRange	[out]	Pointer to boolean when TRUE indicates the passed list is a NULL-terminated paired list (i.e. multiple of 2) of ranges

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

The list should be copied and should not be freed.

See Also:

ICAMERA_GetParm()
AEESize

Return to the List of functions

ICAMERA_IsBrightness()

Description:

This function checks if camera has brightness setting capability.

Prototype:

```
int ICAMERA_IsBrightness
      (
      ICamera * pICamera,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_IsSupport()

Return to the List of functions

ICAMERA_IsContrast()

Description:

This function checks if camera has contrast setting capability.

Prototype:

```
int ICAMERA_IsContrast
      (
      ICamera * pICamera,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_IsSupport()

Return to the List of functions

ICAMERA_IsMovie()

Description:

This function checks if camera has movie recording capability.

Prototype:

```
int ICAMERA_IsMovie
      (
      ICamera * pICamera,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_IsSharpness()

Description:

This function checks if camera has sharpness setting capability.

Prototype:

```
int ICAMERA_IsSharpness
      (
      ICamera * pICamera,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_IsSupport()

Return to the List of functions

ICAMERA_IsSupport()

Description:

This function checks if specified parameter is supported by ICamera Interface. This function is useful to check the camera capabilities like setting of brightness, zoom, etc.

Prototype:

```
int ICAMERA_IsSupport
      (
      ICamera * pICamera,
      int16 nParmID,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
nParmID	[in]	CAM_PARM_XXX parameter ID.See Camera Control Parameters
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_IsZoom()

Description:

This function checks if camera has zoom capability.

Prototype:

```
int ICAMERA_IsZoom
      (
      ICamera * pICamera,
      boolean * pbSupport
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
pbSupport	[out]	Pointer to boolean. TRUE/FALSE => Supported/Unsupported.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Value returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot get parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_IsSupport()

Return to the List of functions

ICAMERA_Pause()

Description:

This function pauses the camera operation. In preview and record modes, the frame callbacks are paused. In record mode, the encoding is also paused.

Prototype:

```
int ICAMERA_Pause(ICamera * pICamera);
```

Parameters:

pICamera	Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

ENOMEMORY: Not enough memory

EBADSTATE: Pause cannot be done in current state

Comments:

This API does not apply to Snapshot mode.

This function results in CAM_STATUS_PAUSE status callback.

In the callback, AEECameraNotify,

`nCmd = CAM_CMD_START and nSubCmd = CAM_MODE_PREVIEW/CAM_MODE_MOVIE.`

See Also:

ICAMERA_Start()
ICAMERA_Preview()
ICAMERA_RecordMovie()
ICAMERA_Resume()

Return to the List of functions

ICAMERA_Preview()

Description:

This function starts the camera operation in preview mode, which causes ICamera to start sending frames to the client.

Prototype:

```
int ICAMERA_Preview(ICamera * pICamera);
```

Parameters:

pICamera Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

EBADPARM: Bad parm is passed

ENOMEMORY: Not enough memory

EBADSTATE: Preview cannot be done in current state

Comments:

You need to set the frame display size before calling this function. All the events that originate due to this API and due to the following APIs will be reported via the user-specified callback:

ICAMERA_Stop()

ICAMERA_Pause()

ICAMERA_Resume()

CAM_STATUS_START callback happens once the preview begins.

CAM_STATUS_FRAME callbacks happen continuously unless you pause.

CAM_STATUS_DONE callback occurs when preview is stopped.

CAM_STATUS_ABORT callback occurs when preview is aborted.

In the callback, AEECameraNotify,

`nCmd = CAM_CMD_START and nSubCmd = CAM_MODE_PREVIEW.`

See Also:

ICAMERA_Start()
ICAMERA_Stop()
ICAMERA_Pause()
ICAMERA_Resume()
ICAMERA_GetFrame()

Return to the List of functions

ICAMERA_QueryInterface()

Description:

This function can be used to

- Get a pointer to an interface or data based on the input class ID
- Query an extended version of the ICamera-derived class
- Support version compatibility

Prototype:

```
int ICAMERA_QueryInterface
      (
      ICamera * pICamera,
      AEECLSID clsReq,
      void ** ppo
      )
```

Parameters:

pICamera	[in]	Pointer to ICamera Interface.
clsReq	[in]	A globally unique id to identify the entity (interface or data) that we are trying to query.
ppo	[out]	Pointer to the interface or data that we want to retrieve. If the value passed back is NULL, the interface or data that we query are not available.

Return Value:

Return SUCCESS on success, otherwise returns error code.

Comments:

If ppo is back a NULL pointer, the interface or data that we query is not available.

Side Effects:

If an interface is retrieved, then this function increments its reference count.

If a data structure is retrieved, then a pointer to the internal structure is given and user should not free it.

See Also:

None

Return to the List of functions

ICAMERA_RecordMovie()

Description:

This function starts the camera operation in movie mode, which causes recorded frames to be sent to the caller while encoding those frames.

Prototype:

```
int ICAMERA_RecordMovie(ICamera * pICamera);
```

Parameters:

pICamera Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

EBADPARM: Bad parm is passed

ENOMEMORY: Not enough memory

EBADSTATE: RecordMovie() cannot be done in current state

Comments:

You need to set the media data before calling this function. Also, you may want to specify active encoding, picture format and quality that should be used for encoding the movie.

All the events that originate due to this API and due to the following APIs will be reported via the user-specified callback:

ICAMERA_Stop()

ICAMERA_Pause()

ICAMERA_Resume()

CAM_STATUS_START callback happens once the recording begins.

CAM_STATUS_FRAME callbacks happen continuously unless you pause.

CAM_STATUS_DONE callback occurs when recording is stopped.

CAM_STATUS_ABORT callback occurs when recording is aborted. In the callback, AEECameraNotify,

`nCmd = CAM_CMD_START and nSubCmd = CAM_MODE_MOVIE.`

See Also:

ICAMERA_Start()
ICAMERA_Stop()
ICAMERA_Pause()
ICAMERA_Resume()
ICAMERA_GetFrame()

Return to the List of functions

ICAMERA_RecordSnapshot()

Description:

This function starts the camera operation in snapshot mode which causes the camera to take a snapshot. If Defer Encoding is not enabled (default), this function causes the snapshot to be encoded.

Prototype:

```
int ICAMERA_RecordSnapshot(ICamera *pICamera);
```

Parameters:

pICamera	Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

EBADPARM: Bad parm is passed

ENOMEMORY: Not enough memory

EBADSTATE: RecordSnapshot() cannot be done in current state

Comments:

You need to set the media data before calling this function. Also, you may want to specify active encoding, picture format and quality that should be used for encoding the snapshot. In the callback, AEECameraNotify, if nCmd = CAM_CMD_START, then nSubCmd = CAM_MODE_SNAPSHOT.

This function results in {CAM_CMD_START, CAM_STATUS_START} callback followed by {CAM_CMD_START, CAM_STATUS_DONE} callback after the snapshot is taken. This is followed by {CAM_CMD_ENCODESNAPSHOT, CAM_STATUS_DONE} callback after the snapshot is encoded.

You can defer encoding by calling ICAMERA_DeferEncoding(pICamera, TRUE); This causes ICAMERA_RecordSnaphot() not to encode the snapshot. You can get the raw snapshot frame using ICAMERA_GetFrame() and call ICAMERA_EncodeSnapshot() to encode the snapshot.

See Also:

ICAMERA_Start()
ICAMERA_EncodeSnapshot()
ICAMERA_DeferEncode()
ICAMERA_GetFrame()

Return to the List of functions

ICAMERA_RegisterNotify()

Description:

This function registers a callback notification function with ICamera object. ICamera reports asynchronous events using this callback.

Prototype:

```
int ICAMERA_RegisterNotify
      (
      IMedia * pICamera,
      PFNCAMERANOTIFY pfnNotify,
      void * pUser
      )
```

Parameters:

pICamera	Pointer to the IMedia Interface object
pfnNotify	User callback function pointer
pUser	User data to be used when calling pfnNotify()

Return Value:

SUCCESS: Successful.

EBADSTATE: Error - IMedia is not in Ready state.

Comments:

None

See Also:

None

Return to the List of functions

ICAMERA_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 ICAMERA_Release(ICamera * pICamera)
```

Parameters:

pICamera: Pointer to the ICamera 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

See Also:

ICAMERA_AddRef()

Return to the List of functions

ICAMERA_Resume()

Description:

This function resumes the camera operation. In preview and record modes, the frame callbacks are resumed. In record mode, the encoding is also resumed.

Prototype:

```
int ICAMERA_Resume(ICamera * pICamera);
```

Parameters:

pICamera	Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

ENOMEMORY: Not enough memory

EBADSTATE: Resume cannot be done in current state

Comments:

This API does not apply to Snapshot mode. This function results in CAM_STATUS_RESUME status callback. In the callback, AEECameraNotify,

`nCmd = CAM_CMD_START and nSubCmd = CAM_MODE_PREVIEW/CAM_MODE_MOVIE.`

See Also:

ICAMERA_Start()
ICAMERA_Pause()
ICAMERA_Preview()
ICAMERA_RecordMovie()
ICAMERA_RecordSnapshot()

Return to the List of functions

ICAMERA_RotateEncode()

Description:

This function rotates the recorded and encoded frame. Only snapshot and movie modes are affected.

Prototype:

```
int ICAMERA_RotateEncode(ICamera * pICamera, int32 nValue);
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Rotation angle

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_ROTATE_ENCODE details.

See Also:

ICAMERA_GetParm()

ICAMERA_RotatePreview()

Return to the List of functions

ICAMERA_RotatePreview()

Description:

This function rotates the preview frame. Only preview mode is affected.

Prototype:

```
int ICAMERA_RotatePreview(ICamera * pICamera, int32 nValue);
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Rotation angle

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_ROTATE_PREVIEW details.

See Also:

ICAMERA_GetParm()

ICAMERA_RotateEncode()

Return to the List of functions

ICAMERA_SetAudioEncode()

Description:

This function sets the active audio encoding type used to encode along with the recorded snapshot/movie.

Prototype:

```
int ICAMERA_SetAudioEncode
      (
      ICamera * pICamera,
      AEECLSID cls,
      uint32 dwExtra
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
cls	Encoding class ID. E.g. AEECLSID_MEDIAQCP, etc.
dwExtra	Extra info regarding the encoding like sub formats. E.g. For AEECLSID_MEDIAQCP, sub-format can be specified as MM_QCP_FORMAT_FIXED_FULL_EVRC.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_AUDIO_ENCODE details.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetBrightness()

Description:

This function sets the brightness of the camera.

Prototype:

```
int ICAMERA_SetBrightness
      (
      ICamera * pICamera,
      int32 nValue
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Brightness value.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_BRIGHTNESS details.

See Also:

ICAMERA_GetParm()
AEEParmInfo

Return to the List of functions

ICAMERA_SetContrast()

Description:

This function sets the contrast of the camera.

Prototype:

```
int ICAMERA_SetContrast
      (
      ICamera * pICamera,
      int32 nValue
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Contrast value.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_CONTRAST details.

See Also:

ICAMERA_GetParm()
AEEParmInfo

Return to the List of functions

ICAMERA_SetDisplaySize()

Description:

This function sets the frame display size where the captured data is displayed.

Prototype:

```
int ICAMERA_SetDisplaySize
      (
      ICamera * pICamera,
      AEESize * pSize
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
pSize	Frame display size within the main display/off-srceen buffer area

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetFramesPerSecond()

Description:

This function sets the frames per second setting of camera in preview or movie mode.

Prototype:

```
int ICAMERA_SetFramesPerSecond
      (
      ICamera * pICamera,
      uint32 dwFPS
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
dwFPS	Frames per second. See dwFPS format in CAM_PARM_FPS documentation.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_FPS details.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetMediaData()

Description:

This function sets the media data where the recorded and encoded data will be saved.

Prototype:

```
int ICAMERA_SetMediaData
      (
      ICamera * pICamera,
      AEEMediaData * pmd,
      const char * cpszMIME
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
pmd	Pointer to media data
cpszMIME	MIME type of the media

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

None

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetParm()

Description:

This function sets the camera control parameters.

Prototype:

```
int ICAMERA_SetParm
      (
      ICamera * pICamera,
      int16 nParmID,
      int32 p1,
      int32 p2
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nParmID	CAM_PARM_XXX. See Camera Control Parameters
p1	Depends on the nParmID parameter
p2	Depends on the nParmID parameter

Return Value:

SUCCESS: Successful. Operation is completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

See Camera Control Parameters for parameter details.

]See Also:

ICAMERA_GetParm()

Camera Control Parameters

Return to the List of functions

ICAMERA_SetQuality()

Description:

This function sets the camera to capture specified picture quality.

Prototype:

```
int ICAMERA_SetQuality
      (
      ICamera * pICamera,
      int16 nQuality
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nQuality	Picture quality

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_QUALITY details.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetSharpness()

Description:

This function sets the sharpness of the camera.

Prototype:

```
int ICAMERA_SetSharpness
      (
      ICamera * pICamera,
      int32 nValue
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Sharpness value.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_SHARPNESS details.

ICAMERA_GetParm() with CAM_PARM_SHARPNESS, returns AEEParmInfo that specifies the sharpness info.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetSize()

Description:

This function sets the camera to record a snapshot or movie in specified size.

Prototype:

```
int ICAMERA_SetSize
      (
      ICamera * pICamera,
      AEESize * pSize
      )
```

Parameters:

pICamera	Pointer to ICamera Interface
pSize	Size of the picture

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_SIZE_LIST details.

See Also:

ICAMERA_GetParm()
CAM_PARM_SIZE_LIST

Return to the List of functions

ICAMERA_SetVideoEncode()

Description:

This function sets the active video/image encoding type used to encode the recorded snapshot/movie.

Prototype:

```
int ICAMERA_SetVideoEncode
      (
      ICamera * pICamera,
      AEECLSID cls,
      uint32 dwExtra
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
cls	Encoding class ID CAM_ENCODE_RAW, AEECLSID_JPEG AEECLSID_MEDIAMPEG4, etc.
dwExtra	Extra info regarding the encoding like sub formats.

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_VIDEO_ENCODE details.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_SetZoom()

Description:

This function sets the zoom of the camera.

Prototype:

```
int ICAMERA_SetZoom
      (
      ICamera * pICamera,
      int32 nValue
      )
```

Parameters:

pICamera	Pointer to ICamera Interface.
nValue	Zoom value

Return Value:

SUCCESS: Successful. Operation completed.

CAM_PENDING: Result returned via the registered callback

EBADPARM: Bad parm

ENOMEMORY: Not enough memory

EBADSTATE: Cannot set parm in the current state

EUNSUPPORTED: Parm not supported by the class

Comments:

For ICAMERA_GetParm() information, see Camera Control Parameters for CAM_PARM_ZOOM details.

ICAMERA_GetParm() with CAM_PARM_ZOOM, returns AEEParmInfo that specifies the zoom info.

See Also:

ICAMERA_GetParm()

Return to the List of functions

ICAMERA_Start()

Description:

This function starts camera operation in preview, snapshot, or movie mode.

Prototype:

```
int ICAMERA_Start
      (
      ICamera * pICamera,
      int16 nMode,
      uint32 dwParam
      );
```

Parameters:

pICamera	Pointer to ICamera Interface
nMode	CAM_MODE_PREVIEW CAM_MODE_SNAPSHOT CAM_MODE_MOVIE
dwParam	Reserved

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

EBADPARM: Bad parm is passed

ENOMEMORY: Not enough memory

EBADSTATE: Start cannot be done in current state

Comments:

All the events that originate due to this API and due to the following API will be reported via the user-specified callback.

ICAMERA_Preview()

ICAMERA_RecordSnapshot()

ICAMERA_RecordMovie()

ICAMERA_Stop()

ICAMERA_Pause()

ICAMERA_Resume()

In the callback, AEECameraNotify,

`nCmd = CAM_CMD_START and nSubCmd = nMode.`

See Also:

AEECameraNotify
ICAMERA_Start()
ICAMERA_Stop()

ICAMERA_Pause()
ICAMERA_Resume()
ICAMERA_GetFrame()
ICAMERA_Preview()
ICAMERA_RecordMovie()
ICAMERA_RecordSnapshot()

Return to the List of functions

ICAMERA_Stop()

Description:

This function stops the current camera operation and puts it in Ready state.

Prototype:

```
int ICAMERA_Stop(ICamera * pICamera);
```

Parameters:

pICamera	Pointer to ICamera Interface.

Return Value:

SUCCESS: Command accepted

EFAILED: General failure

ENOMEMORY: Not enough memory

EBADSTATE: Stop cannot be done in current state

Comments:

This function results in CAM_STATUS_DONE status callback.

In the callback, AEECameraNotify, nCmd = CAM_CMD_START and nSubCmd = CAM_MODE_PREVIEW/CAM_MODE_MOVIE.

See Also:

ICAMERA_Start()
ICAMERA_Preview()
ICAMERA_RecordMovie()
ICAMERA_RecordSnapshot()

Return to the List of functions

IDIB Interface

IDIB is a structure and an interface. IDIB inherits all of the member functions of IBitmap, so an IDIB may be used as an IBitmap by type casting. The IDIB_TO_IBITMAP in-line function is supplied for type safe casting. Unlike other BREW interfaces, IDIB also has public data members. These data members can be used to efficiently read or modify image data.

An application typically obtains an IDIB pointer from an IBitmap pointer by calling IBITMAP_QueryInterface() with the class ID AEECLSID_DIB. Not all IBitmap classes support IDIB, and in those cases the QueryInterface function will return an error code. On success, an IDIB pointer is returned, which must be released when the caller has finished using it.

A bitmap consists of a 2-dimensional array of pixels. IDIB contains members that indicate where in memory the pixels are, and how the pixel values are to be interpreted.

Pixel array structure

The locations and sizes of pixels in the pixel array are described by the **pBmp**, **nPitch**, and **nDepth** members. The **nPitch** field specifies the distance (in bytes) from the beginning of any row to the beginning of the next row. Pitch is typically function of the bitmap width, the padding being applied, and whether it is a top-down or bottom-up bitmap. A bottom-up bitmaps will have a negative pitch value.

Users of an IDIB should honor the **nPitch** value and make no assumptions about padding or direction in the bitmap.The **pBmp** parameter points to the top scan line, y=0, in the memory buffer that holds the pixel data of a DIB. For a top-down DIB, the pointer points to the first row of the buffers pixel data. For a bottom-up DIB, the pointer points to the last row of the buffers pixel data.

Usage example:

For a bitmap of color depth 8 (one byte per pixel), width of 9, and height of 10, the following representations are possible (among others):

nPitch	pBmp	Start of bit array
Top-down BMP file	12	0
Bottom-up BMP file	-12	96
Top-down packed bitmap	9	0
Bottom-up packed bitmap	- 9	72

In all of these cases, and in fact for any 8-bit DIB, the code for reading a pixel remains the same:

```
COLORVALUE = pdib->pBmp[ y * pdib->nPitch + x ]
```

Alignment: Rows typically starts at 32-bit boundaries, but alignment is not guaranteed except in two cases. When **nDepth** is 32, rows should be 32-bit aligned, and when **nDepth** is 16, each row must be aligned on a 16-bit boundary.

Within a row, the left most (x=0) pixel begins at the most significant bit of the fits byte. Pixels are packed, bitwise, and split across bytes if necessary. 1, 2, 4, 8, and 16 bit bitmaps minimize splitting of pixel values across bytes, and yield the most efficiency. While possible and well-defined, sizes that tend to map irregularly to byte boundaries (like 3 or 12) will lead to reduced efficiency.

Pixel values

The type NativeColor is defined to represent values stored in pixels in the pixel array. Palette information (**cntRGB** and **pRGB**, or alternatively **nColorScheme**) describes how pixel values map to red, green, and blue intensity values.

When **cntRGB** is non-zero, the pixel values are treated as indices into the palette, which is an array of 32-bit R-G-B color values. (See **pRGB**.) When **nColorScheme** is non-zero, it identifies a mapping of pixel values to R-G-B values. This can be thought of as a hard-coded palette. (See **nColorScheme**.)

When both **cntRGB** and **nColorScheme** are zero, the color values of the bitmap are undefined.

Note that this mapping between pixel values and colors is somewhat independent of pixel size. A 4096 color bitmap requires 12 bits to represent all color values, but these may be stored in 12-bit (a packed representation) or 16-bit pixels (unpacked).

Transparency is a special case in interpreting pixel values. For operations that support transparency, such as IBITMAP_BltIn() when called with the AEE_RO_TRANSPARENT raster operation, pixels whose values match the transparent color (**ncTransparent**) are treated as transparent, which means that the corresponding pixels in the destination are left unmodified. **ncTransparent** is a NativeColor value, and the matching is performed on pixel values, not on the corresponding R-G-B intensities.

Palette Map

The **pPaletteMap** member is provided as a way for driver software to cache information computed from the bitmap's palette. This is not simply a function of the palette -- it depends also on the graphics algorithm, and any other bitmaps with which the DIB might interact. This member is set to NULL when there is no palette mapping object associated with the IDIB.

Every IDIB implementation must ensure that the **pPaletteMap** object (if set) is released when the IDIB itself is deleted. The IDIB_FlushPalette() macro performed this task.

Any **pPaletteMap** must be released whenever an IDIB is modified such that the interpretation of pixel values are affected (i.e. when **cntRGB**, **pRGB**[], **nColorScheme**, or **ncTransparent** are modified). Otherwise, a graphics algorithm might proceed to use stale data or perhaps even corrupt memory on a subsequent call.

Generally, users can ignore this value unless they modify a DIB palette's data after it has been used (i.e. graphics operations, such as IBitmap member functions, have been used on it). In that case, IDIB_FlushPalette() must be called.

Software Support

IDIB presents a wide range of possibilities for bitmap layouts, but practical constraints limit the number of formats that are supported by other software in the handset. Of the layouts that are supported, not all of those are fully optimized. These limitations can differ from handset to handset. The purpose of DIBs, however, is to communicate bitmap data to or from the graphics driver, so the target device's level of support for particular formats is important to application software. Here are some general guidelines:

- Each handset's primary display (whether monochrome, grey scale, or color) should support blitting from palette-based 1-, 2-, 4-, and 8-bit DIBs.

- Color mapping generally is done in an expedient fashion rather than the most accurate fashion. 1-bit and 8-bit operations should be particularly well optimized. Additionally, 16-bit handsets will support 16-bit DIB formats using either IDIB_COLORSCHEME_555 or IDIB_COLORSCHEME_565 color schemes, but blitting a 5-5-5 DIB to a 5-6-5 device, or vice versa, will incur a performance penalty.

8-bit palette-based DIBs should work well with all color displays, and are much smaller than 16-bit images, so are recommended for color images. Since each 8-bit image can have its own selection of 256 colors, color accuracy should not be a problem.

List of Header files to be included

The following header file is required:

AEEBitmap.h

List of functions

Functions in this interface include:

IDIB_AddRef()
IDIB_FlushPalette()
IDIB_QueryInterface()
IDIB_Release()
IDIB_TO_IBITMAP()

The remainder of this section provides details for each function.

IDIB_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IDIB_Release()

Return to the List of functions

IDIB_FlushPalette()

Description:

This macro is used to release the **pPaletteMap** member of the IDIB structure. This is necessary whenever the DIB's palette is modified, since the information cached in the palette map will no longer be valid. This macro first check whether **pPaletteMap** is NULL. If so, it does nothing. Otherwise, it calls the palette map's release function, and sets **pPaletteMap** to NULL.

Prototype:

```
IDIB_FlushPalette(pdib) if ((pdib)->pPaletteMap)
      {IQI_Release((pdib)->pPaletteMap);
      (pdib)->pPaletteMap = 0;}
```

Parameters:

pdib	[in]	Pointer to IDIB structure.

Return Value:

no return value

Comments:

None

See Also:

None

Return to the List of functions

IDIB_QueryInterface()

Description:

This function is inherited from IQI_QueryInterface().

See Also:

IDIB

Return to the List of functions

IDIB_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IDIB_AddRef()

Return to the List of functions

IDIB_TO_IBITMAP()

Description:

This function provides type safe casting from IDIB Interface pointers to IBitmap Interface pointers. This function should be used when passing an IDIB interface to an IBitmap function. This is safer than a simple cast, since the compiler will verify the pointer type.

Prototype:

```
_inline IBitmap *IDIB_TO_IBITMAP(const IDIB *pIDIB);
```

Parameters:

pIDIB	Pointer to an IDIB interface.

Return Value:

Returns **pIDIB** cast to an IBitmap*.

Comments:

None

See Also:

IDIB

Return to the List of functions

IDNS Interface

IDNS provides a way to perform DNS (Domain Name System) queries. IDNS provides a more general interface to the DNS client than INETMGR_GetHostByName(). Alternate request types and compound queries are supported. The DNS client will keep track of DNS servers and handle retransmission and time-outs. IDNS converts domain names to the DNS protocol representation, and provides a method for decoding compressed domain names in the response. The user is responsible for specifying the content of DNS Question records, locating the data of importance in the response, and interpreting those values. UDP is the only transport supported. Requests and responses are limited to 512 bytes.

List of Header files to be included

The following header file is required:

aeedns.h

List of functions

Functions in this interface include:

IDNS_AddQuestion()
IDNS_AddRef()
IDNS_GetResponse()
IDNS_ParseDomain()
IDNS_QueryInterface()
IDNS_Release()
IDNS_Start()

The remainder of this section provides details for each function.

IDNS_AddQuestion()

Description:

This function adds a question to the set of question records in the request. This must be called before Start() is called. This can be called multiple times to construct a request message consisting of multiple question records.

Prototype:

```
int IDNS_AddQuestion (
      IDNS *pIDNS,
      AEEDNSType nType,
      AEEDNSClass nClass,
      const char *pszDomain
      )
```

Parameters:

nType	DNS question type
nClass	DNS class
pszDomain	Zero-terminated string representing a domain name in dotted notation. Single dot terminators as in “example.com.” are acceptable and treated identically to domain names without a terminating dot. NOTE: Domain search paths and relative domain names are not supported.)

Return value:

SUCCESS means that the question was appended to the request message.

EFAILED, if the new question would make the DNS message exceed the maximum size (512 bytes for DNS over UDP).

EBADPARM, if the domain name is malformed.

AEE_NET_EINVAL, if IDNS is not in the proper state for questions to be added. Questions cannot be added after Start() has been called.

Other errors may be returned; the caller should verify that IDNS_AddQuestion() succeeded.

Comments:

All domains are treated as fully-qualified domains by IDNS. There is no domain suffix search list; suffixes are never attached to the provided domain string. Colon (:), slash (/) and comma (,) characters do not delimit the domain name as in INETMGR_GetHostByName(). When constructing a question record, IDNS does not use the compressed form of domain names.

See Also:

AEEDNSType

AEEDNSClass

Return to the List of functions

IDNS_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IDNS_Release()

Return to the List of functions

IDNS_GetResponse()

Description:

This function is to be called to obtain the DNS response after the query completes. On success, it returns a pointer to a structure with the description of the DNS response. Response data includes pointers to structures; the referenced memory will be valid only for the lifetime of the IDNS object. None of those pointers should be retained or used after the IDNS interface has been released.

Prototype:

```
int IDNS_GetResponse(IDNS *pIDNS, const AEEDNSResponse **pResp);
```

Parameters:

pIDNS	[in]	Pointer to the IDNS Interface object
pResp	[out]	Pointer to a structure describing the response.

Return value:

SUCCESS: A response message was received from a DNS server; ***pResp** describes the result. Success here does not imply that the query was successful, just that a response was received. The application must inspect the AEEDNSResponse structure to determine whether the requested data is present in the response.

In all error cases, ***pResp** will point to an empty AEEDNSResponse structure. No assumptions should be made about the contents.

Error codes include:

ENOMEMORY if memory allocation failure prevented request or response

AEE_NET_ETIMEDOUT if retransmission time-out (no servers responded)

AEE_NET_ENETNONET, if socket-level error occurs

Comments:

None

See Also:

AEEDNSResponse

Return to the List of functions

IDNS_ParseDomain()

Description:

This function converts a DNS representation of a domain name into a zero-terminated string with dot (.) or dash (-) delimiter.

Prototype:

```
char *IDNS_*ParseDomain(IDNS *pIDNS, const byte *pbyDomain, int *pcb);
```

Parameters:

pIDNS	Pointer to the IDNS Interface object
pbyData	Pointer to the start of the domain name. This pointer must point into the DNS response data as described by an AEEDNSItem record. This can be used to decode **pbyDomain** values, or to decode values within the **pbyData[]** array.
pcb	Pointer to value to hold the number of bytes occupied by the domain name (in the source byte array, not in the resulting string). In the case of a malformed domain name, *pcb will be set to zero. If **pcb==NULL**, it will be ignored.

Return value:

Zero-terminated string giving the host name, or NULL on failure. Domain names are returned in dotted notation, with no terminating dot character at the end. Failure may be due to an allocation failure or a malformed domain name; *pcb can be used to distinguish between the two.

Comments:

None

See Also:

None

Return to the List of functions

IDNS_QueryInterface()

Description:

This function is inherited from IQI_QueryInterface().

See Also:

IDNS
Return to the List of functions

IDNS_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IDNS_AddRef()

Return to the List of functions

IDNS_Start()

Description:

This function starts the query, and schedule callback to be called. In order to prevent the callback from firing, and to cancel the operation, the user should release all references to the IDNS. Start() should not be called twice.

Prototype:

```
IDNS_Start(IDNS *pIDNS, PFNNOTIFY pfn, void *pcxt);
```

Parameters:

pIDNS	Pointer to the IDNS Interface object
pfn	Function to be called after the operation completes. This will only be called if the operation starts successfully (i.e. when Start() returned SUCCESS).
pv	Void pointer to be passed to the **pfn**() when it is called.

Return value:

error code:

SUCCESS, if the operation started. This should always succeed when called the first time.

AEE_NET_EINVAL, if this function is called more than once.

Comments:

Releasing all references to the IDNS object will cancel the operation and prevent the callback from being called.

See Also:

PFNNOTIFY

Return to the List of functions

IDownload Interface

This interface provides access to BREW application and file download mechanisms. It is for use ONLY by QUALCOMM, Handset Manufacturer and selected partners.

The IDownload interface consists of interface functions to query list(s) of application categories and/or items, download items and remove/disable items. The interface has been developed on the IWeb interface and shields the developer from all of the complexities involved in downloading applications.

List of functions

Functions in this interface include:

IDOWNLOAD_Acquire()
IDOWNLOAD_AutoDisable()
IDOWNLOAD_Cancel()
IDOWNLOAD_CheckItemUpgrade()
IDOWNLOAD_CheckUpgrades()
IDOWNLOAD_Continue()
IDOWNLOAD_Credit()
IDOWNLOAD_Delete()
IDOWNLOAD_Enum()
IDOWNLOAD_EnumRaw()
IDOWNLOAD_Get()
IDOWNLOAD_GetADSCapabilities()
IDOWNLOAD_GetADSList()
IDOWNLOAD_GetAllApps()
IDOWNLOAD_GetAppIDList()
IDOWNLOAD_GetAppIDListEx()
IDOWNLOAD_GetAutoDisableList()
IDOWNLOAD_GetAvailable()
IDOWNLOAD_GetCategory()
IDOWNLOAD_GetCategoryList()
IDOWNLOAD_GetConfigItem()
IDOWNLOAD_GetEULA()
IDOWNLOAD_GetHeaders()
IDOWNLOAD_GetItemInfo()
IDOWNLOAD_GetItemList()
IDOWNLOAD_GetModInfo()
IDOWNLOAD_GetSize()
IDOWNLOAD_GetSizeEx()

IDOWNLOAD_Lock()
IDOWNLOAD_LogEnumInit()
IDOWNLOAD_LogEnumNext()
IDOWNLOAD_OnStatus()
IDOWNLOAD_Restore()
IDOWNLOAD_Search()
IDOWNLOAD_SetADS()
IDOWNLOAD_SetHeaders()
IDOWNLOAD_SetSubscriberID()

The remainder of this section provides details for each function.

IDOWNLOAD_Acquire()

Description:

Asynchronously downloads an application and registers a billing record with the server for the associated price lt value.

Prototype:

```
void IDOWNLOAD_Acquire
      (
      IDownload * po,
      DLITEMID id,
      DLPRICEID idPrice,
      PFNDLCOMMAND pfn,
      void * pcxt
      )
```

Parameters:

po	Pointer to the IDownload interface object.
id	Application ID.
idPrice	Price value ID.
pfn	Pointer to the handle command function
pcxt	Pointer to the user define data

Return Value:

Upon completion, the callback specified is called with the associated completion code/error value. A code of 0 indicates success.

Comments:

None

Side Effects:

This call will attempt to initiate a network connection.

See Also:

None

Return to the List of functions

IDOWNLOAD_AutoDisable()

Description:

This function is uaed to auto-disable applications. Applications are"auto-disabled" in the following order...

1) List is scanned and applications are marked for "auto-disable" in least-recently-usedorder until enough space is recovered.

2) This sub-list is scanned backward and applications are "unmarked" if the space thenecessary space can be achieved without them.

This covers the following example:

Space Required: 33K

App A 10K

App B 11K

App C 23K

After Step 1, all three applications are marked for disable.

After Step 2, only App A and App C are marked. App B is no longer marked because it can be left enabled and the space can still be recovered.

Prototype:

```
int IDOWNLOAD_AutoDisable(IDownload * po,DLITEMID iID);
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Item ID for the operation.

Return Value:

SUCCESS - If applications are successfully auto-disabled

EFSFULL - Insufficient FS storage space (dwFSAvail < dwFSRequired)

ENOMEMORY - Insufficient contiguous RAM for the item (dwRAM < dwEstRAMRequired)

Comments:

None

See Also:

IDOWNLOAD_GetAutoDisableList()

IDOWNLOAD_Lock()

Return to the List of functions

IDOWNLOAD_Cancel()

Description:

Cancels all pending IDOWNLOAD operations.

Prototype:

```
void IDOWNLOAD_Cancel(IDownload * po)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_CheckItemUpgrade()

Description:

This function checks whether there is an upgrade for the given ItemID.

Prototype:

```
void IDOWNLOAD_CheckItemUpgrade
      (
      IDownload * po,
      DLITEMID id,
      PFNDLENUM pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
id	Item ID for the operation.
pfn	User callback to be called for item retrieved.
pcxt	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_CheckUpgrades()

Return to the List of functions

IDOWNLOAD_CheckUpgrades()

Description:

This function checks for upgrades for all the items.

Prototype:

```
void IDOWNLOAD_CheckUpgrades
      (
      IDownload * po,
      PFNDLENUM pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
pfn	User callback to be called for item retrieved.
pcxt	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_CheckItemUpgrade()

Return to the List of functions

IDOWNLOAD_Continue()

Description:

This method is called to indicate how the download engine should process an in-progress request. It is intended for use following a status callback of type DEVT_AI_ASK, DEVT_AI_DENY, DEVT_AI_SUCCESS, DEVT_AI_FAILURE.

Prototype:

```
void IDOWNLOAD_Continue(IDownload * po, boolean bContinue)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
bContinue	[in]	TRUE to continue.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_Credit()

Description:

This function credits the user.

Prototype:

```
void IDOWNLOAD_Credit
      (
      IDownload * po,
      const char * psz,
      PFNDLCOMMAND pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
psz	Credit-back access ticket.
pfn	User callback to be called for item retrieved.
pcxt	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_Delete()

Description:

Removes application files from persistent memory. If the boolean "bRemoveAllFiles" parameter is specified, all files and sub-directories for the application are removed. If not, the main resource and module files are removed.

Prototype:

```
int IDOWNLOAD_Delete
      (
      IDownload * po,
      DLITEMID id,
      boolean bRemoveAllFiles
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Application ID.
bRemoveAllFiles	[in]	Indicates whether all files should be removed.

Return Value:

SUCCESS if the item is disabled or removed.

EBADPARM if the itemnot found.

EITEMBUSY if the itemspecified is running/active.

EBADSID if the wrong subscriber attempts to remove item

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_Enum()

Description:

This is the post-1.01 mechanism that supports advanced category and item enumeration. The PFNDLENUM function callback is called for each item retrieved with a pointer to the item retreived. When the operation is complete, the callback is called one final time with a NULL DLEnumItem pointer.

Enumeration of the base/root category is initiated by calling the function with the DL_CATEGORY_ROOT item ID value.

Calls to this function passing an item ID for an item of type DLI_CATEGORY will enumerate the items inside that category. This allows the caller to enumerate the list of categories and applications using a mechanism similar to that used for file/directory enumeration.

A call to this function for an item of type other than DLI_CATEGORY will return the DLItemInfo for that particular item.

A call to this mechanism cancels any other pending calls.

NOTE: Caching of information is provided inside the protocol. There is no need to cache information. Information will not be retrieved from the network unless the cache is invalid.

Prototype:

```
void IDOWNLOAD_Enum
      (
      IDownload * po,
      DLITEMID id,
      PFNDLENUM pfn,
      void * pcxt
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Item ID for the operation.
pfn	[in]	User callback to be called for each item retrieved.
pcxt	[in]	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_EnumRaw()

Return to the List of functions

IDOWNLOAD_EnumRaw()

Description:

This is the post-1.0.1 mechanism that supports advanced category and item enumeration. Returns all possible purchasing method for every known application with out any checks such as inconsistent purchasing methods, if it is an already resident app, etc.,. The PFNDLENUM function callback is called for each item retrieved with a pointer to the item retreived. When the operation is complete, the callback is called one final time with a NULL DLEnumItem pointer.

Enumeration of the base/root category is initiated by calling the function with the DL_CATEGORY_ROOT item ID value. Calls to this function passing an item ID for an item of type DLI_CATEGORY will enumerate the items inside that category. This allows the caller to enumerate the list of categories and applications using a mechanism similar to that used for file/directory enumeration.A call to this function for an item of type other than DLI_CATEGORY will return the DLItemInfo for that particular item.A call to this mechanism cancels any other pending calls.

NOTE: Caching of information is provided inside the protocol. There is no need to cache information. Information will not be retrieved from the network unless the cache is invalid.

Prototype:

```
void IDOWNLOAD_EnumRaw
      (
      IDownload * po,
      DLITEMID id,
      PFNDLENUM pfn,
      void * pcxt
      );
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Item ID for the operation.
pfn	[in]	User callback to be called for each item retrieved.
pcxt	[in]	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_Enum()

Return to the List of functions

IDOWNLOAD_Get()

Description:

Asynchronously downloads an application from the server. This involves getting the package file, verifying it, and extracting the content files. The application's MIF file is the last file written to the file system.

Prototype:

```
void IDOWNLOAD_Get
      (
      IDownload *po,
      DLITEMID id,
      PFNDLCOMMAND pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
id	Application ID
pfn	Callback function pointer
pctxt	Pointer to be passed back in the callback function

Return Value:

Upon completion, the callback specified is called with the associated completion code/error value. A code of 0 indicates success.

Comments:

During operation several events are posted to the client application class specified by clsID. These events are:

EVT_DOWNLOAD_COMPLETE to nofity client that the download has finished, either successfully or not. Will be posted once.

Side Effects:

This call will attempt to initiate a network connection.

See Also:

IDOWNLOAD_Acquire()

Return to the List of functions

IDOWNLOAD_GetADSCapabilities()

Description:

This function returns the ADS capabilities. This allows the application to determine whether some menu items (such as search) should be displayed.

Prototype:

```
void IDOWNLOAD_GetADSCapabilities(IDownload * po)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

ADS_CAP_XXXX flags

0 = No ADS server connected

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetADSList()

Description:

This function returns a list of ADS servers. It is supported only on test enabled handsets. It is unsupported in production releases.

Prototype:

```
ADSInfoEntry * IDOWNLOAD_GetADSList(IDownload * po, int * pnCount)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
pnCount	[in/out]	Pointer to fill with count of servers. If NULL, the existing list is freed.

Return Value:

Pointer to list of ADS servers or NULL if unsupported.

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetAllApps()

NOTE: This function is a Version 1.0 legacy function. Use IDOWNLOAD_Enum() instead.

Description:

Retrieves all the known applications that has been installed

Prototype:

```
void IDOWNLOAD_GetAllApps(IDownload *po);
```

Parameters:

po	Pointer to the IDownload interface object.

Return Value:

None

Comments:

Upon completion the callback is called with the appropriate error value.

If the error code is 0 (success), the item list pointer provided is

valid.

See Also:

IDOWNLOAD_GetItemList()

Return to the List of functions

IDOWNLOAD_GetAppIDList()

Description:

This function returns a 0-terminated list of application IDs. Each DLITEMID can be used to query specific information about the application via the IDOWNLOAD_GetModInfo function.

Prototype:

```
DLITEMID * IDOWNLOAD_GetAppIDList(IDownload * po)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

On SUCCESS, returns a 0-terminated list of DLITEMID values. This list is valid until a subsequent call is made to IDOWNLOAD_GetAppIDList or the interface is released.

On FAILURE, returns NULL.

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetAppIDListEx()

Description:

This function returns a NULL terminated list of application IDs including those that are protected (modules that cannot be deleted by the user). Each DLITEMID can be used to query specific information about the application via the IDOWNLOAD_GetModInfo() function.

Prototype:

```
DLITEMID * IDOWNLOAD_GetAppIDListEx(IDownload *po);
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

SUCCESS: Returns a NULL terminated list of DLITEMID values.

FAILURE: Returns NULL

Comments:

The list returned is valid until a subsequent call is made to IDOWNLOAD_GetAppIDListEx() or the interface is released.

See Also:

IDOWNLOAD_GetAppIDList()

Return to the List of functions

IDOWNLOAD_GetAutoDisableList()

Description:

This function retrieves the list of entries that can be auto-disabled. The idWant/dwExtra parameters can be specified in order to mark those items that would be candidates to disable based upon size and date/time last used.

The list is returned sorted in least-recently used order. By using DLITEMID, this call is equivalent to the IDOWNLOAD_AutoDisable() function without the function automatically disabling the items.

Prototype:

```
DLDisableEntry * IDOWNLOAD_GetAutoDisableList
      (
      IDownload * po,
      DLITEMID idWant,
      uint32 dwExtra,
      int * pnCount,
      int * pnErr
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
idWant	[in]	ID of item that may be downloaded. 0 if all entries
dwExtra	[in]	Extra number of bytes desired.
pnCount	[out]	Number of entries in the list
pnErr	[out]	Pointer to error

Return Value:

Returns the list that can be auto-disabled

NULL - Indicates an error *pnErr:

SUCCESS - idWant + dwExtra size is available in file system

EFSFULL - idWant + dwExtra cannot be satisified by disable

Comments:

None

See Also:

IDOWNLOAD_Lock()

Return to the List of functions

IDOWNLOAD_GetAvailable()

Description:

This method is called to populate available file and RAM space for potential downloads.

Prototype:

```
int IDOWNLOAD_GetAvailable(IDownload * po,DLSizeInfo * psi);
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
psi	[out]	Size information

Return Value:

AEE_SUCCESS if successful

EFSFULL - Insufficient FS storage space
(dwFSAvail < dwFSRequired)

ENOMEMORY - Insufficient contiguous RAM for the item
(dwRAM < dwEstRAMRequired)

EOUTOFNODES - Insufficient file handles available
(nFilesAvail < nEstFileRequired)

EBADPARM - If psi is NULL or the package is invalid

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetCategory()

NOTE: This function is a Version 1.0 legacy function depricated. Use IDOWNLOAD_Enum() instead.

Description:

Asynchronously retrieves the list of categories for the specified category ID.

Prototype:

```
void IDOWNLOAD_GetCategory
      (
      IDownload *po,
      DLCATID id,
      PFNDLITEMLIST pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
id	Category ID (0 - all)
pfn	Pointer to callback function.
pctxt	Pointer to user data context passed as first argument to function.

Return Value:

None

Comments:

Upon completion the callback is called with the appropriate error value.

If the error code is 0 (success), the item list pointer provided is

valid.

Side Effects:

This call will attempt to initiate a network connection.

See Also:

IDOWNLOAD_Enum()

Return to the List of functions

IDOWNLOAD_GetCategoryList()

NOTE: This function is a Version 1.0 legacy function depricated. Use IDOWNLOAD_Enum() instead.

Description:

Asynchronously retrieves the list of categories available on the server.

Prototype:

```
void IDOWNLOAD_GetCategoryList
      (
      IDownload * po,
      PFNDLCATEGORYLIST pfn,
      void * pcxt
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

Upon completion, the callback is called with the appropriate error value.

If the error code is 0 (success), the category list pointer provided is valid.

Comments:

None

Side Effects:

This call will attempt to initiate a network connection.

See Also:

IDOWNLOAD_Enum()

Return to the List of functions

IDOWNLOAD_GetConfigItem()

Description:

This function retrieves the device configuration information related to the download services.

Prototype:

```
int IDOWNLOAD_GetConfigItem
      (
      IDownload * po,
      int i,
      void * pItem,
      int nSize
      );
```

Parameters:

po	[in]		Pointer to the IDownload interface object.
i	[in]	is fully consistent with OEM_GetConfig() changes.	
		CFGI_AUTOSTART,	AEECLSID - Auto-Started Applet (AEE_Init())
		CFGI_BUSY_CURSOR_OFFSET,	Position of hourglass (AEERect *)
	char	CFGI_CARDID	Unique identifier that identifies the attached card. The input buffer for this will be the size returned by CFGI_CARDID_LEN
	int	CFGI_CARDID_LEN	Size in bytes of the CARDID (For e.g., RUIM./SIM)
		CFGI_CLOSE_KEYS	OEMCloseKeys * (see structure below)
		CFGI_DATA_NETWORK	OEMDataNetwork *
		CFGI_DEBUG_KEY	OEMDebugKey * see OEM_GetConfig()
		CFGI_DISALLOW_DORMANCY	boolean, if TRUE, disallow dormancy,
		CFGI_DNS_IP1	32-bit IP, Domain Name Server(1) in network byte-order
		CFGI_DNS_IP2	32-bit IP, Domain Name Server(2) in network byte-order
		CFGI_DORMANCY_NO_SOCKETS	boolean, whether to hold PPP (go dormant) even if no sockets are open
		CFGI_DOWNLOAD	AEEDownloadInfo

	CFGI_DOWNLOAD_BUFFER,	Size in bytes to buffer data during download before calling fs_write (default 10K)
	CFGI_DOWNLOAD_FS_INFO,	DLItemSize * (Fill dwFSAvail, dwFSSize)
	CFGI_FILE_CACHE_INFO,	OEMFileCacheInfo * (see structure below)
	CFGI_FIRST_OEM=CFGI_MAX	OEM added config items should start at this value
uint32	CFGI_GPSONE_LOCK,	GPS lock
uint32	CFGI_GPSONE_SVRIP,	GPS server IP address
uint32	CFGI_GPSONE_SVRPORT,	GPS server IP port
uint32	CFGI_GPSONE_TRANSPORT,	OEM GPS transport (IP, Data burst)
	CFGI_HTTP_BUFFER,	Size in bytes of HTTP read buffer (default 4K)
	CFGI_MAX,	Holds max AEE value, not a function
	CFGI_MAX_DISPATCH_TIME,	Maximum time BREW should spend in the dispatcher before relinquishing control (default = 250 msecs)
	CFGI_MIN_IDLE_TIME,	Minimum time BREW must relinquish from dispatcher (default = 35 msecs)
	CFGI_MOBILEINFO,	AEEMobileInfo
MIFFS Limit	CFGI_MODULE_FSLIMIT,	This identifies the maximum files and maximum space that can be used up by a module. The default value for these are set to the maximum permissible limit. MIFFSLimit is broken down into the following subcomponents: Subcomponents - Type - Description wMaxFiles - uint16 - Maximum number of files in EFS this module is allowed to create dwMaxSpace - uint32 - Maximum EFS space this module is allowed to consume
	CFGI_NET_CONNTIMEOUT,	time in milliseconds! to wait for connect()
	CFGI_PROVISION_FIRST=0x1000,	Offset to build dependent items
	CFGI_PROVISION_LAST=0x2000,	End of build dependent items
	CFGI_SCREEN_SAVER,	AEEScreenSaverInfo *

	CFGI_SLEEP_TIMER_RESOLUTIO,	Timer resolution during when processor/os is in SLEEP mode (default = 1.2 seconds)
	CFGI_SUBSCRIBERID,	32-byte ASCIIZ
	CFGI_SUBSCRIBERID_LEN	Size in bytes of subscriber ID. The default used if error returned. The NULL terminator is counted in the count returned
uint32	CFGI_SYSMEM_SIZE	Size in bytes reserved to the system in low-memory (default = 2K)

Return Value:

None

Comments:

None

See Also:

OEM_GetConfig()

Return to the List of functions

IDOWNLOAD_GetEULA()

Description:

This function gets the EULA for the given ItemID. When the text is fetched, it invokes the callback function passed as argument to this function.

Prototype:

```
void IDOWNLOAD_GetEULA
      (
      IDownload * po,
      DLITEMID id,
      PFNDLTEXT pfn,
      void * pcxt
      );
```

Parameters:

po	Pointer to the IDownload interface object.
id	Item ID for the operation.
pfn	User callback to be called for item retrieved.
pcxt	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

PFNDLTEXT

Return to the List of functions

IDOWNLOAD_GetHeaders()

Description:

Returns the current HTTP headers set via the IDOWNLOAD_SetHeaders() call.

Prototype:

```
const char * IDOWNLOAD_GetHeaders(IDownload * po)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

NULL: No headers set

Header strings

Comments:

None

See Also:

IDOWNLOAD_SetHeaders()

Return to the List of functions

IDOWNLOAD_GetItemInfo()

Description:

This is the post-1.01 mechanism that allows the caller to query information about the item associated the specified item ID. The download engine will either retrieve cached information regarding the item or request the information from the server.

This call is passed a user callback. This callback will be called when the information for the associated item has been retreived.

Unlike a call to IDOWNLOAD_Enum, this call will retrieve information ONLY about the specified item. It will not enumerate the contents of an item of type DLI_CATEGORY.

A call to this mechanism cancels any other pending calls.

NOTE: Caching of information is provided inside the protocol. There is no need to cache information. Information will not be retrieved from the network unless the cache is invalid.

Prototype:

```
void IDOWNLOAD_GetItemInfo
      (
      IDownload * po,
      DLITEMID id,
      PFNDLENUM pfn,
      void * pcxt
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Item ID for the operation.
pfn	[in]	User callback to be called for each item retrieved.
pcxt	[in]	User context handle passed as first parameter to callback.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetItemList()

Description:

** Version 1.0 Legacy Function - Use IDOWNLOAD_Enum() instead * *

Asynchronously retrieves the list of applications for the specified category ID. Note that if the specified id is 0, all known applications will be retrieved.

Prototype:

```
void IDOWNLOAD_GetItemList
      (
      IDownload * po,
      DLCATID id,
      DLItemType t,
      PFNDLITEMLIST pfn,
      void * pcxt
      )
```

Parameters:

po	Pointer to the IDownload interface object.
id	Category ID (0 - all)
t	Type of item to download (0 - all)
pfn	Pointer to callback function.
pctxt	Pointer to user data context passed as first argument to function

Return Value:

Upon completion the callback is called with the appropriate error value.

If the error code is 0 (success), the item list pointer provided is valid.

Comments:

None

Side Effects:

This call will attempt to initiate a network connection.

See Also:

None

Return to the List of functions

IDOWNLOAD_GetModInfo()

Description:

This function allocates and returns a structure containing information regarding applications associated with a particular DLITEMID.

Prototype:

```
AppModInfo * IDOWNLOAD_GeModInfo(IDownload * po, DLITEMID appID)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
appID	[in]	Application ID (returned from IDOWNLOAD_GetAppIDList()).

Return Value:

SUCCESS: Returns a pointer to the AppModInfo. This pointer is valid until a subsequent call is made to IDOWNLOAD_GetModInfo or the interface is released.

FAILURE: NULL.

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetSize()

Description:

This function returns the total size required to download any missing portions of the specified item. If non-NULL, the pdwTotal is filled with the total size of all files for the item.

Prototype:

```
uint32 IDOWNLOAD_GetSize
      (
      IDownload * po,
      DLITEMID iID,
      uint32 * pdwTotal
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
iID	[in]	Item ID.
pdwTotal	[out]	Total bytes of all files (resident and non-resident).

Return Value:

Size of non-resident file(s) to be downloaded.

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_GetSizeEx()

Description:

This method calculates the size required to store an item of 1-N packages. It returns an error if the size required is unavailable on the device.

Prototype:

```
int IDOWNLOAD_GetSizeEx
      (
      IDownload * po,
      DLITEMID iID,
      DLItemSize * psi
      );
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
iID	[in]	Item ID to check
psi	[out]	Size information

Return Value:

SUCCESS - If the size is obtained

EFSFULL - Insufficient FS storage space (dwFSAvail < dwFSRequired)

ENOMEMORY - Insufficient contiguous RAM for the item (dwRAM < dwEstRAMRequired)

EOUTOFNODES - Insufficient file handles available (nFilesAvail < nEstFileRequired)

EBADPARM - If psi is NULL or the package is invalid

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_Lock()

Description:

This function is used to either lock or unlock a module. Locking a module prevents it from being auto-disabled.

Prototype:

```
boolean IDOWNLOAD_Lock(IDownload * po, DLITEMID id, boolean bLock);
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Item ID for the operation.
bLock	[in]	TRUE if needs to be locked, FALSE otherwise

Return Value:

TRUE - Module locked or unlocked

FALSE - Lock or unlock failed

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_LogEnumInit()

Description:

Initializes the enumeration of download log entries.

Prototype:

```
int IDOWNLOAD_LogEnumInit(IDownload * po)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.

Return Value:

SUCCESS - Successfully initialized

EFAILED - Log file does not exist.

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_LogEnumNext()

Description:

Returns the next available log entry. Returns FALSE if there are no more log items to enumerate.

Prototype:

```
boolean IDOWNLOAD_LogEnumNext(IDownload * po, DLLogItem * pli)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
pli	[in/out]	Pointer to log item entry to fill.

Return Value:

TRUE - success

FALSE - no more items

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_OnStatus()

Description:

This function is passed a function callback that is called when any activity occurs as a result of calls to the IDownload class. This includes status regarding recalled applications, download status, etc.

Prototype:

```
void IDOWNLOAD_OnStatus(IDownload * po, PFNDLSTATUS pfn, void * pUser)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
pfn	[in]	Pointer to status callback function.
pUser	[in]	Pointer to user data context passed as first argument to function.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_Restore()

Description:

Asynchronously restores an applet that has been removed to save memory. The application is downloaded free of charge and no billing transaction is generated.

Prototype:

```
void IDOWNLOAD_Restore
      (
      IDownload * po,
      DLITEMID id,
      PFNDLCOMMAND pfn,
      void * pcxt
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
id	[in]	Application ID.

Return Value:

Upon completion, the callback specified is called with the associated completion code/error value. A code of 0 indicates success.

Comments:

None

Side Effects:

This call will attempt to initiate a network connection.

See Also:

None

Return to the List of functions

IDOWNLOAD_Search()

Description:

This is the post-1.01 function initiates a search of ADS items that fulfill the search criteria specified. The return value works identically to the IDOWNLOAD_Enum() function.

Prototype:

```
void IDOWNLOAD_Search
      (
      IDownload * po,
      const AECHAR * psz,
      DLSearchType st,
      PFNDLENUM pfn,
      void * pcxt
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
psz	[in]	Search string. The string specified is a comma separated list of keywords,
st	[in]	Search type (ANY, ALL).
pfn	[in]	Search callback.
pcxt	[in]	User context.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_Enum()

Return to the List of functions

IDOWNLOAD_SetADS()

Description:

This function allows the caller to set the server that the download mechanism will use. It is provided ONLY for debug purposes and is NOT supported on production handsets.

Prototype:

```
boolean IDOWNLOAD_SetADS(IDownload * po, ADSInfo * ps)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
ps	[in]	Pointer to structure containing server information.

Return Value:

TRUE - success

FALSE - function not supported

Comments:

None

See Also:

None

Return to the List of functions

IDOWNLOAD_SetHeaders()

Description:

This is the post-1.01 function allows the application to specify a special HTTP headers that will be sent to the ADS on all requests. The format is as follows:

"Name:Val\r\nName:Val\r\nName:Val\r\n"

Passing a NULL value for the header removes the extra headers.

Prototype:

```
void IDOWNLOAD_SetHeaders(IDownload * po, const char * pszHeader)
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
pszHeaders	[in]	Pointer to HTTP headers.

Return Value:

None

Comments:

None

See Also:

IDOWNLOAD_GetHeaders()

Return to the List of functions

IDOWNLOAD_SetSubscriberID()

Description:

This function allows the caller to set the subscriber ID.

Prototype:

```
void IDOWNLOAD_SetSubscriberID
      (
      IDownload * po,
      const char * pszSID,
      int nSize
      )
```

Parameters:

po	[in]	Pointer to the IDownload interface object.
pszSID	[in]	Pointer to SID.
nSize	[in]	Size in bytes of subscriber ID (if <= 0 assumed to be DEFAULT_SUBSCRIBERID_LEN).

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

IFont Interface

This interface provides functions for drawing and measuring text. Applications will not typically call IFont member functions directly. IDisplay provides a friendlier and more convenient interface to drawing text, with support for system colors and clipping state, as well as many features related to drawing text, such as underlining, centering, framing, and erasing backgrounds. IFont's functions are stricter about arguments and less flexible in defaulting values. As IDisplay builds on IFont and any IFont can be passed to IDisplay for it to manage, using IFont directly offers no increase in functionality.

IFont is an interface with multiple implementations. Some IFont classes might work only with a particular class of IBitmap, whereas others will work with any IBitmap that supports 1-bit blits. By default, IDisplay uses IFont objects provided by the OEM. These can differ in size and appearance from handset to handset, and some may not support off-screen bitmaps.

List of Header files to be included

The following header file is required:

AEEFont.h

List of functions

Functions in this interface include:

IFONT_AddRef()
IFONT_DrawText()
IFONT_GetInfo()
IFONT_MeasureText()
IFONT_QueryInterface()
IFONT_Release()

The remainder of this section provides details for each function.

IFONT_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IFONT_Release()

Return to the List of functions

IFONT_DrawText()

Description:

This function draws text into a bitmap. The x and y parameters describe the placement of the top-left corner of the left most character cell.

The drawing operation is limited to the rectangle ***prcClip**. Any portions of the text that fall outside of this rectangle (or outside the bounds of the bitmap) will not be drawn. Clipping affects whether or not pixels are drawn; clipping never affects where things are drawn.

All parameters to IFONT_DrawText() should be valid. Invalid or special values are not interpreted to have special meanings, as in IDisplay. The **dwFlag** parameter consists of a set of bit flags that select certain options. The values are defined as for IDISPLAY_DrawText(), but only IDF_TEXT_TRANSPARENT is supported. When IDF_TEXT_TRANSPARENT is specified, only the foreground portion (i.e. the graphical representation the character) is drawn over any pixels previously occupying the destination rectangle. Otherwise, each character cell is drawn in its entirety (both foreground and background pixels).

Prototype:

```
int IFONT_DrawText
      (
      IFont *pIFont,
      IBitmap *pDst,
      int x,
      int y,
      const AECHAR *pcText,
      int nChars,
      NativeColor foreground,
      NativeColor background,
      const AEERect *prcClip,
      uint32 dwFlags
      )
```

Parameters:

pIFont	[in]	Pointer to the IFont Interface.
pDst	[in]/[out]	Pointer to the destination IBitmap
x	[in]	X coordinate of the text string.
y	[in]	Y coordinate of the text string.
pcText	[in]	Text string to be drawn.
nChars	[in]	Text string length. If this is -1, then the length is automatically computed by this function
foreground	[in]	Color to draw text.
background	[in]	Color to draw background.
prcClip	[in]	Clipping rectangle in which the text string must be drawn.
dwFlags	[in]	Properties bitmap that dictates the appearance of the text display.

Return Value:

SUCCESS, if successful.

Error code, if otherwise

EUNSUPPORTED, if the underlying IBitmap does not support operations required by the font.

Other implementation-specific error codes

Comments:

Negative **x** and **y** values are legal and indicate a starting position to the left of or above the top of the bitmap. In such cases, any portion of the text that extends into the bitmap (and into ***prcClip**) is drawn.

When the width or height of ***prcClip** is negative, nothing is drawn.

The foreground and background color values are NativeColor values and not **RGBVAL** values.

These can be obtained by calling the destination bitmap's IBITMAP_RGBToNative() member function.

See Also:

NativeColor

IBITMAP_RGBToNative()

IFONT_MeasureText()

Return to the List of functions

IFONT_GetInfo()

Description:

This function fills the AEEFontInfo structure with information about the font. The ascent and descent values are returned.

The size of the structure is passed for backward compatibility. The implementation should only fill the structure up to the specified size. If the size is larger that than the sizeof(AEEFontInfo), this function should return EUNSUPPORTED.

Prototype:

```
int IFONT_GetInfo(IFont * pIFont, AEEFontInfo * pinfo, int nSize)
```

Parameters:

pIFont	[in]	Pointer to the IFont Interface.
pinfo	[out]	Pointer to the AEEFontInfo structure to fill.
nSize	[in]	Size of structure to fill.

Return Value:

SUCCESS, if font info is retrieved.

EUNSUPPORTED, if the version determined by the **nSize** is not supported.

Comments:

IFONT_GetInfo() should always succeed when a valid **pinfo** pointer is passed and **nSize** is equal to sizeof(AEEFontInfo).

See Also:

AEEFontInfo

IFONT_MeasureText()

Return to the List of functions

IFONT_MeasureText()

Description:

This function measures the width of text. It calculates the number of characters that fit in the given maximum width as well as the number of pixels that those characters take up. All parameters to IFONT_MeasureText() must be valid. Invalid or special values are not interpreted to have special meanings.

Prototype:

```
int IFONT_MeasureText
      (
      IFont *pIFont,
      const AECHAR *pcText,
      int nChars,
      int nMaxWidth,
      int *pnFits,
      int *pnPixels
      )
```

Parameters:

pIFont	[in]	Pointer to the IFont Interface.
pcText	[in]	Text string to be measured in pixels.
nChars	[in]	The number of AECHAR characters to measure. Zero (0) indicated an empty string (width 0 (zero)) Negative values may not be specified.
nMaxWidth	[in]	Maximum available screen width in pixels.
pnFits	[out]	Number of characters that can fit in the screen of the given available width.
pnPixels	[out]	Total width of the text string in pixels that can fit in the available space.

Return Value:

SUCCESS, if successful

Error Code, if unsuccessful

Comments:

None

See Also:

IFONT_GetInfo()

Return to the List of functions

IFONT_QueryInterface()

Description:

This function retrieves a pointer to an interface conforming to the definition of the specified class ID. This can be used to query for extended functionality, like future versions or proprietary extensions.

Upon a successful query, an instance of the interface is returned. The caller is responsible for calling Release() at some point in the future. One exception is when the pointer returned is not an interface pointer. In that case, the memory will share the lifetime of the object being queried, and the returned pointer will not be used to free or release the object.

Prototype:

```
int IFONT_QueryInterface(IFont * pIFont, AEECLSID id, void ** p)
```

Parameters:

pIFont	[in]	Pointer to the IFont Interface.
id	[in]	A globally unique ID to identify the entity (interface or data) that is to be queried.
p	[out]	Pointer to the data or interface that is to be retrieved. If the value passed back is NULL, the interface or data being queried is not available.

Return Value:

SUCCESS, on success,

ECLASSNOTSUPPORT, if class ID not supported

Comments:

On failure, QueryInterface() must set *p to NULL.

See Also:

None

Return to the List of functions

IFONT_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IFONT_AddRef()

Return to the List of functions

IGSM1xControl Interface

IGSM1xControl interface enables GSM1x capability on the mobile. It contains interfaces that are the building blocks for GSM1x Activation BREW Applet.

The interfaces support provisioning of GSM1x data from SIM or R-UIM to a specially designated NAM in NV as well as switching between NAMs at run time.

Typically, GSM1x Activation BREW Applet will follow the following sequence of operations:

- Find out the current mode using IGSM1xControl_GetCurrentMode().
- Find out which modes are available using IGSM1xControl_GetAvailableModes().
- If no provisioning modes are available, go to "Emergency calls only" state using IGSM1xControl_ActivateNonGSM1xMode() and exit.
- Choose the new provisioning mode either automatically or by interacting with the user.
- If the new mode is CDMA1x, select it using IGSM1xControl_ActivateNonGSM1xMode().
- If the new mode is GSM1x, do the following sequence (order is important) of operations:
  - Call IGSM1xControl_ProvisionGSM1xParameters() to provision IMSI, ACCOLC and MSISDN.
  - Form GSM1x PRL and set it using IGSM1xControl_SetGSM1xPRL().
  - Set home and locked SID/NID pairs using IGSM1xControl_SetGSM1xSIDNIDPairs()
  - Call IGSM1xControl_EnableGSM1xMode().

GSM1x Activation BREW Applet can, optionally, provide interactive editing capability for PLMN selector and forbidden PLMN information stored in SIM/R-UIM.

In addition to its core functionality, this class is used to signal GSM1x BREW Applets (other than GSM1x Activation BREW Applet) whenever GSM1x mode is activated or de-activated. IGSM1xControl sends event notification whenever GSM1x mode changes (enabled or disabled). Whenever a new mode is activated on IGSM1xControl it sends out a NMASK_GSM1xCONTROL_STATUS_CHANGE notification. AEEGSM1xControl_statusType is sent as dwParam member of the EVT_NOTIFY event.

To send out notification, a helper class IGSM1xControlNotifier is used. Methods for IGSM1xControlNotifier should not be called directly by BREW applets. Brew applet should specify the class id for IGSM1xControlNotifier and the NMASK=NMASK_GSM1xCONTROL_STATUS_CHANGE in its MIF file.

In general, all GSM1x applets should behave as follows:

On Init

- Register to receive NMASK_GSM1xCONTROL_STATUS_CHANGE.
- Call IGSM1xSig_GetStatus()to ensure that GSM1x capability is enabled.
  If GSM1x capability is disabled, application can exit with an appropriate message or wait until it receives a GSM1xSIG_STATUS_CHANGE event with GSM1xSIG_ACTIVE.

Runtime

- If a GSM1xSIG_STATUS_CHANGE event is received, handle it appropriately.

The IGSM1xControl interface is obtained via the ISHELL_CreateInstance() mechanism.

List of Header files to be included

The following header file is required:

AEEGSM1xControl.h

List of functions

Functions in this interface include:

IGSM1xControl_ActivateNonGSM1xMode()
IGSM1xControl_EnableGSM1xMode()
IGSM1xControl_GetAvailableModes()
IGSM1xControl_GetCurrentMode()
IGSM1xControl_GetDFPresence()
IGSM1xControl_GetGSM1xPRL()
IGSM1xControl_GetGSM1xSIDNIDPairs()
IGSM1xControl_GetPLMN()
IGSM1xControl_GetSupportedProvisioningModes()
IGSM1xControl_GetUIMUniqueId()
IGSM1xControl_ProvisionGSM1xParameters()
IGSM1xControl_SetGSM1xPRL()
IGSM1xControl_SetGSM1xSIDNIDPairs()
IGSM1xControl_SetPLMN()
IGSM1xControl_ValidatePRL()

The remainder of this section provides details for each function.

See Also:

IGSM1xSig Interface
IQueryInterface
Return to the IGSM1xControl Interface

IGSM1xControl_ActivateNonGSM1xMode()

This function

This function activates the specified provisioning mode:

AEEGSM1XCONTROL_1X_NV_PROV_MASK

AEEGSM1XCONTROL_1X_RUIM_PROV_MASK

AEEGSM1XCONTROL_EMERGENCY_PROV_MASK

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_ActivateNonGSM1xMode
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_DFPresenceBitMaskType  mode
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl Interface object.
mode	The desired provisioning mode.

Return Value:

AEEGSM1xControl_statusType

Comments:

This function is usually called by the GSM1x Activation App when a user desires to switch to CDMA 1x provisioning mode..

Side Effects:

NMASK_GSM1xSIG_STATUS_CHANGE notification sent to all registered apps.

See Also:

None

Return to the List of functions

IGSM1xControl_EnableGSM1xMode()

This function

This routine commands DMSS's Call Manager module to switch to the GSM1x NAM. Call Manager will take the phone offline and then re-do the system determination.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_EnableGSM1xMode
      (
      IGSM1xControl *instancePtr
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

This routine is synchronious - it waits till the phone finishes system determination (that can take significant amount of time.) This function is usually called by the GSM1x Activation App as the final step in activating GSM1x mode.

Side Effects:

NMASK_GSM1xCONTROL_STATUS_CHANGE notification is sent to all registered applications.

See Also:

None

Return to the List of functions

IGSM1xControl_GetAvailableModes()

Description:

This function returns (as a bit mask) which of the provisioning modes are currently available for selection. This is determined by the software build and by the presence and the type of the User Identity Module.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetAvailableModes
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_modeBitMaskType *modeMask
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
modeMask	Pointer to a memory location to receive the bitmask which indicates the supported modes.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

None

Return to the List of functions

IGSM1xControl_GetCurrentMode()

Description:

This function retrieves the current provisioning mode.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetCurrentMode
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_modeBitMaskType *modeMask
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
modeMask	Pointer to a memory location to receive the current provisioning mode.

Return Value:

AEEGSM1xControl_statusType- the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

Can be used by the GSM1x Activation App to display the current provisioning mode.

See Also:

None

Return to the List of functions

IGSM1xControl_GetDFPresence()

Description:

This function returns the indication which directory files (DFs) are present on the currently available user identity card (SIM or R-UIM.)

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetDFPresence
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_DFPresenceBitMaskType *presenceMask
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
presenceMask	Pointer to a memory location to receive the bitmask that indicates which DFs are present.

Currently, the following DFs are supported:

AEEGSM1XCONTROL_MF_PRESENT

AEEGSM1XCONTROL_CDMA_DF_PRESENT

AEEGSM1XCONTROL_GSM_DF_PRESENT

To be supported in a future release:

AEEGSM1XCONTROL_DCS1800_DF_PRESENT

The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

AEEGSM1XCONTROL_DCS1800_DF_PRESENT is currently not supported, its mask will not be returned by this call.

See Also:

None

Return to the List of functions

IGSM1xControl_GetGSM1xPRL()

Description:

This function returns the packed PRL stored in the NAM assigned to GSM1x. Format of the PRL depends on the software build IS683A or IS683C).

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetGSM1xPRL
      (
      IGSM1xControl *pIGSM1xControl,
      uint16 maxPRLSizeBytes,
      byte *packedPRL,
      )
```

Parameters:

`pIGSM1xControl`	Pointer to the IGSM1xControl object.
maxPRLSizeBytes	specifies the maximum size of the buffer pointed by packedPRL.
packedPRL	Pointer to a memory location to receive the PRL. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned valuesare defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_SetGSM1xPRL()

Return to the List of functions

IGSM1xControl_GetGSM1xSIDNIDPairs()

Description:

This function retrieve Home and Locked SID/NID pairs stored in NV.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetGSM1xSIDNIDPairs
      (
      IGSM1xControl *pInstance,
      uint16 HomeSIDNIDMaxCnt,
      AEEGSM1xControl_SIDNIDPairType *HomeSIDNIDPairs,
      uint16 *ActualHomeSIDNIDCnt,
      uint16 LockedSIDNIDMaxCnt,
      AEEGSM1xControl_SIDNIDPairType *LockedSIDNIDPairs,
      uint16 *ActualLockedSIDNIDCnt
      )
```

Parameters:

pInstance	Pointer to the IGSM1xControl object.
HomeSIDNIDMaxCnt	Maximum number of SID/NID pairs that can be stored in the memory location pointed by HomeSIDNIDPairs.
HomeSIDNIDPairs	Pointer to a memory location to receive Home SID/NID pairs. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.
ActualHomeSIDNIDCnt	Pointer to a memory location to receive the actual count of Home SID/NID pairs written to HomeSIDNIDPairs. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.
LockedSIDNIDMaxCnt	Maximum number of SID/NID pairs that can be stored in the memory location pointed by LockedSIDNIDPairs.
LockedSIDNIDPairs	Pointer to a memory location to receive Locked SID/NID pairs. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.
ActualLockedSIDNIDCnt	Pointer to a memory location to receive the actual count of Home SID/NID pairs written to LockedSIDNIDPairs. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_SetGSM1xSIDNIDPairs()

Return to the List of functions

IGSM1xControl_GetPLMN()

Description:

This function reads from a SIM or R-UIM card and returns the PLMN entries for all types specified by the provided bitmask.

A Home PLMN is retrieved from the EFimsi counting the EFad.

A PLMN Selector is retrieved from the EFplmnsel.

A Forbidden PLMN is retrieved from the EFfplmn.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetPLMN
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_PLMNTypeBitMaskType types,
      uint16 maxPLMNEntriesCnt,
      AEEGSM1xControl_PLMNTripletType *PLMNBuf,
      uint16 *actualPLMNEntriesCnt
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object. types specifies a bitmask that specifies which types of PLMN information are requested AEEGSM1XCONTROL_HOME_PLMN AEEGSM1XCONTROL_SEL_PLMN AEEGSM1XCONTROL_FORBIDDEN_PLMN) .
maxPLMNEntriesCnt	Maximum number of PLMN entries that can fit into the location pointed by PLMNBuf.
PLMNBuf	Pointer to a memory location to receive the array of PLMN elements. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.
actualPLMNEntriesCnt	Pointer to a memory location to receive the actual number of entries copied into the location pointed by PLMNBuf. The contents of the location pointed by actualPLMNEntriesCnt are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_SetPLMN()

Return to the List of functions

IGSM1xControl_GetSupportedProvisioningModes()

Description:

This function returns the bit mask that indicates which of the possible provisioning modes are supported by the software (OEM). Note, that this calls does not take into account the presence and the type of a User Identity Module (smartcard.) Thus, even if some mode are supported by the software, user might not be able to use them because the user does not have a "right" UIM.

Prototype:

```
AEEGSM1xControl_statusType
IGSM1xControl_GetSupportedProvisioningModes
      (
      IGSM1xControl *instancePtr,
      AEEGSM1xControl_DFPresenceBitMaskType *modeMask
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
modeMask	Bit mask that contains zero or more values AEEGSM1XCONTROL_GSM1X_PROV_MASK, AEEGSM1XCONTROL_1X_NV_PROV_MASK, AEEGSM1XCONTROL_1X_RUIM_PROV_MASK.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix

AEEGSM1XCONTROL_STATUS_

Comments:

Relies on the implementation provided by OEMs.

See Also:

None

Return to the List of functions

IGSM1xControl_GetUIMUniqueId()

Description:

This function returns the unique ICCId stored in EFiccid field on SIM or R-UIM. The normal length is 10 bytes.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_GetUIMUniqueId
      (
      IGSM1xControl *pInstance,
      uint16 maxBufLen,
      byte *pId,
      uint16 *actualLen
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
maxBufLen	Maximum length of a memory buffer pointed by pId.
pId	Pointer to a memory location to receive the ICCId. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned. This location should have at least 10 bytes available.
actualLen	Pointer to a memory location to receive the actual length of ICCId. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None.

See Also:

None.

Return to the List of functions

IGSM1xControl_ProvisionGSM1xParameters()

Description:

This routine reads GSM IMSI, ACCOLC, and MSISDN from the present UIM card if any), converts it to CDMA IMSI, ACCOLC and MSISDN according to GSM1x provisioning algorithm, and writes the results into in NV, associated with GSM1x NAM.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_ProvisionGSM1xParameters
      (
      IGSM1xControl *instancePtr
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

None

Return to the List of functions

IGSM1xControl_SetGSM1xPRL()

Description:

Description:

This function validates supplied packed PRL and (if valid) writes it to the GSM1x NAM in NV.

The supplied PRL must have the following format:

reserved (1 byte) will be filled by this function

prl_version (2 bytes)

size (2 bytes) PRL size in bits

valid (1 byte) boolean

roaming_list(variable length) packed IS683A format

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_SetGSM1xPRL
      (
      IGSM1xControl *instancePtr,
      byte *packedPRL
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
packedPRL	Pointer to a memory location that contains PRL to be written.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_GetGSM1xPRL()

Return to the List of functions

IGSM1xControl_SetGSM1xSIDNIDPairs()

Description:

This function sets the specified Home and Locked SID/NID pairs in NV.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_SetGSM1xSIDNIDPairs
      (
      IGSM1xControl *pInstance,
      uint16 HomeSIDNIDCnt,
      AEEGSM1xControl_SIDNIDPairType *HomeSIDNIDPairs,
      uint16 LockedSIDNIDCnt,
      AEEGSM1xControl_SIDNIDPairType *LockedSIDNIDPairs
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
HomeSIDNIDCnt	Number of SID/NID pairs located in the buffer pointed by HomeSIDNIDPairs.
HomeSIDNIDPairs	Pointer to a memory location that contains home SID/NID pairs.
LockedSIDNIDCnt	Number of SID/NID pairs located in the buffer pointed by LockedSIDNIDPairs.
LockedSIDNIDPairs	Pointer to a memory location that contains locked SID/NID pairs.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_GetGSM1xSIDNIDPairs()

Return to the List of functions

IGSM1xControl_SetPLMN()

Description:

This function writes the supplied PLMN information to EFplmnsel and/or EFfplmn fields in DFgsm on the currently available SIM or R-UIM card.

The entries having type AEEGSM1XCONTROL_HOME_PLMN are ignored.

The entries having type AEEGSM1XCONTROL_SEL_PLMN are written to EFplmnsel in the same order as they are present in the supplied array.

If there are more entries specified than can fit into EFplmnsel field,the extra entries are ignored. The entries having typeAEEGSM1XCONTROL_FORBIDDEN_PLMN are written to EFfplmn fieldin the same order as they are present in the supplied array.If there are more entries specified than can fit into EFfplmn field,the extra entries are ignored.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_SetPLMN
      (
      IGSM1xControl *instancePtr,
      uint16 PLMNEntriesCnt,
      AEEGSM1xControl_PLMNTripletType *PLMNBuf
      )
```

Parameters:

instancePtr	Pointer to the IGSM1xControl object.
PLMNEntriesCnt	Number of entries in the supplied array.
PLMNBuf	Pointer to the array containing the PLMN entries to be written.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_GetPLMN()

Return to the List of functions

IGSM1xControl_ValidatePRL()

Description:

This function validates the supplied packed PRL. In order to validate, the supplied PRL must have the same format IS683A or IS683C) as the phone software. The PRL format does not allow specification of the standard used.

Prototype:

```
AEEGSM1xControl_statusType IGSM1xControl_ValidatePRL
      (
      IGSM1xControl *pInstance,
      byte *packedPRL,
      boolean *isValid
      )
```

Parameters:

pInstance	Pointer to the IGSM1xControl object.
packedPRL	Pointer to a memory location that contains PRL to be validated.
isValid	Pointer to a memory location to receive the information whether the supplied PRL is valid or not. The contents of this location are undefined unless AEEGSM1XCONTROL_STATUS_SUCCESS is returned.

Return Value:

AEEGSM1xControl_statusType - the set of returned values are defined by constants whose name starts with the prefix AEEGSM1XCONTROL_STATUS_

Comments:

None

See Also:

IGSM1xControl_SetGSM1xPRL()

Return to the List of functions

IGSM1xSig Interface

Description:

IGSM1xSig interface enables GSM1x capability on the mobile device. It provides an interface to the GSM1x Signaling layer. It also provides a method to check the status of the GSM1x capability.

It provides the following services:

Sending GSM1x Signaling Msg to network

Notification upon receiving GSM1x Signaling Msg from Network

Checking the status of GSM1x.

The IGSM1xSig interface is obtained via the ISHELL_CreateInstance mechanism.

To send event notifications, IGSM1xSig uses a helper class IGSM1xSigNotifier. Methods for IGSM1xSigNotifier should not be called directly by BREW applets. Brew applet should specify the class id for IGSM1xSigNotifier and the NMASK in its MIF file.

Only one type of event notification is provided:

NMASK_GSM1xSIG_PROTOCOL_TYPE
This event is send whenever a GSM1x signaling message is received. While registering applet should specify the value indicating the protocol type applet is interested in. An applet will only receive signaling messages for the protocol types it has registered for.

A pointer to AEEGSM1xSig_SignalingMessageType struct is sent as dwParam member of the EVT_NOTIFY event.

In general if an applet is interested in messages for protocol type “x”, it should specify a NMASK of (NMASK_GSM1xSIG_PROTOCOL_TYPE | x).

Possible values for protocol type are specified in enum type AEEGSM1xSig_ProtocolTypes.

List of Header files to be included

The following header file is required:

AEEGSM1xSig.h

List of functions

Functions in this interface include:

IGSM1xSig_GetStatus()
IGSM1xSig_SendSignalingMessage()
IGSM1xSig_SendSignalingReject()

The remainder of this section provides details for each function.

IGSM1xSig_GetStatus()

Description:

This method is used to retrieve the current GSM1x status on the phone. This method lets the caller know if GSM1x capability is enabled or disabled on the mobile device.

Prototype:

```
AEEGSM1xSig_Status IGSM1xSig_GetStatus
      (
      IGSM1xSig * po
      )
```

Parameters:

po: Pointer to the IGSM1xSig object

Return Value:

GSM1xSIG_ACTIVE - GSM1x capability enabled on the mobile device.

GSM1xSIG_INACTIVE - GSM1x capability disabled on the mobile device.

Comments:

None.

See Also:

IGSM1xControl Interface

Return to the List of functions

IGSM1xSig_SendSignalingMessage()

Description:

This method is used to send a GSM1x signaling message (except for GSM1x Authentication Req/Rsp Message) to the network.

Prototype:

```
int IGSM1xSig_SendSignalingMessage
      (
      IGSM1xSig *po,
      AEEGSM1xSig_SignalingMessageType *pMsg
      )
```

Parameters:

po: Pointer to the IGSM1xSig object

pMsg: Pointer to the AEEGSM1xSig_SignalingMessageType struct containing the GSM1x signaling msg details

Return Value:

SUCCESS - GSM1x Signaling Message queued up.

EBADPARM - Value in pMsg is invalid (ex. attempting to send Auth msg)

EFAILED - General Failure in sending out the signaling message

EITEMBUSY - No more buffers left for sending messages.

EGSM1x_INACTIVE - Phone is not in IGSM1x mode

EBADCLASS: - IPhone is not initialized.

Comments:

This method doesn't guarantee deilvery of the message to the network, it only ensures that message was sent from the mobile device. It is up to the calling routines to use other mechanism (ex. explicit ack) to detemine if the network entity received the message

See Also:

AEEGSM1xSig_SignalingMessageType

Return to the List of functions

IGSM1xSig_SendSignalingReject()

Description:

This method is used to send a GSM1x signaling reject message to the network.

Prototype:

```
int IGSM1xSig_SendSignalingReject
      (
      IGSM1xSig *po,
      AEEGSM1xSig_RejectMessageType *pMsg
      )
```

Parameters:

po: Pointer to the IGSM1xSig object

pMsg: Pointer to the AEEGSM1xSig_RejectMessageType struct containing the GSM1x signaling reject msg

Return Value:

SUCCESS - GSM1x Reject Message queued up.

EBADPARM - Value in pMsg is invalid (ex. attempting to send Auth msg)

EFAILED - General Failure in sending out the signaling message

EITEMBUSY - No more buffers left for sending messages.

EGSM1x_INACTIVE - Phone is not in IGSM1x mode

EBADCLASS: - IPhone is not initialized.

Comments:

This method doesn't guarantee deilvery of reject messages to the network.

See Also:

None

Return to the List of functions

IGSMSMS

Description:

IGSMSMS is a simple interface to the GSM1x support layer in the device.

It provides the following services:

Sending GSM1x SMS messages (SMS_SUBMIT)

Extracting SMS text and TL data from GSM1x SMS messages

The IGSMSMS interface is obtained via the ISHELL_CreateInstance mechanism.

List of Header files to be included

The following header file is required:

AEEGSMSMS.h

List of functions

Functions in this interface include:

IGSMSMS_CreateDefaultMessage()
IGSMSMS_DecodeMessage()
IGSMSMS_DecodeUserData()
IGSMSMS_DeleteAllMessages()
IGSMSMS_DeleteMessage()
IGSMSMS_EncodeUserData()
IGSMSMS_GetMessage()
IGSMSMS_GetMessageStatus()
IGSMSMS_GetMemoryCapExceededFlag()
IGSMSMS_GetSCAddress()
IGSMSMS_GetStatusReport()
IGSMSMS_GetStoreSize()
IGSMSMS_GetTPMR()
IGSMSMS_IsInit()
IGSMSMS_MoveMessage()
IGSMSMS_SendMoreMemoryAvailable()
IGSMSMS_SendSMSDeliverReport()
IGSMSMS_SendSMSSubmit()
IGSMSMS_SetSCAddress()
IGSMSMS_SetMemoryCapExceededFlag()
IGSMSMS_SetMessageStatus()
IGSMSMS_SetTPMR()

IGSMSMS_StoreMessage()
IGSMSMS_StoreStatusReport()

The remainder of this section provides details for each function.

IGSMSMS_CreateDefaultMessage()

Description:

This function initializes a Mobile Originated (MO) message structure to the default values that are most commonly used.

The defaults for a GSMSMSSubmitType are:

SCAddr - Filled in with Default SC Address from SIM

RD - Reject Duplicates set to True

VP - Validity Period

SRR - Status Report Request set to True;

UDHI - User Data Header Indicator set to False

RP - Reply Path Parameter is not set (RP=False)

MR - Message Reference is 0 and is filled in when sent

DA - Destination Address is set to "0" and must be overwritten by the application

PID - Protocol ID is SME to SME by default

DCS - Data coding Scheme is set to GSM-7bit, class None

UDL - User Data Length is set to 0

UD - User Data is initialized to all zeros

The defaults for a GSMSMSDeliverReportType:

SCAddr - Filled in with Default SC Address from SIM

UDHI - User Data Header Indicator set to False

FCS_present - Failure Cause present is set to False (indicating no failure)

FCS - Failure Cause is set to 0 (reserved)

PID_present - Protocol ID is not present (PID_present=False)

PID - Protocol ID is SME to SME by default

DCS_present - Data Coding Scheme is not present (DCS_present=False)

DCS - Data coding Scheme is set to GSM-7bit, class None

UDL_present - User Data Length is not present (DCS_present=False)

UDL - User Data Length is set to 0

UD - User Data is initialized to all zeros

Prototype:

```
int IGSMSMS_CreateDefaultMessage
      (IGSMSMS *po,
      GSMSMSMsgType type,
      GSMSMSMsg *pGsmMsg);
```

Parameters:

po	Pointer to the IGSMSMS object
type	The type of message to create

pGsmMsg Pointer to the GSM SMS Msg structure to initialize.

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If the the paremeters were not valid: AEE_GSMSMS_EBADPARAM

If the message was not valid: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_DecodeMessage()

Description:

This function decodes a raw SMS message into an appropriate structure representation.

The UserData (UD) will not be decoded and can be subsequently:

```
AECHAR wszText[GSMSMS_MAX_UD_CHAR];
GSMSMSMsg DecodedMsg;
      // Decode Message
IGSMSMS_DecodeMessage(pMe, pNotifyMsg->rawMsg, sizeof(pNotifyMsg-
>rawMsg), &DecodedMsg);
      // Decode UserData
IGSMSMS_DecodeUserData(pMe, DecodedMsg, &wszText[0],
sizeof(wszText));
```

Prototype:

```
int IGSMSMS_DecodeMessage
      (
      IGSMSMS * po,
      const GSMSMSRawMsg *pRawMsg,
      const GSMSMSMsg *pMsg
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pRawMsg	Pointer to the raw GSM SMS message to decode
pMsg	Pointer to the GSMSMSMsg struct to fill with the decoded information

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the store could not be decoded: AEE_GSMSMS_EFAILED

If pRawMsg or pMsg are NULL: AEE_GSMSMS_EBADPARAM

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_DecodeUserData()

Description:

This function is used decode the UserData into UNICODE for display. If the UserData contains a user data header it will be skipped.

ONLY THE RAW TEXT IS RETURNED by this method. If the message is a concatinated or EMS message, it must be decoded directly by an external library or by the applet.

See the description of IGSMSMS_DecodeMessage for an example of how this is used.

If wstrlen is less than the decoded data length, the decoded data will be truncated and NULL terminated.

Prototype:

```
IGSMSMS_DecodeUserData
      (
      IGSMSMS * po,
      const GSMSMSMsg *pMsg,
      AECHAR *pwzStr,
      uint16 wstrlen
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pMsg	Pointer to the decoded SMS message
pwzStr	Pointer to the unicode string to fill with the decoded data
wstrlen	Size in bytes of the wstr unicode string

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the data could not be decoded: AEE_GSMSMS_EFAILED

If pMsg or pwzStr are NULL: AEE_GSMSMS_EBADPARAM

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_DeleteAllMessages()

Description:

This function deletes all entries of a specified type from the store (SIM, NVRAM or NVRAM Voivemail). It provides a way to delete only mobile originated (MO), mobile terminated (MT), or all messages.

Prototype:

```
int IGSMSMS_DeleteAllMessages
      (
      IGSMSMS * po,
      uint16 msgMask,
      GSMSMSStorageType deleteFrom
      )
```

Parameters:

po	Pointer to the IGSMSMS object
msgMask	Indicates the type of messages to delete. Any combination of the following GSMSMS_SIM_MO GSMSMS_SIM_MT GSMSMS_SIM_ALL
deleteFrom	Indicates whether to delete from the SIM or from NVRAM database.

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If deleteFrom is invalid: AEE_GSMSMS_EBADPARAM

If the messages could not be deleted: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_DeleteMessage()

Description:

This function deletes an entry from a specified slot on the store (SIM, NVRAM or NVRAM Voicemail).

Prototype:

```
void IGSMSMS_DeleteMessage
      (
      IGSMSMS * po,
      uint16 index,
      GSMSMSStorageType deleteFrom
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	Slot index of message to delete
deleteFrom	Indicates whether to delete from the SIM or from NVRAM database.

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If deleteFrom is invalid: AEE_GSMSMS_EBADPARAM

If the message could not be deleted: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_EncodeUserData()

Description:

This function is used to decode the UserData into UNICODE for display. If a UserData Header is included then address of the offset user data should be passed to this function. The udlen parameter must also be reduced accordingly. If the message will not fit in the space provided it will be truncated.

Prototype:

```
IGSMSMS_EncodeUserData
      (
      IGSMSMS * po,
      const AECHAR *pwszText,
      byte *pDest,
      byte destLen,
      GSMEncodingType encoding,
      byte *pEncodedLen
      )
```

Parameters:

po	[in]	Pointer to the IGSMSMS object
pwszText	[in]	Pointer to the unicode string to encode
pDest	[out]	Pointer to the part of the UD field to encode the data
destLen	[in]	Size in bytes of the UD field left for the encoded string
encoding	[in]	Encoding to use for text
pEncodedLen	[out]	Length of the encoded text in bytes

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If pUD is NULL, pwszText is NULL or encoding is invalid: AEE_GSMSMS_EBADPARAM

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetMessage()

Description:

This function retrieves an SMS entry from a specified slot on the store (SIM or NVRAM). The message is retrieved in raw format and can be decoded using IGSMSMS_DecodeMessage and IGSMSMS_DecodeUserData.

Prototype:

```
int IGSMSMS_GetMessage
      (
      IGSMSMS * po,
      uint16 index,
      GSMSMSRawMsg *pMsg,
      GSMSMSStorageType readFrom
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	Slot index of message to retrieve
pMsg	Pointer to struct to be filled in with the retrieved message
readFrom	Indicates whether to read from the SIM or from NVRAM database

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If pMsg is NULL, or readFrom is invalid: AEE_GSMSMS_EBADPARAM

If the messages could not be read: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetMessageStatus()

Description:

This function reads the status of a message in the specified store. The status will indicate whether the specified slot is free or if the message stored there is mobile terminated or mobile originated. If the message is mobile terminated it will also indicate whether the message has been read or not.

If the message is mobile originated, it will also indicate whether the message has been sent or not and if sent, whether there is a pending or received status report for the message.

Prototype:

```
int IGSMSMS_GetMesageStatus
      (
      IGSMSMS * po,
      uint16 index,
      GSMSMSStatusType *pStatus,
      GSMSMSStorageType readFrom
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	The index in the store to read from
pStatus	Pointer to the status to read
readFrom	Indicates whether to use the SIM or NVRAM database

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If readFrom is invalid: AEE_GSMSMS_EBADPARAM

If the entry could not be updated: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetMemoryCapExceededFlag()

Description:

This function retrieves the default Memory Capacity Exceeded Flag on the SIM. In accordance with the GSM TS 11.11 spec, pFlag is set to 0 if the flag is set, and set to 1 if the flag is not set. All other values are reserved.

Prototype:

```
void IGSMSMS_GetMemoryCapExceededFlag(IGSMSMS * po, uint8 *pFlag)
```

Parameters:

po: Pointer to the IGSMSMS object

pFlag: Pointer to variable to set to the value of the flag

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the flag could not be read: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetSCAddress()

Description:

This function retrieves the default GSM Service Center address from the SIM.

Prototype:

```
int IGSMSMS_GetSCAddress
      (
      IGSMSMS * po,
      GSMSMSAddress const *address
      )
```

Parameters:

po	Pointer to the IGSMSMS object
address	Pointer to the structure to fill in with the default SC address

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the SC address could not be retrieved: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetStatusReport()

Description:

This function retrieves an SMS StatusReport entry that corresponds to a SMS_Submit message in a specified slot on the store (SIM or NVRAM).

Prototype:

```
int IGSMSMS_GetStatusReport
      (
      IGSMSMS * po,
      uint16 index,
      GSMSMSRawMsg *pMsg,
      GSMSMSStorageType readFrom
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	Slot index of the SMS_Submit corresponds to the status report
pMsg	Pointer to the GSM SMS Msg to write the status report to
readFrom	Indicates whether to read from the SIM or from NVRAM database.

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If readFrom is invalid: AEE_GSMSMS_EBADPARAM

If the message could not be read: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetStoreSize()

Description:

This function returns the size of the store in slots.

Prototype:

```
int AEEGSMSMS_GetStoreSize
      (
      IGSMSMS * po,
      GSMSMSStorageType readFrom,
      uint16 *pCount
      )
```

Parameters:

po	Pointer to the IGSMSMS object
readFrom	Message store to query
pCount	Pointer to the variable to set to the number of slots in the store

Return Value:

If successful: AEE_GSMSMS_SUCCESS

Otherwise: AEE_GSMSMS_EBADPARAM

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_GetTPMR()

Description:

This function retrieves the last used TP-MR from EF-SMSS on the SIM.

Prototype:

```
int IGSMSMS_GetTPMR(IGSMSMS * po, uint8 *pTPMR)
```

Parameters:

po: Pointer to the IGSMSMS object

pTPMR: Pointer to variable to set to the value of the TP-MR

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the flag could not be read: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_IsInit()

Description:

This function indicates to the user whether the IGSMSMS interface has finished internal initialization. If the interface is not initialized, the applet can register for NMASK_GSMSMS_INIT in IGSMSMSNotifier to be notified when the initialization is complete.

Prototype:

```
boolean IGSMSMS_IsInit(IGSMSMS *po)
```

Parameters:

po: Pointer to the IGSMSMS object

Return Value:

TRUE if initialized, otherwise FALSE

Comments:

None

See Also:

None

See Also:

None

Return to the List of functions

IGSMSMS_MoveMessage()

Description:

This function moves a GSM SMS message from one store to another. It will also allow the the user to move from one slot to another on the same store.

Prototype:

```
int IGSMSMS_MoveMessage
      (
      IGSMSMS *po,
      GSMSMSStorageType moveFrom,
      uint16 fromIndex,
      GSMSMSStorageType moveTo,
      uint16 *pToIndex
      )
```

Parameters:

po	Pointer to the IGSMSMS object
moveFrom	Store to move the message from
fromIndex	Index of the message to move
moveTo	Store to move the message to
pToIndex	Pointer to the index of the stored message, if set to GSMSMS_INDEX_ANY an empty slot will be selected and returned

Return Value:

AEE_GSMSMS_SUCCESS,If successful.

AEE_GSMSMS_ESTORE_FULL, If the store was full.

AEE_GSMSMS_EFAILED, If the message could not be stored.

AEE_GSMSMS_ENOSERVICE, if the device is not in GSM1x mod.

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SendMoreMemoryAvailable()

Description:

This function allows an application to send a GSM RP Layer SMMA message. This function is called in accordance with the GSM 23.040 spec. If the message times out, pReport->sendResult is set to AEE_GSMSMS_ETIMEDOUT. All other pReport fields are invalid. If the message is successfully sent, pReport->sendResult is set to AEE_GSMSMS_SUCCESS.

Prototype:

```
int IGSMSMS_MoreMemoryAvailable
      (
      IGSMSMS *po,
      AEECallback *pCb,
      GSMSMSSMMAReport *pReport
      )
```

Parameters:

po: Pointer to the IGSMSMS object

pCb: Pointer to AEECallback to call upon response message arrival

pReport: Pointer to report structure to fill with response

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If the message could not be sent: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SendSMSDeliverReport()

Description:

This function allows an application to send a GSM SMS_DELIVER_REPORT message. This should be sent in response to receiving an SMS_DELIVER message. The user would first fill out a GSMSMSDeliverReportType structure either manually or by calling IGSMSMS_CreateDefaultMessage to fill in the structure and only modifying the parameters that are different from the default. For a simple SMS_DELIVER_REPORT message, this would typically be the

`UserData (GSMSMSSubmitType->UD)`

and the destination address

`(GSMSMSSubmitType->DA).`

Prototype:

```
int IGSMSMS_SendSMSDeliverReport
      (
      IGSMSMS *po,
      const GSMSMSDeliverReportType *pDeliverReport
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pDeliverReport	Pointer to the SMS Deliver Report message to send

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If the the paremeters were not valid: AEE_GSMSMS_EBADPARAM

If the message was not valid: AEE_GSMSMS_EFAILED

Comments:

None

Return to the List of functions

IGSMSMS_SendSMSSubmit()

Description:

This function allows an application to send a GSM SMS_SUBMIT message. This can be used to send a SMS/EMS message and is also used for SIM toolkit. The user would first fill out a GSMSMSSubmitType structure either manually or by calling IGSMSMS_CreateDefaultMessage to fill in the structure and only modifying the parameters that are different from the default. For a simple SMS_SUBMIT message, this would typically be the Destination Address (DA) and the UserData (UD,UDL).

When the SMS_SubmitReport message is received, the pCB function will be called and the status can be retrieved from the GSMSMSSendReport structure. If a status report message was requested, the SMS_StatusReport message will be delivered via the callback specified in the OnMTMessage registration.

Prototype:

```
int IGSMSMS_SendSMSSubmit
      (
      IGSMSMS *po,
      const GSMSMSSubmitType * pMsg,
      AEECallback *pCb,
      GSMSMSSendReport *pReport
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pMsg	Pointer to the SMS Submit message to send
pCb	This callback will be invoked by AEE when the SMS_SUBMIT_REPORT is received.
pReport	Pointer to structure to be filled in reporting the submit status

Return Value:

AEE_GSMSMS_EBUSY if a transaction is pending

AEE_GSMSMS_EBADPARAM if pMsg is NULL or pCb is NULL

AEE_GSMSMS_EENCODE if the message could not be encoded

AEE_GSMSMS_SUCCESS if successful

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SetSCAddress()

Description:

This function updates the default SC address on the SIM.

Prototype:

```
void IGSMSMS_SetSCAddress
      (
      IGSMSMS * po,
      const GSMSMSAddress *pAddress
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pAddress	Pointer to the structure to containing the new SC address

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the SC address could not be set: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SetMemoryCapExceededFlag()

Description:

This function updates the Memory Capacity Exceeded Flag on the SIM. In accordance with the GSM TS 11.11 spec, flag should be set to 0 if the MemCapExceded flag is set, and set to 1 if the it is not set. All other values are reserved.

Prototype:

```
void IGSMSMS_SetMemoryCapExceededFlag(IGSMSMS * po, uint8 flag)
```

Parameters:

po: Pointer to the IGSMSMS object

pFlag: New setting for the MemCapExceded flag

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If flag is invaliid: AEE_GSMSMS_EBADPARAM

If the flag could not be set: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SetMessageStatus()

Description:

This function updates the status for a message in the specified store.

It can be used to mark a :

MT message as read or unread

MO message as sent or not sent

MO message as having a pending status report

MO message as having a received status report

MO message as having a stored status report

Prototype:

```
int IGSMSMS_SetMesageStatus
      (
      IGSMSMS * po,
      uint16 index,
      GSMSMSStatusType status,
      GSMSMSStorageType writeTo
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	Slot index of message to mark
status	The status to write
writeTo	Indicates whether to use the SIM or NVRAM database

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If writeTo is invalid: AEE_GSMSMS_EBADPARAM

If the entry could not be updated: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_SetTPMR()

Description:

This function sets the last used TP-MR from EF-SMSS on the SIM.

Prototype:

```
int IGSMSMS_SetTPMR(IGSMSMS * po, uint8 TPMR)
```

Parameters:

po: Pointer to the IGSMSMS object

TPMR: New value for the TP-MR in EF-SMSS on the SIM

Return Value:

If successful: AEE_GSMSMS_SUCCESS

If the flag could not be read: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_StoreMessage()

Description:

This function stores a GSM SMS message on the specified store in the specified index.

If *pIndex is set to GSMSMS_STORE_SIM before the call, a free slot will be selected and the value of *pIndex will be updated. Otherwise, the value of *pIndex will be used.

If the storeTo parameter is set to GSMSMS_STORE_NVRAM the message will be stored in the first free slot in the NVRAM mailbox.

If the storeTo parameter is set to storeTo GSMSMS_STORE_SIM, the message will be stored in the first free slot in the SIM.

If the storeTo parameter is set to GSMSMS_STORE_NVRAM_VM, the currently stored voicemail message is replaced with the specified voicemail message.

Prototype:

```
int IGSMSMS_StoreMessage
      (
      IGSMSMS *po,
      const GSMSMSMsg * pMsg,
      GSMSMSStorageType storeTo,
      uint16 *pIndex
      )
```

Parameters:

po	Pointer to the IGSMSMS object
pMsg	Pointer to the GSM SMS Msg to store
storeTo	Indicates whether to store to the SIM or to NVRAM database
pIndex	Pointer to variable containing the index to to store the message at. If the index is set to GSMSMS_INDEX_ANY an index will be selected and the value of *pIndex will be updated.

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If the store was full: AEE_GSMSMS_ESTORE_FULL

If the message could not be stored: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

IGSMSMS_StoreStatusReport()

Description:

This function stores a GSM SMS_StatusReport message that corresponds to a SMS_Submit message stored at the specified slot on the SIM or in NVRAM.

In GSM, the SMSR directory on the SIM contains the StatusReport. This entry contains an index into the SMS directory for the corresponding SMS Submit message. When the entry in the SMS directory is deleted, the corresponding entry in the SMSR is no longer valid. Therefore, the status report must be stored on the same store as the SMS_Submit message. If the SMS_Submit message is ever moved to another store,

the SMS_StatusReport must be moved at the same time or it will be lost. If the status report is stored in NVRAM only the portion of the message that would be saved to the SMSR is preserved.

Prototype:

```
int IGSMSMS_StoreStatusReport
      (
      IGSMSMS *po,
      uint16 index,
      const GSMSMSMsg * pMsg,
      GSMSMSStorageType storeTo
      )
```

Parameters:

po	Pointer to the IGSMSMS object
index	Pointer to slot index of the corresponding SMS_Submit message
pMsg	Pointer to the GSM SMS Msg to store
storeTo	Indicates whether the SMS_Submit message is on the SIM or in NVRAM database

Return Value:

If successful: AEE_GSMSMS_SUCCESS.

If pMsg is NULL or storeTo is invalid: AEE_GSMSMS_EBADPARAM

If the message could not be stored: AEE_GSMSMS_EFAILED

Comments:

None

See Also:

None

Return to the List of functions

ILogger Interface

BREW provides a standardized and extensible data logging interface, which allows a BREW application developer to log data using a number of different transport mechanisms.

Below are the primary logging transport implementations. A BREW application developer selects one by creating an ILogger instance with one of the following class IDs:

Class ID	Description
AEECLSID_LOGGER_FILE	Sends log items to a file.
AEECLSID_LOGGER_WIN	Sends log items to the Emulator output window.

Each implementation is responsible for handling and writing to a specific transport but the data being sent is transport independent.

The header file AEELoggerTypes.h provides definitions for the logging data types common to both BREW's ILogger interface and the PC side log parser in a client/server type of architecture.

The file implementation outputs data to the output file in the following BREW packet format:

The Windows implementation of the ILogger interface writes all outgoing logs to the BREW output window using the following format:

bkt:xx typ:xx cID:xx iID:xx FILENAME LINENUMBER MESSAGE ARGS

in which

bkt	Log bucket
typ	Log type
cID	ClassID of the currently running BREW application
iID	User-defined instance ID
FILENAME	Optional file name where log was sent
LINENUMBER	Optional line number where log was sent
MESSAGE	User defined text message
ARGS	Optional arguments using OEMLogger_PutMsg()

When compiling a release version of a BREW application, the constant AEE_LOG_DISABLE may be defined, which, using the preprocessor, removes all OEMLogger interface logging functions, except the instance creation and getting and setting parameters processes. This constant must be defined before a new BREW application includes AEELogger.h.

The contents of log data is determined by the type element of the BREW Log header. Three standard log types are predefined by BREW, but the BREW application developer can also define as many custom log types as required. The three standard BREW-defined log types are as follows:

Type	Description	Data contains
AEE_LOG_TYPE_TEXT	ASCII text message	If you use this log type, the data contains nSize bytes of ASCII text.
AEE_LOG_TYPE_BIN_MSG	AEELogTypeBinMsg	If you use this log type, the data contains one AEE LogTypeBinMsg structure.
AEE_LOG_TYPE_BIN_BLK	Block of arbitrary binary data	If you use this log type, the data contains nSize bytes of arbitrary binary data.

Log items are sent and filtered in one of 255 distinct, general purpose buckets. These log buckets are filtered by the developer at run time using ILOGGER_SetParam() and ILOGGER_GetParam() or on the PC side, using a post processor.

The structure AEELogTypeBinMsg contains the following elements:

Element	Description
Header	b7,b6 – bits reserved b5,b4 – number of args b3 bit – file name present b2,b1,b0 – message level
Line	Line number in application code where this log item was sent
args[MAX_LOG_TYPE_BIN_MSG_ARGS]	Contains zero or more 32 bit integer values
pszMsg[MAX_LOG_TYPE_BIN_MSG_TEXT_SIZE]	pszMsg contains two consecutive NULL terminated strings: the first is the file name where the log message was sent and the second is an ASCII text message

List of Header files to be included

The following header files are required:

AEELogger.h

AEELoggerTypes.h

List of functions

Functions in this interface include:

ILOGGER_AddRef()
ILOGGER_GetParam()
ILOGGER_Printf()
ILOGGER_PutItem()
ILOGGER_PutMsg()
ILOGGER_Release()
ILOGGER_SetParam()

The remainder of this section provides details for each function.

ILOGGER_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

ILOGGER_Release()

Return to the List of functions

ILOGGER_GetParam()

Description:

This function is called to get the configuration of the ILOGGER interface. Supported Parameters depends on the current implementations support. See AEELogParamType.

Prototype:

```
int ILOGGER_GetParam
      (
      ILogger *pILogger,
      AEELogParamType pType,
      void* pParam
      )
```

Parameters:

pILogger	Pointer to the ILogger Interface object
pType	Parameter to modify
pParam	Pointer to be filled with settings parameter

Return Value:

SUCCESS Parameter handled successfully

EBADPARM NULL parameter pointer

EUNSUPPORTED Parameter type or option not supported

EFAILED General failure, option not handled successfully

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

ILOGGER_Printf()

Description:

This function is called to send a formatted ASCII text message.

ILOGGER_Printf() is a MACRO that allows variable arguments. It must be called as follows:

```
ILOGGER_Printf( pMe->m_pILogger,
( pMe->m_pILogger,
USER_BUCKET1,
__FILE__,
(uint16)__LINE__,
"msg",
args ) );
```

Notice that the second argument is actually multiple arguments in parentheses, and args can be multiple comma separated values

Prototype:

```
int ILOGGER_Printf
      (
      ILogger *pILogger,
      AEELogBucketType bucket,
      const char *pszFileName,
      uint16 nLineNum,
      const char *pszFormat,
      ... );
```

Parameters:

pILogger,	Pointer to the ILOGGER object
bucket	Bucket to place item
pszFileName	Name of file calling this function
nLineNum	Line number in file where it was called
pszFormat	ASCII text string similar to a printf format string
...	Format string arguments

Return Value:

SUCCESS Log sent successfully

EBADPARM Invalid pointer to pszFormat

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocate required memory

EFAILED Log not sent

-- The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

Return to the List of functions

ILOGGER_PutMsg()

Description:

This function is called to send a predefined binary message and allows fast logging due to the limited formatting required and the fixed size of the outgoing log message. The outgoing binary message's data is of type structure AEELogBinMsgType, which is defined in AEELoggerTypes.h.

Prototype:

```
int ILOGGER_PutMsg
      (
      ILogger *pILogger,
      AEELogBucketType bucket,
      const char *pszFileName,
      uint16 nLineNum,
      const char *pszMsg,
      uint8 nNumArgs,
      uint32 args[ MAX_LOG_TYPE_BIN_MSG_ARGS ]
      )
```

Parameters:

pILogger,	Pointer to the ILOGGER object
bucket	Bucket to place item
pszFileName	ASCII NULL terminated name of file calling this function
nLineNum	Line number in file where it was called
pszMsg	ASCII NULL terminated text message
nNumArgs	length of the args array
args	array containing uint32 arguments

Return Value:

SUCCESS Log sent successfully

EBADPARM Invalid pointer to pszMsg or nNumArgs too large

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocate required memory

EFAILED Log not sent

-- The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBinMsgType

AEELogBucketType

Return to the List of functions

ILOGGER_PutItem()

Description:

This function is called to send a prioritized user defined binary message. Here are the steps to define a user log item type:

1. Choose a user item number and define a meaningful name to it.

Example:

```
#define MY_APPS_LOG_ITEM_TYPE AEE_LOG_TYPE_USER_1
```

2. Define a structure that corresponds to you're new type,

Example:

```
typedef struct{
uint8 foo1;
uint32 foo2;
uint8 fooString[ STRING_SIZE ];
} myAppsItem;
```

3. Enable the PC software that will be reading the logging output to recognize the log item type AEE_LOG_TYPE_USER_1 (which in this case is MY_APPS_LOT_ITEM_TYPE)

4. Call ILOGGER_PutItem() with MY_APPS_LOG_ITEM_TYPE, a pointer to an instance of myAppsItem, and the size of myAppsItem.

Prototype:

```
int ILOGGER_PutItem
      (
      ILogger *pILogger,
      AEELogBucketType bucket,
      AEELogItemType type,
      uint16 nSize,
      uint8 *pItem
      )
```

Parameters:

pILogger,	Pointer to the ILOGGER object
bucket	Bucket to place item
type	User defined item type
nSize	Size of type in bytes
pItem	Pointer to instance of type

Return Value:

SUCCESS Log sent successfully

EBADPARM Invalid pointer to pItem or size equal to zero

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocate required memory

EFAILED Log not sent

-- The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

AEELogItemType

Return to the List of functions

ILOGGER_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

ILOGGER_AddRef()

Return to the List of functions

ILOGGER_SetParam()

Description:

This function is called to configure the performance and behavior of the ILOGGER interface. Supported parameters depend on the current implementation's support, see AEELogParamType for more information.

Prototype:

```
int ILOGGER_SetParam
      (
      ILogger *pILogger,
      AEELogParamType pType,
      uint32 param,
      void* pParam
      )
```

Parameters:

pILogger	Pointer to the ILOGGER object
pType	Parameter to modify
param	New settings parameter
pParam	Pointer to new settings parameter

Return Value:

SUCCESS Parameter handled successfully

EUNSUPPORTED Parameter type or option not supported

EFAILED General failure, option not handled successfully

EBADPARM NULL parameter pointer

-- The following log codes only apply to file logging

EFILENOEXISTS Output log file is closed

EFILEEXISTS Output log file is open

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

IPosDet Interface

This interface provides services for position determination using sector information or GPS information. In order to use the sector-based position determination methods such as IPOSDET_GetSectorInfo(), the sector information privileges are required. Similarly, for GPS based position determination methods such as IPOSDET_SetGPSConfig(), IPOSDET_GetGPSConfig(), and IPOSDET_GetGPSInfo(), position determination privileges are required.

IPOSDET_GetGPSInfo() is an asynchronous method which use AEECallback. Care must be taken to ensure that the callbacks and information structures passed to these methods by reference remain in scope till the callback returns. Also, if multiple requests for sector information or GPS info. are made without waiting for the callbacks to return, the behavior of the interface will be unpredictable.

BREW SDK users can set the GPS emulation in the Tools->GPS Emulation menu to use a pre-recorded NMEA file as GPS input, or connect an NMEA-output capable GPS device. An offline utility called NMEALogger.exe can be used to record an NMEA file from data coming from a GPS device connected to the serial port of the desktop/laptop. This NMEA file can be used later as GPS input. See *SDK User's Guide* and *SDK Utilities Guide* for details.

List of Header files to be included

The following header file is required:

AEEPosDet.h

List of functions

Functions in this interface include:

IPOSDET_AddRef()
IPOSDET_GetGPSConfig()
IPOSDET_GetGPSInfo()
IPOSDET_GetOrientation()
IPOSDET_GetSectorInfo()
IPOSDET_QueryInterface()
IPOSDET_Release()
IPOSDET_SetGPSConfig()

The remainder of this section provides details for each function.

IPOSDET_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IPOSDET_Release()

Return to the List of functions

IPOSDET_GetGPSConfig()

Description:

This function gets the current GPS configuration of the GPS engine.

Prototype:

```
int IPOSDET_GetGPSConfig(IPosDet *pIPosDet, AEEGPSConfig *pConfig);
```

Parameters:

pIPosDet	[in]	Pointer to the IPosDet Interface object.
pConfig	[out]	Pointer to GPS configuration. See AEEGPSConfig for details.

Return Value:

SUCCESS, if the function succeeded

Error codes, if otherwise.

EPRIVLEVEL, if the caller does not have PL_POS_LOCATION privilege levels to invoke this function

EBADPARM, if **pConfig** is NULL

EUNSUPPORTED, if this function is not supported.

Comments:

Unless this function is called, the GPS engine is configured with default settings on the first call to IPOSDET_GetGPSInfo(). Only the position determination requests following a call to IPOSDET_SetGPSConfig() will use the new configurations. IPOSDET_GetGPSInfo() requests pending response callbacks may not be affected by this method.

See Also:

AEEGPSConfig

IPOSDET_SetGPSConfig()

Return to the List of functions

IPOSDET_GetGPSInfo()

Description:

This function returns information for GPS based position location. It returns latitude, longitude, altitude information, as well as vector information such as horizontal and vertical velocity, heading, and the uncertainty of the horizontal information. This is an asynchronous call, and the callback specified by pcb is called on completion.

Prototype:

```
int IPOSDET_GetGPSInfo
      (
      IPosDet *pIPosDet,
      AEEGPSReq req,
      AEEGPSAccuracy accuracy,
      AEEGPSInfo *pGPSInfo,
      AEECallback *pcb,
      )
```

Parameters:

pIPosDet	[in]	Pointer to the IPosDet Interface object.
req	[in]	Request type: AEEGPS_GETINFO_LOCATION, AEEGPS_GETINFO_VELOCITY, AEEGPS_GETINFO_ALTITUDE. The flags can be combined to get more than one type of information.
accuracy	[in]	Selected level of accuracy for this request.
pGPSInfo	[out]	On input, this must be a valid pointer to the AEEGPSInfo structure. On callback, the members of this struct contain GPS information. The caller must ensure that this structure is valid till the callback specified by pcb gets called.
pcb	[in]	Callback function which gets called on completion of position determination.

Return Value:

SUCCESS, if the function succeeded

EPRIVLEVEL, if the caller does not have sufficient privilege levels (PL_POS_LOCATION) to invoke this function

EBADPARM, if **pGPSInfo** or pcb is NULL

EUNSUPPORTED, if this function is not supported.

EFAILED, general failure

Comments:

None

See Also:

AEEGPSInfo

Return to the List of functions

IPOSDET_GetOrientation()

Description:

This function returns device's orientation in the horizontal plane. This is an asynchronous call, and the callback specified by pcb is called on completion.

Prototype:

```
int IPOSDET_GetOrientation
      (
      IPosDet *pif,
      AEEOrientationInfo *pOrInfo,
      AEECallback *pcb,
      );
```

Parameters:

pif	[in]	The interface pointer.
pOrInfo	[out]	On input, this must be a valid ptr to the AEEOrientationInfo structure with the member wSize indicating the space available in bytes. On callback, the members of this struct contain Orientation information. The caller must ensure that this structure is valid till the callback specified by pcb gets called.
pcb	[in]	Callback function which gets called on completion of position determination.

Return Value:

SUCCESS: if the function succeeded

EPRIVLEVEL: if the caller does not have sufficient privilege levels

(PL_POS_LOCATION) to invoke this function

EBADPARM: if pGPSInfo or pcb is NULL

EUNSUPPORTED: if this function is not supported.

EFAILED: general failure

Comments:

None

See Also:

AEEOrientationInfo

Return to the List of functions

IPOSDET_GetSectorInfo()

Description:

This function returns information for sector-based position location, such as the SystemID, NetworkID, BaseStationID, BaseStationClass, and best Pilot. To invoke this function, the caller (application) must have the PL_SECTORINFO privilege level. Without this privilege level, the function fails.

Prototype:

```
int IPOSDET_GetSectorInfo
      (
      IPosDet * pIPosDet,
      AEESectorInfo * pSecInfo
      )
```

Parameters:

pIPosDet	[in]	Pointer to the IPosDet Interface pointer.
pSecInfo	[out]	This must be a value pointer to the AEESectorInfo structure. On return, the members of this struct contain sector information.

Return Value:

AEE_SUCCESS, if the function succeeded.

Error codes, if otherwise.

EPRIVLEVEL, if the caller does not have sufficient privilege levels (PL_SECTORINFO) to invoke this function.

EUNSUPPORTED, if this function is not supported.

EFAILED, general failure.

Comments:

None

See Also:

AEESectorInfo

Return to the List of functions

IPOSDET_QueryInterface()

Description:

This function asks an object for another API contract from the object in question.

Prototype:

```
int IPOSDET_QueryInterface
      (
      IPosDet * pIPosDet,
      AEECLSID idReq,
      void * * ppo
      )
```

Parameters:

pIPosDet	[in]	Pointer to the IPosDet Interface object.
idReq	[in]	Requested ClassID exposed by the object.
ppo	[out]	Returned object. Filled by this function.

Return Value:

SUCCESS, interface found.

Error codes, if otherwise.

ENOMEMORY, insufficient memory.

ECLASSNOTSUPPORT, requested interface is unsupported.

Comments:

The pointer in ***ppo** is set to the new interface (with **refcount** positive), or NULL if the ClassID is not supported by the object.

See Also:

None

Return to the List of functions

IPOSDET_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IPOSDET_AddRef()

Return to the List of functions

IPOSDET_SetGPSConfig()

Description:

This function sets the GPS configuration to be used by the GPS engine.

Prototype:

```
int IPOSDET_SetGPSConfig(IPosDet *pIPosDet, AEEGPSConfig *pConfig);
```

Parameters:

pIPosDet	Pointer to the IPosDet Interface object.
pConfig	Pointer to GPS configuration. See AEEGPSConfig for details.

Return Value:

SUCCESS, if the function succeeded

Error codes, if otherwise.

EPRIVLEVEL, if the caller does not have sufficient privilege levels (PL_POS_LOCATION) to invoke this function

EBADPARM, if **pConfig** is NULL

EUNSUPPORTED, if this function is not supported.

Comments:

Unless this function is called, the GPS engine is configured with default settings on the first call to IPOSDET_GetGPSInfo(). Only the position determination requests following a call to IPOSDET_SetGPSConfig() will use the new configurations. IPOSDET_GetGPSInfo() requests pending response callbacks may not be affected by this method.

See Also:

AEEGPSConfig

IPOSDET_GetGPSConfig()

Return to the List of functions

IRingerMgr Interface

The IRingerMgr interface provides the BREW interface with the abilityto manage ringers on the device. This interface is obtained by calling ISHELL_CreateInstance() with AEECLSID_RINGER. The class allows the caller to:

- Create a ringer
- Obtain a list of created ringers
- Remove a ringer
- Play a ringer
- Obtain a list of supported ringer formats
- Obtain a list of ringer categories
- Set a category's ringer

List of Header files to be included

The following header file is required:

AEERinger.h

List of functions

Functions in this interface include:

IRINGERMGR_AddRef()
IRINGERMGR_Create()
IRINGERMGR_EnumCategoryInit()
IRINGERMGR_EnumNextCategory()
IRINGERMGR_EnumNextRinger()
IRINGERMGR_EnumRingerInit()
IRINGERMGR_GetFormats()
IRINGERMGR_GetNumberFormats()
IRINGERMGR_GetRingerID()
IRINGERMGR_GetRingerInfo()
IRINGERMGR_RegisterNotify()
IRINGERMGR_Remove()
IRINGERMGR_SetRinger()
IRINGERMGR_Stop()
IRINGERMGR_Play()
IRINGERMGR_PlayEx()
IRINGERMGR_PlayFile()
IRINGERMGR_PlayStream()
IRINGERMGR_RegisterNotify()
IRINGERMGR_Release()
IRINGERMGR_Remove()
IRINGERMGR_SetRinger()
IRINGERMGR_Stop()

The remainder of this section provides details for each function.

IRINGERMGR_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

IRINGERMGR_Release()

Return to the List of functions

IRINGERMGR_Create()

Description:

This function creates a new ringer.

Prototype:

```
int IRINGERMGR_Create
      (
      IRingerMgr * pIRingerMgr,
      const AECHAR * pszName,
      AEESoundPlayerFile format,
      IAStream * ps
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pszName	Name of the ringer.
format	Ringer format.
ps	Stream to ringer data.

Return Value:

SUCCESS, ringer is being created.

EFAILED, Unable to create ringer

Comments:

None

See Also:

AEESoundPlayerFile

Return to the List of functions

IRINGERMGR_EnumCategoryInit()

Description:

This function initializes the enumeration context for category enumeration.

Prototype:

```
int IRINGERMGR_EnumCategoryInit(IRingerMgr * pIRingerMgr)
```

Parameters:

pIRingerMgr Pointer to the IRingerMgr Interface object.

Return Value:

SUCCESS, enumeration initialized.

EFAILED, Unable to initialize enumeration

Comments:

There is one iteration for this function. Always call this before calling IRINGERMGR_EnumNextCategory().

See Also:

IRINGERMGR_EnumNextCategory()

Return to the List of functions

IRINGERMGR_EnumNextCategory()

Description:

This function enumerates the next ringer category.

Prototype:

```
boolean IRINGERMGR_EnumNextCategory
      (
      IRingerMgr * pIRingerMgr,
      AEERingerCat * pi
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pi	Pointer to the ringer category information to fill.

Return Value:

TRUE, if successful.

FALSE, if function fails or there are no more categories to enumerate.

Comments:

Always call IRINGERMGR_EnumCategoryInit() before calling IRINGERMGR_EnumNextCategory().

See Also:

AEERingerCat

IRINGERMGR_EnumCategoryInit()

Return to the List of functions

IRINGERMGR_EnumNextRinger()

Description:

This function enumerates the next ringer.

Prototype:

```
boolean IRINGERMGR_EnumNextRinger
      (
      IRingerMgr * pIRingerMgr,
      AEERingerInfo * pi
      )
```

Parameters:

pIRingerMgr	[in]	Pointer to the IRingerMgr Interface object.
pi	[out]	Pointer to the ringer information to fill.

Return Value:

TRUE, if successful.

FALSE, Failed to enumerate next ringer

Comments:

Always call IRINGERMGR_EnumRingerInit() before calling this.

See Also:

AEERingerInfo

IRINGERMGR_EnumRingerInit()

Return to the List of functions

IRINGERMGR_EnumRingerInit()

Description:

This function initializes enumeration of the list of ringers.

Prototype:

```
int IRINGERMGR_EnumRingerInit(IRingerMgr * pIRingerMgr)
```

Parameters:

pIRingerMgr Pointer to the IRingerMgr Interface object.

Return Value:

SUCCESS, enumeration initialized.

EFAILED, Unable to initialize enumeration.

Other implementation-specific error codes

Comments:

There is one iteration for this function. Always call this before calling IRINGERMGR_EnumNextRinger().

See Also:

IRINGERMGR_EnumNextRinger()

Return to the List of functions

IRINGERMGR_GetFormats()

Description:

This function fills a list of the supported ringer formats.

Prototype:

```
int IRINGERMGR_GetFormats
      (
      IRingerMgr * pIRingerMgr,
      AEESoundPlayerFile * pwFormats,
      int nCount
      )
```

Parameters:

pIRingerMgr	[in]	Pointer to the IRingerMgr Interface object.
pwFormats	[out]	Pointer to a list of formats of size **nCount** * sizeof(AEESoundPlayerFile).
nCount	[in]	Number of format entries to fill.

Return Value:

SUCCESS, buffer filled with ringer format entries.

Other error codes.

EFAILED, Size of return buffer is invalid.

Comments:

None

See Also:

AEESoundPlayerFile

IRINGERMGR_GetNumberFormats()

Return to the List of functions

IRINGERMGR_GetNumberFormats()

Description:

This function retrieves the number of ringer formats supported on the device.

Prototype:

```
int IRINGERMGR_GetNumberFormats(IRingerMgr * pIRingerMgr)
```

Parameters:

pIRingerMgr Pointer to the IRingerMgr Interface object.

Return Value:

Number of ringer formats supported.

Comments:

None

See Also:

IRINGERMGR_GetFormats()

Return to the List of functions

IRINGERMGR_GetRingerID()

Description:

This function returns the ringer ID for a ringer given the file name.

Prototype:

```
AEERingerID IRINGERMGR_GetRingerID
      (
      IRingerMgr * pIRingerMgr,
      const char * pszFile
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pszFile	Root file name of ringer.

Return Value:

AEERingerID of the ringer specified.

AEE_RINGER_ID_NONE, if the function fails.

Comments:

None

See Also:

AEERingerID

AEERingerInfo

Return to the List of functions

IRINGERMGR_GetRingerInfo()

Description:

This function retrieves information about the specified ringer.

Prototype:

```
int IRINGERMGR_GetNumberFormats
      (
      IRingerMgr * pIRingerMgr,
      AEERingerID id,
      AEERingerInfo * pi
      )
```

Parameters:

pIRingerMgr	[in]	Pointer to the IRingerMgr Interface object.
id	[in]	Ringer ID.
pi	[out]	Pointer to the ringer info structure to fill.

Return Value:

SUCCESS, ringer information valid.

Other error codes.

EFAILED, invalid ringer ID, or pi is NULL.

Comments:

None

See Also:

AEERingerID

AEERingerInfo

Return to the List of functions

IRINGERMGR_Play()

Description:

This function plays an installed ringer.

Prototype:

```
int IRINGERMGR_Play
      (
      IRingerMgr * pIRingerMgr,
      AEERingerID id,
      uint32 dwPause
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
id	Ringer ID to play.
dwPause	Time to pause between replays; 0 (zero) if single play.

Return Value:

SUCCESS, ringer begins playing.

EFAILED, invalid ringer.

Other implementation-specific error codes.

Comments:

None

See Also:

AEERingerID

IRINGERMGR_Stop()

Return to the List of functions

IRINGERMGR_PlayEx()

Description:

This function plays the specified ringer using the following items, in order, as the source:

- An installed ringer ID
- An input file name
- An input IAStream object

The action allows the caller to test ringers before placing them in the ringer directory.

Prototype:

```
int IRINGERMGR_PlayEx
      (
      IRingerMgr * pIRingerMgr,
      AEERingerID id,
      const char * pszFile,
      IAStream * pStream,
      uint32 dwPause
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
id	Ringer ID to play.
pszFile	Input file name.
pStream	Input stream.
dwPause	Time to pause between replays; 0 (zero) if single play.

Return Value:

SUCCESS, ringer begins playing.

EFAILED, Invalid ID, input file name, stream, or unable to play ringer.

Comments:

None

See Also:

AEERingerID

IRINGERMGR_Stop()

Return to the List of functions

IRINGERMGR_PlayFile()

Description:

This function plays a ringer when given an input file name. This action allows the caller to test ringers before placing them in the ringer directory.

Prototype:

```
int IRINGERMGR_PlayFile
      (
      IRingerMgr * pIRingerMgr,
      const char * pszFile,
      uint32 dwPause
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pszFile	Input file name.
dwPause	Time to pause between replays (0 if single play).

Return Value:

SUCCESS, ringer begins playing.

EFAILED, invalid input file name or unable to play ringer.

Other implementation-specific error codes.

Comments:

None

See Also:

IRINGERMGR_Stop()

Return to the List of functions

IRINGERMGR_PlayStream()

Description:

This function plays the specified ringer using an input IAStream Interface object, allowing the caller to test a ringer that may be located in a file (IFile Interface), in memory (IMemAStream Interface), or through the network (ISocket Interface).

Prototype:

```
int IRINGERMGR_PlayStream
      (
      IRingerMgr * pIRingerMgr,
      IAStream * pStream,
      uint32 dwPause
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pStream	Input file name.
dwPause	Time, in milliseconds, to pause between replays; 0 (zero) if single play.

Return Value:

SUCCESS, ringer begins playing.

EFAILED, invalid input stream or unable to play ringer.

Other implementation-specific error codes.

Comments:

This function only supports network stream from TCP socket but not UDP socket.

See Also:

None

Return to the List of functions

IRINGERMGR_RegisterNotify()

Description:

This function registers or deregisters a notification callback when playback or creation events are complete.

Prototype:

```
void IRINGERMGR_RegisterNotify
      (
      IRingerMgr * pIRingerMgr,
      PFNRINGEREVENT pfn,
      void * pUser
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
pfn	Pointer to the user callback (NULL to deregister).
pUser	Pointer to user data for callback. It can be NULL if no identifying data is required.

Return Value:

None

Comments:

None

See Also:

PFNRINGEREVENT

Return to the List of functions

IRINGERMGR_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

IRINGERMGR_AddRef()

Return to the List of functions

IRINGERMGR_Remove()

Description:

This function removes the specified ringer.

Prototype:

```
int IRINGERMGR_Remove(IRingerMgr * pIRingerMgr, AEERingerID id)
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
id	Ringer ID.

Return Value:

SUCCESS, ringer removed.

EFAILED, invalid ringer ID or unable to remove ringer.

Other implementation-specific error codes.

Comments:

This function cannot be used to remove OEM installed ringers.

See Also:

AEERingerID

Return to the List of functions

IRINGERMGR_SetRinger()

Description:

This function allows the caller to to set a ringer for the specified category.

Prototype:

```
int IRINGERMGR_SetRinger
      (
      IRingerMgr * pIRingerMgr,
      AEERingerCatID idCat,
      AEERingerID id
      )
```

Parameters:

pIRingerMgr	Pointer to the IRingerMgr Interface object.
idCat	Category for the ringer.
id	ID of the ringer.

Return Value:

SUCCESS, ringer set.

Other error codes.

EFAILED, ringer not set.

Comments:

None

See Also:

AEERingerID

AEERingerCatID

IRINGERMGR_EnumNextCategory()

Return to the List of functions

IRINGERMGR_Stop()

Description:

This function terminates the playback of a ringer.

Prototype:

```
int IRINGERMGR_Stop(IRingerMgr * pIRingerMgr)
```

Parameters:

pIRingerMgr Pointer to the IRingerMgr Interface object.

Return Value:

SUCCESS, ringer is stopping.

Other error codes.

EFAILED, no ringer is playing.

Comments:

None

See Also:

IRINGERMGR_Play()

Return to the List of functions

IRUIM Interface

The Interface provides

List of Header files to be included

The following header file is required:

AEERUIM.h

List of functions

Functions in this interface include:

IRUIM_AddRef()
IRUIM_CHVDisable()
IRUIM_CHVEnable()
IRUIM_GetCHVStatus()
IRUIM_GetId()
IRUIM_GetPrefLang()
IRUIM_IsCardConnected
IRUIM_PINChange()
IRUIM_PINCheck()
IRUIM_QueryInterface()
IRUIM_Release()
IRUIM_UnblockCHV()
IRUIM_VirtualPINCheck()
OEMRUIMAddr_GetFuncs()

The remainder of this section provides details for each function.

IRUIM_AddRef()

Description:

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

Prototype:

```
uint32 IRUIM_AddRef(IRUIM *pIRUIM)
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

Return Value:

Incremented reference count for the object.

Comments:

A valid object returns a positive reference count. Otherwise, 0 (zero) is returned.

See Also:

IRUIM_Release()

Return to the List of functions

IRUIM_CHVDisable()

Description:

This function will disable CHV1 if the last call to IRUIM_PINCheck() was successful.

Prototype:

```
int IRUIM_CHVDisable(IRUIM *pIRUIM)
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

Return Value:

AEE_SUCCESS if CHV1 was disabled,

EFAILED otherwise or other OEM specified error code

Comments:

The successful completion of this command allows unprotected access to all files protected by CHV1. Ths function performs the DISABLE CHV functionality as described in 3GPP TS 11.11.

See Also:

None

Return to the List of functions

IRUIM_CHVEnable()

Description:

This function will enable CHV1 if the passed in PIN is correct.

Prototype:

```
boolean IRUIM_CHVEnable(IRUIM *pIRUIM, const char *pPin)
```

Parameters:

pIRUIM	Pointer to the IRUIM Interface object.
pPin	Pointer to an eight digit character string. If NULL, use the PIN previously entered with IRUIM_PINCheck(). This does not need to be NULL terminated.

Return Value:

TRUE, if CHV1 was enabled,

FALSE, if otherwise

Comments:

If CHV1 has been disabled, you need to include the CHV1 as the argument to this function since the 'ENABLE CHV1' command requires it. If CHV1 is enabled, the user will have already entered CHV1 so you do not need to include it as an argument (pass in NULL). This function performs the ENABLE CHV functionality as described in 3GPP TS 11.11. If the operation returns TRUE, the PIN is stored and will be used for the next call to IRUIM_VirtualPINCheck().

See Also:

IRUIM_PINCheck()

IRUIM_VirtualPINCheck()

Return to the List of functions

IRUIM_GetCHVStatus()

Description:

This function returns the current R-UIM status

Prototype:

```
int IRUIM_GetCHVStatus(IRUIM *pIRUIM, AEECHVStatus *pCHVStatus)
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

pCHVStatus: [out]. Pointer to the returned CHV status.

Return Value:

AEE_SUCCESS if operation succeeded

EFAILED otherwise or other OEM specified error code

Comments:

None.

See Also:

None

Return to the List of functions

IRUIM_GetId()

Description:

This function returns the identification number of the R-UIM.

Prototype:

```
int IRUIM_GetId(IRUIM *pIRUIM, char *pId, int *pnLen)
```

Parameters:

pIRUIM	[in]	Pointer to the IRUIM Interface object.
pId	[in/out]	Card ID. If set to NULL when called, then pnLen will contain the required size of the ID when the function returns.
pnLen	[in/out]	If pId is NULL when this function is called, pnLen will return the number of bytes required to hold the entire Id. Otherwise, pnLen should be set to the number of bytes requested and will return the number of bytes actually provided.

Return Value:

AEE_SUCCESS if no problem occurred

ENOMEMORY - Insufficient memory

Or other OEM specified error code

Comments:

None.

See Also:

None

Return to the List of functions

IRUIM_GetPrefLang()

Description:

This function returns the highest priority preferred language on the R-UIM card.

There may be a second, third,...,nth preferred language on theR-UIM. This function will only return the first preferred langauge.

Prototype:

```
int IRUIM_GetPrefLang(IRUIM *pIRUIM, int *pLang, int *pEncoding)
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

pLang: [out]. Most preferred langauge returned from the R-UIM.

pEncoding: [out]. Encoding of the most preferred language.

Return Value:

None

Comments:

AEE_SUCCESS if no problems occurred,

EBADPARM - Parameters invalid

ENOMEMORY - Insufficient memory

Or other OEM specified error code

See Also:

None

Return to the List of functions

IRUIM_IsCardConnected

Description:

This function checks to see if the R-UIM card is present.

Prototype:

```
boolean IRUIM_IsCardConnected(IRUIM *pIRUIM)
```

Parameters:

pIRUIM	Pointer to the IRUIM Interface object.

Return Value:

TRUE if a R-UIM card was found,

otherwise FALSE.

Comments:

FALSE will be returned if the card is not initialized, is powered down, or is faulty.

See Also:

None

Return to the List of functions

IRUIM_PINChange()

Description:

This function will change the designated CHV on the R-UIM to the PIN passed in. IRUIM_PINCheck() or another command that sets the virtual pin must be called before this command.

Prototype:

```
int IRUIM_PINChange
      (
      IRUIM *pIRUIM,
      AEECHVType chv,
      const char *pPin
      )
```

Parameters:

pIRUIM	Pointer to the IRUIM Interface object.
chv	PIN to be checked.
pPin	Pointer to an eight digit character string to be used as the new PIN. This does not need to be NULL terminated.

Return Value:

AEE_SUCCESS if the PIN was changed,

EFAILED Or other OEM specified error code

Comments:

The appropriate CHV must be enabled and not blocked when this function is called. This function performs the CHANGE CHV functionality as described in 3GPP TS 11.11. If the operation returns AEE_SUCCESS, the new PIN will be stored and used for the next call to IRUIM_VirtualPINCheck().

See Also:

IRUIM_PINCheck()

IRUIM_VirtualPINCheck()

Return to the List of functions

IRUIM_PINCheck()

Description:

This function will compare the designated CHV on the R-UIM with the PIN passed in.

Prototype:

```
boolean IRUIM_PINCheck(IRUIM *pIRUIM, AEECHVType chv, const char
*pPin)
```

Parameters:

pIRUIM	Pointer to the IRUIM Interface object.
chv	PIN to be checked.
pPin	Pointer to an eight digit character string.

Return Value:

TRUE if the user input matches the CHV on the R-UIM, otherwise FALSE.

FALSE will be returned if is NULL.

FALSE will also be returned if the RIUM is not connected as defined above in the description of IRUIM_IsCardConnected().

Comments:

The appropriate CHV must be enabled and not blocked when this function is called. This function performs the VERIFY CHV functionality as described in 3GPP TS 11.11. If the operation returns TRUE, the PIN is stored and will be used for the next call to IRUIM_VirtualPINCheck().

See Also:

IRUIM_VirtualPINCheck()

Return to the List of functions

IRUIM_QueryInterface()

Description:

This function retrieves a pointer to an interface conforming to the specified class ID. This can be used to query for extended functionality, like future versions or proprietary extensions. Upon a successful query, the interface is returned after the AddRef methodis called. The caller is responsible for call the Release method at some point in the future.

Prototype:

```
int IRUIM_QueryInterface
      (IRUIM *pIRUIM,
      AEECLSID idReq,
      void **ppo)
```

Parameters:

pIRUIM	[in]	Pointer to the IRUIM Interface object.
idReq	[in]	Requested class ID exposed by the object
ppo	[out]	Returned object. Filled by this method

Return Value:

AEE_SUCCESS - Interface found

ENOMEMORY - Insufficient memory

ECLASSNOTSUPPORT - Requested interface is unsupported

EFAILED - Any general failure.

Or other OEM specified error code

Comments:

The pointer in *ppo is set to the new interface (with refcount positive), or NULL if the ClassID is not supported by the object. This function can be called with idReq = AEECLSID_ADDRBOOK to return an address book object. However, CHV1 must have been previously disabled or verified before this function is called.

The ppo MUST not be NULL.

See Also:

IRUIM_CHVDisable()

IRUIM_PINCheck()

IAddrBook Interface

Return to the List of functions

IRUIM_Release()

Description:

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

Prototype:

```
uint32 IRUIM_Release(IRUIM *pIRUIM)
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

Return Value:

Decremented reference count for the object.

0 (zero), If the object has been freed and is no longer valid.

Comments:

None

See Also:

IRUIM_AddRef()

Return to the List of functions

IRUIM_UnblockCHV()

Description:

This function will unblock a CHV which has been previously blocked using the passed in unblock CHV and PIN. An application must have PL_SYSTEM privilege complete this function.

Prototype:

```
int IRUIM_UnblockCHV(IRUIM *pIRUIM,
      AEECHVType chv,
      char *pUnblockPin,
      char *pPin)
```

Parameters:

pIRUIM	Pointer to the IRUIM Interface object.
chv	PIN to be unblocked.
pUnblockPin	Pointer to an eight digit character unblock string.This does not need to be NULL terminated.
pPin	Pointer to an eight digit character string. This does not need to be NULL terminated.

Return Value:

AEE_SUCCESS if the user input was valid.

EPRIVLEVEL if the calling application. does not have PL_SYSTEM privilege.

EFAILED for other errors

Comments:

CHV1 will be restored to a the pPin value. This function performs the UNBLOCK CHV functionality as described in 3GPP TS 11.11. If the operation returns AEE_SUCCESS, the PIN is stored and will be used for the next call to IRUIM_VirtualPINCheck().

See Also:

None

Return to the List of functions

IRUIM_VirtualPINCheck()

Description:

If IRUIM_PINCheck() has been previously called, this function can be used to re-verify the CHV1 PIN without accessing the R-UIM. It compares the input PIN with the previous PIN value.

Prototype:

```
boolean IRUIM_VirtualPINCheck
      (IRUIM *pIRUIM,
      AEECHVType chv,
      const char *pPin
      )
```

Parameters:

pIRUIM: [in]. Pointer to the IRUIM Interface object.

chv: [in]. PIN to be checked.

pPin: [in]. Pointer to the eight digit character string.

Return Value:

TRUE if the user input matches the previously read value of CHV1 on the R-UIM, otherwise FALSE.

FALSE FALSE will be returned if is NULL.

FALSE will also be returned if the R-UIM is not connected as defined above in the description of IRUIM_IsCardConnected().

Comments:

This function should only be called after IRUIM_PINCheck() has been called successfully.

See Also:

IRUIM_PINCheck()

Return to the List of functions

OEMRUIMAddr_GetFuncs()

Description:

This function is called when the BREW AddressBook interface is created with a class ID of AEECLSID_RUIM. This function returns the OEM RUIM functions that are needed by the IAddrBook interface to access the R-UIM phonebook.

Prototype:

```
VTBL(IOEMAddrBook) *OEMAddrBook_Init(void);
```

Parameters:

None

Return Value:

Returns the table of OEMRUIM functions needed by the IAddrBook interface to access the R-UIM phonebook.

Comments:

None

See Also:

IRUIM_QueryInterface()

Return to the List of functions

ITAPI Interface

TAPI is a simple interface to the telephony layer in the device. It provides the following services:

- Retrieving Telephony status
- Placing voice calls
- Extracting SMS text from SMS messages
- Obtaining caller ID on incoming/in-progress calls
- Registering for SMS Messages

The ITAPI interface is obtained via the ISHELL_CreateInstance mechanism.

Notifications Sent by this Class:

The TAPI class allows applications to register for the following Notifications:

a. NMASK_TAPI_STATUS
b. NMASK_TAPI_SMS_TEXT
c. NMASK_TAPI_SMS_TS

Receiving SMS Messages:

BREW Applications can register to be notified when a SMS message comes to the system. This registration can be done in the MIF. When the message comes into the system, applications that have registered to be notified receive the EVT_NOTIFY event. The dwParam of this event contains detailed information about the message.

The following masks can be used to register for SMS Notifications through BREW:

1. NMASK_TAPI_SMS_TEXT: This allows applications to register for all Text Messages (messages with TeleServce ID: SMS_TELESERVICE_CMT_95). When a text message arrives, the application is notified through the EVT_NOTIFY event. The dwParam of this event is of type AEENotify. The pData member in this AEENotify Structure will be of type AEESMSTextMsg and contains the actual text message.

2. NMASK_TAPI_SMS_TS: This allows applications to register for SMS message of a specific TeleService ID. To construct the actual 32-bit mask to be used in the MIF for the registration, the upper 16 bits of the mask must contain the TeleService ID value and the lower 16 bits must contain the value 0x0004 (which corresponds to NMASK_TAPI_SMS_TS).

Example:

> To register for a message with TS ID: 0x1002, the 32-mask should be: 0x10020004

Applications can register for multiple messages by creating the 32-mask for each message.

Example:

> To register for messages with TS IS: 1002 and 1003, the application must register two separate masks: 0x10020004 and 0x10030004.

When the SMS message of this TS ID is received by the system, the application is notified through the EVT_NOTIFY event. The dwParam of this event is of type AEENotify. The pData member in the AEENotify Structure is of type AEESMSMsg. This structure contains detailed information about the message.

Registering for Device Status Change:

NMASK_TAPI_STATUS: Applications can use the TAPI class to be notified whenver there is a change in the telephony status of the device. To register for this notification, applications must use the mask NMASK_TAPI_STATUS.

Whenever there is a status-change, applications receive the EVT_NOTIFY event. The dwParam of this event is of type AEENotify.

The pData member inside this AEENotify structure is of type TAPIStatus and contains detailed information about the current telephony status of the device.

List of Header files to be included

The following header file is required for ITAPI

AEETAPI.h

List of functions

Functions in this interface include:

ITAPI_AddRef()
ITAPI_ExtractSMSText()
ITAPI_GetCallerID()
ITAPI_GetStatus()
ITAPI_IsDataSupported()
ITAPI_IsVoiceCall()
ITAPI_MakeVoiceCall()
ITAPI_OnCallStatus()
ITAPI_OnCallEnd()
ITAPI_Release()
ITAPI_SendSMS()

The remainder of this section provides details for each function.

ITAPI_AddRef()

Description:

This function is inherited from IBASE_AddRef()

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 an AEESMSMsg.

The typical means to use this function:

- When applications register for a text message by using the notification mask NMASK_TAPI_SMS_TS and by specifying the TeleService ID as SMS_TELESERVICE_CMT_95, the application gets notified using the EVT_NOTIFY mechanism.
- The **dwParam** to this event is of type AEENotify. The **pData** inside the AEENotify structure is of type AEESMSMsg.
- For text messages, this function ITAPI_ExtractSMSText() can be invoked using this AEESMSMsg so as to extract the actual text portion of the message.
- A recommended way for applications to register for text messages is using the notification mask: NMASK_TAPI_SMS_TEXT.
- When this mask is used, the **pData** inside the AEENotify structure received during the notification is already of type AEESMSTextMsg and contains the actual text of the message.

Prototype:

```
AEESMSTextMsg * ITAPI_ExtractSMSText
      (
      ITAPI * pITAPI,
      const AEESMSMsg * pMsg
      )
```

Parameters:

pITAPI	Pointer to the ITAPI Interface object.
pMsg	Pointer to the input AEESMSMsg.

Return Value:

NULL, if this function fails.

If successful, this function returns a pointer to AEESMSTextMsg containing the actual text. This buffer is valid until the next call to ITAPI_ExtractSMSText() or until the interface is released.

Comments:

None

See Also:

AEESMSMsg

AEESMSTextMsg

Return to the List of functions

ITAPI_GetCallerID()

Description:

This function retrieves the ID, in digits, of an incoming or outgoing voice call.

Prototype:

```
boolean ITAPI_GetCallerID(ITAPI * pITAPI, AECHAR * pDest, int nSize)
```

Parameters:

pITAPI	Pointer to the ITAPI Interface object.
pDest	Destination pointer.
nSize	Size in bytes of the destination buffer.

Return Value:

TRUE, if call in progress and buffer filled.

FALSE, if no call in progress or invalid buffer.

Comments:

None

See Also:

None

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:

```
int ITAPI_GetStatus(ITPAI * pITAPI, TAPIStatus * ps)
```

Parameters:

pITAPI	Pointer to the ITAPI Interface object.
ps	Pointer to the status information to be filled.

Return Value:

SUCCESS, if valid status information.

EBADPARM, if bad parameter.

Comments:

None

See Also:

TAPIStatus

ISHELL_RegisterNotify()

Return to the List of functions

ITAPI_IsDataSupported()

Description:

This method can be used to determine whether the handset supports data service.

Prototype:

```
boolean ITAPI_IsDataSupported(ITapi *pITAPI)
```

Parameters:

pITAPI	Pointer to the ITAPI object

Return Value:

e:

TRUE, if the handset supports data service

FALSE, if the handset does not support data service

Comments:

None

See Also:

None

Return to the List of functions

ITAPI_IsVoiceCall()

Description:

This method can be used to determine whether the current call in the system is a voice call.

Prototype:

```
boolean ITAPI_IsVoiceCall(ITapi *pITAPI)
```

Parameters:

pITAPI	Pointer to the ITAPI object

Return Value:

TRUE, if the current call is a voice call

FALSE, if the current call is NOT a voice call

Comments:

None

See Also:

ITAPI_MakeVoiceCall()

Return to the List of functions

ITAPI_MakeVoiceCall()

Description:

This method is called to place a voice call. The number dialed is specified in the digits string. No call is placed if the input string is empty or NULL. Only the following digits are allowed: **0-9**, **#**, *****. 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 (TCP), the data call is ended and the call is placed.

This function enforces the privacy policies established by the carrier. This may include intermediate prompts to the user using dialogs.

Typically, when this function is invoked, a dialog is displayed to the user requesting whether it is OK to place a call. When the user clicks “YES”, the call is placed.

The event flow to the application when this function is invoked:

1. A dialog is displayed to the user.
2. When that dialog is dismissed, the event EVT_DIALOG_END is sent to the application.
3. At this point, the application must re-draw the screen.
4. If user accepted to place the call, the event EVT_APP_SUSPEND is sent to the application.
5. When the call finishes, the event EVT_APP_RESUME is sent to the application.
6. The application must re-draw the screen.

Prototype:

```
int ITAPI_MakeVoiceCall
      (
      ITAPI * pITAPI,
      const char * pszNumber,
      AEECLSID clsReturn
      )
```

Parameters:

pITAPI	Pointer to the ITAPI Interface object
pszNumber	Pointer to number to dial
clsReturn	Classid of the applet to be run when the call is completed

Return Value:

SUCCESS, if the function is in progress.

EBADPARM, if the number is invalid.

EALREADY, if there is a voice call already in progress.

Comments:

If **clsReturn** is 0, the current application will be resumed.

When ITAPI_MakeVoiceCall is invoked. After the privacy dialog is selected by user - EVT_DIALOG_END is sent to the application with **dwParam** indicating the response. The **dwParam** has 1 for a "Yes" response and 2 for "No" response from user

See Also:

AEECLSID

Return to the List of functions

ITAPI_OnCallEnd()

Description:

This method is identical to ITAPI_OnCallStatus().

Prototype:

```
int ITAPI_OnCallEnd
      (
      ITAPI * pITAPI,
      PFNNOTIFY pfn,
      void * pUser,
      uint32 dwDelay,
      uint16 wFlags
      )
```

Parameters:

See ITAPI_OnCallStatus()

Return Value:

See ITAPI_OnCallStatus()

Comments:

None

See Also:

See ITAPI_OnCallStatus()

Return to the List of functions

ITAPI_OnCallStatus()

Description:

This method can be used to register a Callback function that will be invoked by BREW when there is a change in the call-status. It supports flags that can be used to specify what type of call-states the application cares about.

Prototype:

```
int ITAPI_OnCallStatus
      (
      ITAPI * pITAPI,
      PFNNOTIFY pfn,
      void * pUser,
      uint32 dwDelay,
      uint16 wFlags
      )
```

Parameters:

pITAPI	Pointer to the ITAPI object
pfn	Notification function to be called when any call-status changes
pUser	User Data to be passed to the notification function when it is invoked.
dwDelay	The time period in milliseconds BREW waits after the call-state has changed and before notifying the application
wFlags	:The following flags are supported:

OCS_CANCEL	Cancel a previously registered Callback Function
OCS_UNIQUE_PFN	When this flag is set, any previous registrations of the same callback function with different data pointers are cancelled.
OCS_ONE_SHOT	Informs BREW that this CB function is to be registered for just one notification. Once a single call-status change occurs, this notification is invoked and the CB function is removed from the internal list. This CB will not longer be invoked.
OCS_INCOMING	Register for notifications when there is an incoming call
OCS_ORIG	Register for notifications when call-originations happen
OCS_CONVERSATION	Register for notifications when call enters the conversation state (i.e. Two way state)
OCS_IDLE	Register for notifications when the call is ended
OCS_OFFLINE	Register for notifications when the device loses coverage

OCS_ALL	Register for all call-state transitions (incoming, orig, conversation, idle)

Return Value:

SUCCESS, if successfully registered

ENOMEMORY, if failed

Comments:

None

See Also:

None

Return to the List of functions

ITAPI_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

ITAPI_AddRef()

Return to the List of functions

ITAPI_SendSMS()

Description:

This method is used to send SMS messages from the handset.This function can be used to send text messages to either a specific BREW application on another handset or a general text message to the handset. It sends messages to a destination mobile number or an email ID. When this function is called, it is likely that it will result in a SUSPEND/RESUME sequence to the BREW application. This is initiated by the handset when it sends the SMS message out.

The sequential flow of control is as follows:

1. The application invokes ITAPI_SendSMS()
2. The handset begins the process of sending the SMS message out
3. The notification function specified by **pfn** is invoked with the status of the message delivery

Associated with this process, the application may receive EVT_APP_SUSPEND and EVT_APP_RESUME events sometime after step 1.

Prototype:

```
int ITAPI_SendSMS
     (
     ITAPI *pITapi,
     const char * pszDst,
     const char * pszMsg,
     AEECLSID clsDst,
     PFNSMSSTATUS pfn,
     void *pUser
     )
```

Parameters:

pITapi	Pointer to the ITAPI Interface object
pszDst	Number or email ID of the destination where message must be sent to. If this is set to NULL and if **clsDst** is non-zero, this function sends the EVT_APP_MESSAGE event to the application (clsDst) on the local handset and the **dwParam** of that event shall contain pszMsg. In this case, the return value of the function is the same as the return value of ISHELL_SendEvent(). The notification function will not be called since this is a local delivery of the message.
pszMsg	Text message to be sent to the destination mobile. If this is set to NULL, the function returns EBADPARM
clsID	If non-zero, it specifies the class ID of the BREW applicaiton on the destination mobile to which this message must be sent

pfn	Notification function that is invoked to inform the status of the SMS message sent
pUser	User data to be sent to the notification function

Examples:

```
ITAPI_SendSMS (pITapi, "8581112222", "Hello World",
0, MyMOSMSNotify, pMe );

ITAPI_SendSMS ( pITapi, "foo@sample.com", "Hello
World", 0, MyMOSMSNotify, pMe );
```

Return Value:

SUCCESS, if successful. After the message is sent to the other handset, the notification function is invoked with the status. The status passed to the notification function can be SUCCESS or EFAILED.

EITEMBUSY, if the device is busy and cannot send this SMS. Generally, a SMS message cannot be sent if the notification function has not yet been called for a previously sent SMS message.

EBADPARM, if pszMsg is NULL or pszDst and clsDst are both set to NULL.

EBADCLASS, if TAPI is not enabled on this handset.

EFAILED, if there is a general failure in sending the SMS.

Examples:

```
ITAPI_SendSMS(pITapi, "8581112222", "Hello World", 0,
MyMOSMSNotify, pMe);

ITAPI_SendSMS(pITapi, "foo@sample.com", "Hello
World",0,MyMOSMSNotify,pMe);
```

Comments:

None

See Also:

None

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 device user. The text control handles the translation of device user key presses into characters. Your application only needs to pass keypress events to the text control while it is active and retrieve the text from the control when device 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 device 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 makes it possible for the device user to select the specified mode while the text control is active. The text control allows you to specify a Soft Key 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, that 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_FIXSETRECT, if set, the actual height more closely represents requested height.

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 device 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_SetSoftKeyMenu() associates a Soft Key menu control with the text control. This is typically a Soft Key menu that you have created and that appears on the screen while the text control is displayed. ITEXTCTL_SetSoftKeyMenu() adds an item to the Soft Key menu that allows the device 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 device user to select the new text entry mode. After the device user selects the new mode, the text control is reactivated and the device user continues entering text. While entering text, the device user can press the SELECT key to leave text-edit mode and activate the Soft Key menu. While the Soft Key menu is active, the device 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 contains the text control.
3. If specified, call ITEXTCTL_SetTitle() 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 ITEXTCTL_SetSoftKeyMenu() to specify the Soft Key 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 device user has completed entering text, call ITEXTCTL_GetText() or ITEXTCTL_GetTextPtr() to retrieve the text the device user has entered. (If you are using a Soft Key menu, the device 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 Soft Key menu is present).
9. Call ITEXTCTL_Release() to free the text control when you no longer need it.

List of Header files to be included

The following header file is required for ITextCtl

AEEText.h

List of functions

Functions in this interface include:

The remainder of this section provides details for each function.

ITEXTCTL_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

ITEXTCTL_Release()

Return to the List of functions

ITEXTCTL_EnumModeInit()

Description:

This function initializes the mode enumeration mechanism for the test control. Any time you want to enumerate the text control, this function must be called first.

Prototype:

void ITEXTCTL_EnumModeInit(ITextCtl * pITextCtl)

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

ITEXTCTL_EnumNextMode()

Description:

This function is called to enumerate the text control modes.

Prototype:

```
AEETextInputMode ITEXTCTL_EnumNextMode
      (
      ITextCtl * pITextCtl,
      AEETextInputModeInfo * pmInfo
      )
```

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
pmInfo	Optional pointer to receive Text Mode Information. If you do not wish to receive this information, call this function with a NULL value as the second parameter.

Return Value:

An enum of type AEETextInputMode to indicate the next input mode.

If the enumeration is complete AEE_TM_NONE will be returned.

Comments:

None

See Also:

AEETextInputMode

Return to the List of functions

ITEXTCTL_GetCursorPos()

Description:

This function gets the position of a cursor in a text control object.

Prototype:

```
int32 ITEXTCTL_GetCursorPos(ITextCtl * pITextCtl)
```

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object

Return Value:

Absolute position of cursor in text control

Comments:

None

See Also:

ITEXTCTL_SetCursorPos()

Return to the List of functions

ITEXTCTL_GetInputMode()

Description:

This function allows the caller to get the selected text input mode and the string associated with it.

Prototype:

```
AEETextInputMode ITEXTCTL_GetInputMode
      (
      ITextCtl *pITextCtl,
      AEETextInputModeInfo * pmInfo
      )
```

Parameters:

pITextCtl	[in]	Pointer to the ITextCtl Interface object
pmInfo	[in/out]	Input: a pointer to a AEETextInputModeInfo Info structure to be filled OR can be NULL, so as to not fill a structure and return current mode. Output: If a valid pointer is given it is filled with the current mode and the string associated with that mode.

Return Value:

An enum of type AEETextInputModeInfo to indicate the input mode set.

Comments:

If a AEETextInputModeInfo pointer is given the **tmMode** field, it will match the return value of this function. The **pmInfo** field is not required if the callee is just checking the AEETextInputModeInfo and does not need the string associated with it.

See Also:

AEETextInputModeInfo

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.

TP_FIXSETRECT, if set, the actual height more closely represents requested height.

Comments:

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 size only for text, not title. This is particularly useful after a control is created to determine its optimal/default size and position.

NOTE:

If the property TP_FIXSETRECT is set, this function fills the AEERect with the actual bounding rectangle of the control, which is not necessarily the rectangle passed in ITEXTCTL_SetRect().

If the property TP_FIXSETRECT is NOT set, this function returns the rectangle that was passed in to ITEXTCTL_SetRect().

Prototype:

```
void ITEXTCTL_GetRect(ITextCtl * pITextCtl, AEERect * prc)
```

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object.
prc	Rectangle to be filled with the coordinates of the text control object.

Return Value:

None

Comments:

None

See Also:

AEERect

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,
      AECHAR * 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

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:

```
AECHAR * 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

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 processes only set title, set text, and the pressing the 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 the 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

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

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

See Also:

None

Return to the List of functions

ITEXTCTL_Release()

Description:

This function is inherited from IBASE_Release().

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 and 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

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

See Also:

None

Return to the List of functions

ITEXTCTL_SetCursorPos()

Description:

This function is used to set the position of a cursor in a text control object. You can use the following defines for **nOffset** to place the text at the start or end.

Prototype:

```
void ITEXTCTL_SetCursorPos(ITextCtl * pITextCtl, int32 nOffset)
```

Parameters:

pITextCtl	Pointer to the ITextCtl Interface object
nOffset	Placement of the text object TC_CURSOREND - Place the cursor at the end of the text. TC_CURSORSTART - Place the cursor at the beginning of the text.

Return Value:

None

Comments:

If **nOffset** is < 0 the cursor is placed at the beginning of the text.

If **nOffset** is > the Length of the text, the cursor is placed at the end of the text

See Also:

ITEXTCTL_GetCursorPos()

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:

An enum of type AEETextInputMode to indicate the input mode set.

Comments:

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:

The implementation of this function may vary between devices. Some devices may allow text to be entered beyond the maximum size set by this function.

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.

TP_FIXSETRECT, if set, the actual height more closely represents requested height.

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 fills the AEERect data structure with the coordinates of the current bounding rectangle to determining the size of the text, not the title. This is particularly useful after a control is created to determine its optimal/default size and position.

NOTE:

If the property TP_FIXSETRECT is set, this function fills the AEERect data structure with the actual bounding rectangle of the control, which may not be the rectangle set using ITEXTCTL_SetRect().

If the property TP_FIXSETRECT is NOT set, this function returns the AEERect data structure which contains the coordinates of rectangle set using ITEXTCTL_SetRect().

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 a device screen width as width and (device screen height - text height) as height starting from the upper left corner.

See Also:

AEERect

ITEXTCTL_GetRect()

Return to the List of functions

ITEXTCTL_SetSoftKeyMenu()

Description:

This function replaces the existing Soft Key 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 AECHAR * 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

None

Comments:

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,
      AECHAR * 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

ITransform Interface

ITransform, like IBitmap, provides functions for accessing a bitmap. This interface provides functions for doing blits with transformations. It supports two kinds of transformation simple (with ITRANSFORM_TransformBltSimple()) and arbitrary Affine Transforms (with ITRANSFORM_TransformBltComplex()).

Unlike many BREW interfaces, ITransform cannot be obtained through ISHELL_CreateInstance(). Instead, it may be obtained through the QueryInterface method of a bitmap object that supports it. For instance, if you had an IBitmap Interface to a bitmap object, you would call IBITMAP_QueryInterface() with the class ID AEECLSID_TRANSFORM.

Not all bitmap implementations support ITransform.

List of Header files to be included

The following header file is required:

AEETransform.h

List of functions

Functions in this interface include:

ITRANSFORM_AddRef()
ITRANSFORM_QueryInterface()
ITRANSFORM_Release()
ITRANSFORM_TransformBltComplex()
ITRANSFORM_TransformBltSimple()

The remainder of this section provides details for each function.

ITRANSFORM_AddRef()

Description:

This function is inherited from IBASE_AddRef().

See Also:

ITRANSFORM_Release()

Return to the List of functions

ITRANSFORM_QueryInterface()

Description:

This function retrieves a pointer to an interface conforming to the definition of the specified class ID. This can be used to query for extended functionality, like future versions or proprietary features. Upon a successful query, the interface is returned with an incremented instance. The caller is responsible for calling Release() at some point in the future. One exception is when the pointer returned is not an interface pointer. In that case, the memory will share the lifetime of the object being queried, and the returned pointer will not be used to free or release the object.

Prototype:

int ITRANSFORM_QueryInterface(ITransform *pITransform, AEECLSID id, void **p);

Parameters:

pITransform	[in]	Pointer to ITransform interface.
id	[in]	A globally unique id to identify the entity (interface or data) that we are trying to query.
p	[out]	Pointer to the data or interface that we want to retrieve. If the value passed back is NULL, the interface or data that we query are not available.

Return Value:

SUCCESS, if successful.

Error code, if otherwise.

Comments:

On failure, ***p** should be set to NULL, but it is good form to explicitly set ***p** to NULL before calling QueryInterface().

See Also:

ITransform Properties

Return to the List of functions

ITRANSFORM_Release()

Description:

This function is inherited from IBASE_Release().

See Also:

ITRANSFORM_AddRef()

Return to the List of functions

ITRANSFORM_TransformBltComplex()

Description:

This function blits a one bitmap to another, applying a set of arbitrary Affine transformations.

Prototype:

```
int ITRANSFORM_TransformBltComplex
      (
      ITransform *pITransform,
      int xDst,
      int yDst,
      IBitmap *pSrc,
      int xSrc,
      int ySrc,
      unsigned dxSrc,
      unsigned dySrc,
      const AEETransformMatrix *pMatrixTransform,
      uint8 unComposite
      )
```

Parameters:

pITransform	Pointer to the ITransform interface of the destination bitmap
xDst, yDst	Coordinate in destination where upper left corner of source will be drawn. This assumes that no transform is being applied. The location is specified by the center of the source area, which is drawn at (xDst + dxSrc / 2, yDst + dySrc / 2).
pSrc	Source bitmap. The bitmap types supported for pSrc vary from implementation to implementation, but when pSrc is of the same type as the destination bitmap, support is guaranteed.
xSrc, ySrc	Upper left corner of source bitmap to be blitted to destination.
dxSrc, dySrc	Width and height of source bitmap area to be blitted.
pMatrixTransform	Pointer to AEETransformMatrix structure that specifies the transformation to be used.
unComposite	See ITransform Properties.

Return Value:

SUCCESS, if successful.

EUNSUPPORTED, if blit is not supported.

Comments:

None

See Also:

ITRANSFORM_TransformBltSimple()

AEETransformMatrix

ITransform Properties

Return to the List of functions

ITRANSFORM_TransformBltSimple()

Description:

This function blits a one bitmap to another, applying a set of simple, predefined transformations.

Prototype:

```
int ITRANSFORM_TransformBltSimple
      (
      ITransform *pITransform,
      int xDst,
      int yDst,
      IBitmap *pSrc,
      int xSrc,
      int ySrc,
      unsigned dxSrc,
      unsigned dySrc,
      uint16 unTransform,
      uint8 unComposite)
```

Parameters:

pITransform	Pointer to the ITransform interface of the destination bitmap
xDst, yDst	Coordinate in destination where upper left corner of source will be drawn. This assumes that no transform is being applied. The location is specified by the center of the source area, which is drawn at (xDst + dxSrc / 2, yDst + dySrc / 2).
pSrc	Source bitmap. The bitmap types supported for pSrc vary from implementation to implementation, but when pSrc is of the same type as the destination bitmap, support is guaranteed.
xSrc, ySrc	Upper left corner of source bitmap to be blitted to destination.
dxSrc, dySrc	Width and height of source bitmap area to be blitted.
unTransform	Set of flags that specify transformation to perform. See ITransform Properties.
unComposite	See ITransform Properties.

Return Value:

SUCCESS, if successful.

EUNSUPPORTED, if blit is not supported.

Comments:

None

See Also:

ITRANSFORM_TransformBltComplex()

ITransform Properties

Return to the List of functions

OEM AEE Interface

This section describes the Application Execution Environment (AEE) functions that must be called by the OEM layer, and the required/preferred call sequences.

List of functions

Functions in this interface include:

AEE_Active()
AEE_AutoInstall()
AEE_BuildPath()
AEE_CheckPtr()
AEE_CheckStack()
AEE_CreateControl()
AEE_Dispatch()
AEE_EnumRegHandlers()
AEE_Event()
AEE_Exception()
AEE_Exit()
AEE_FreeMemory()
AEE_GetAppContext()
AEE_GetClassInfo()
AEE_GetShell()
AEE_Init()
AEE_IsInitialized()
AEE_IsTestDevice()
AEE_Key()
AEE_KeyHeld()
AEE_KeyPress()
AEE_KeyRelease()
AEE_LinkSysObject()
AEE_NetEventOccurred()
AEE_RegisterForDataService()
AEE_RegisterForValidTime()
AEE_Resume()
AEE_ResumeEx()
AEE_SetAppContext()
AEE_SetEventHandler()
AEE_SetSysTimer()
AEE_SocketEventOccurred()
AEE_Suspend()
AEE_TimerExpired()

The remainder of this section provides details for each function.

AEE_Active()

Description:

This function can be called by the OEM layer to determine the ClassID of the active applet or control in the AEE. It returns 0 (zero) if there is no active applet. The function is provided primarily for key handling so that the OEM layer can determine whether keypad events must be passed to the AEE or handled by the existing user interface (UI).

Prototype:

```
AEECLSID AEE_Active(void)
```

Parameters:

None

Return Value:

AEECLSID of the active applet or control.

0 (zero) if there is no active applet.

Comments:

None

See Also:

None

Return to the List of functions

AEE_AutoInstall()

Description:

This function allows the OEM to automatically install an item on the device. The function performs the following tasks:

- Scans the device to see if the specified item is already installed.
- If the specified item is not installed, displays a “Configuring Appls...” pop-up window.
- Queries the ADS for information about the item and downloads it.
- Runs the first application listed for the item.

Prototype:

```
int AEE_AutoInstall(DLITEMID id, DLPRICEID idPrice)
```

Parameters:

id	ADS/BDS item identifier.
idPrice	ADS/BDS price handle. If 0 (zero), uses the subscription or purchase handle.

Return Value:

EALREADYLOADED if the specified application is already installed.

SUCCESS if the application has been run to download the selected item.

Comments:

None

See Also:

None

Return to the List of functions

AEE_BuildPath()

Description:

This function constructs a fully qualified file path based on the substring passed to the function. It also determines whether the path is in the AEE shared directory.

Prototype:

```
const char * AEE_BuildPath
      (
      IShell * po,
      const char * pszSub,
      char * pszDest,
      uint16 * pwDirType
      )
```

Parameters:

po	Pointer to the ISHELL interface.
pszSub	Input path substring.
pszDest	Final output path string.
pwDirType	Flag that notes whether the path is in the AEE shared directory.

Return Value:

Final output path string.

NULL if the input path is NULL or the length exceeds MAX_FILE_NAME.

Comments:

None

See Also:

None

Return to the List of functions

AEE_CheckPtr()

Description:

This function validates a chunk of memory. It checks for one of the following conditions:

- Memory is non-NULL.
- Memory is in valid range.
- Memory is not in heap.
- Memory update will not overwrite/overread a heap node.

Prototype:

```
boolean AEE_CheckPtr
      (
      const char * pszFunc,
      void * pMem,
      uint32 nSize,
      boolean bWrite
      )
```

Parameters:

pszFunc	Pointer to string detailing the exception if an exception occurs.
pMem	Memory buffer.
dwSize	Size in bytes.
bWrite	Pointer is writeable memory or not.

Return Value:

TRUE if it is a valid pointer.

FALSE otherwise.

Comments:

If the pointer is in RAM but not in the heap, this function returns TRUE.

Sends the following exceptions:

AEE_EXCEPTION_MEMPTR, if the pointer is bad.

AEE_EXCEPTION_HEAP, if the pointer is in the heap, but would overread or overwrite a heap node

See Also:

None

Return to the List of functions

AEE_CheckStack()

Description:

This function checks whether a stack overflow has occurred.

Prototype:

```
void AEE_CheckStack(const char * pszFunc)
```

Parameters:

pszFunc	Pointer to the string detailing the exception if an exception occurs.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_CreateControl()

Description:

This function allows the OEM layer to create user interface controls for use by the existing user interface (UI), if one is provided.

Prototype:

```
int AEE_CreateControl(OEMCONTEXT pdc, AEECLSID iid, void ** po)
```

Parameters:

pdc	Pointer to OEMCONTEXT for the display.
iid	Class of the control.
po	Destination pointer to be filled.

Return Value:

0 (zero) if successful.

Comments:

None

See Also:

ISHELL_CreateInstance() (See the *BREW API Reference Guide*.)

Return to the List of functions

AEE_Dispatch()

Description:

This function must be called by the OEM layer whenever the AEE signal has been detected within the thread where the AEE_Init() call was made. The function performs the following tasks:

- Checks the status of any pending BREW Resume function and calls it.
- Checks to see if any BREW-related timers have expired.

Prototype:

```
uint32 AEE_Dispatch(void)
```

Parameters:

None

Return Value:

AEE_DISPATCH_TIMERS_PENDING, if BREW has active short-term timers pending

AEE_DISPATCH_CALLBACKS_PENDING, if BREW has active callbacks pending

AEE_DISPATCH_THROTTLING if BREW, if is consuming too much time - throttling

0(zero), if BREW has no high-priority callbacks or timers pending.

Comments:

None

See Also:

AEE_Init()

AEE_Exit()

Return to the List of functions

AEE_EnumRegHandlers()

Description:

This helper function provides OEMs with the ability to:

- Enumerate handlers for a specified type/mime type;
- Enumerate all handlers for a specified handler type;
- Enumerate all handlers for all types;

Example:

```
static void EnumTest(Me * pMe)
   {
   AEEClassInfo ci;
   AECHAR szName[32];
   MEMSET(&ci,0,sizeof(AEEHandlerInfo));
   ci.pszAppName = szName;
   ci.nNameSize = sizeof(szName);
   AEE_EnumRegHandlers(pMe,AEECLSID_APP, "application/foo",
      EnumCB, pMe,&ci);
   }
static boolean EnumCB(void * pcxt,AEECLSID clsType,const char *
pszMime, AEECLSID cls, AEEClassInfo * pci)
   {
   Me * pMe = (void *)pcxt;
   AddToMyMenu(pMe,pci);
   return(TRUE);
   }
```

Prototype:

```
int AEE_EnumRegHandlers
      (
      AEECLSID clsType,
      const char * pszMimeType,
      PFNREGCB pfn
      ,void * pUser,
      AEEClassInfo * pci
      )
```

Parameters:

clsType	Type of handlers (example: AEECLSID_APP, AEECLSID_VIEW, etc.)
pszMimeType	Mime type or extension
pfn	Pointer to callback for each entry
pUser	User data pointer for callback
pci	Pointer to AEEClassInfo structure to fill for each entry

Return Value:

0 on SUCCESS

Comments:

Enumeration will stop if handler returns FALSE

See Also:

None

Return to the List of functions

AEE_Event()

Description:

This function sends an event to the active BREW application. It is equivalent to calling ISHELL_SendEvent(), and reduces the complexity of the OEM layers.

Prototype:

```
boolean AEE_Event(AEEEvent evt, uint16 wParam, uint32 dwParam)
```

Parameters:

evt	AEE Event code.
wParam	Extra parameter (16 bits).
dwParam	Extra parameter (32 bits).

Return Value:

TRUE if the event is handled.

FALSE if the event is ignored.

Comments:

None

See Also:

AEE_KeyPress()

AEE_KeyRelease()

AEE_Key()

AEE_KeyHeld()

Return to the List of functions

AEE_Exception()

Description:

This function allows the OEM to send exceptions under special conditions. This causes the following to occur:

- An immediate jump from the current context back to the root of the dispatcher (or message pump).
- A report of the error in a modal form (timer-based pause).
- Termination of the offending BREW application.

Prototype:

```
void AEE_Exception(const char * pszFunc, AEEExceptionType e)
```

Parameters:

pszFunc	Pointer to string detailing the exception.
e	Exception type.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_Exit()

Description:

This function closes the AEE. It must be called to effectively close the BREW layers. The function performs the following tasks:

- Unloads any active applications or modules.
- Frees any memory used by the AEE.
- Releases any open files used by the AEE.
- Releases the SMS layer if the AEE had opened it.

This function must be called by the OEM layer to close the AEE.

Prototype:

```
void AEE_Exit(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

AEE_Init()

Return to the List of functions

AEE_FreeMemory()

Description:

This function can be called by the OEM layer when RAM resources reach a critical low level. In this case, the AEE attempts to free any unused RAM.

Prototype:

```
void AEE_FreeMemory(uint32 nNeeded)
```

Parameters:

nNeeded	Required RAM.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_GetAppContext()

Description:

This function gets the current application context.

Prototype:

```
void * AEE_GetAppContext(void)
```

Parameters:

None

Return Value:

Pointer to the current application context.

Comments:

None

See Also:

AEE_SetAppContext()

Return to the List of functions

AEE_GetClassInfo()

Description:

This helper function is provided for OEMs to gain easy access to extended information regarding BREW classes. Although the data can be obtained through the use of other interfaces (ISHELL_QueryClass and IDOWNLOAD), this function is provided for more ready access to these parameters.

Prototype:

```
int AEE_GetClassInfo(AEECLSID cls, AEEClassInfo * phi)
```

Parameters:

None??

Return Value:

0 on SUCCESS

Comments:

None

See Also:

None

Return to the List of functions

AEE_GetShell()

Description:

This function returns the IShell interface pointer to the active AEE shell. This pointer can then be used by the OEM layer to invoke any other ISHELL interface function.

Prototype:

```
IShell * AEE_GetShell(void)
```

Parameters:

None

Return Value:

Pointer to the shell interface.

Comments:

None

See Also:

None

Return to the List of functions

AEE_Init()

Description:

This function initializes the AEE. During initialization, the AEE performs the following tasks:

- Builds the list of loaded modules (static and dynamic).
- Initializes a timer if an alarm has been set.
- Initializes any pending notification objects.
- Initializes the SMS layer if available.

The AEE passes the operating system signal to the AEE_Init() function when AEE_Dispatch() must be called. The OEM layer must call AEE_Dispatch() during user interface (UI) initialization before making any other AEE calls.

Prototype:

```
IShell * AEE_Init(uint32 dwAEESig)
```

Parameters:

dwAEESig	Operating system signal that is reserved for use by the AEE.

Return Value:

Pointer to the AEE shell object.

Comments:

None

See Also:

AEE_Exit()

Return to the List of functions

AEE_IsInitialized()

Description:

This function checks whether BREW has been initialized.

Prototype:

```
boolean AEE_IsInitialized(void)
```

Parameters:

None

Return Value:

TRUE if initialized.

FALSE otherwise.

Comments:

None

See Also:

None

Return to the List of functions

AEE_IsTestDevice()

Description:

This function checks whether the device is a test device.

Prototype:

```
boolean AEE_IsTestDevice(void)
```

Parameters:

None

Return Value:

TRUE if it is a test device.

FALSE otherwise.

Comments:

None

See Also:

None

Return to the List of functions

AEE_Key()

Description:

This function sends a key event to the AEE. It is important as most BREW applications only process EVT_KEY events generated by this call.

Prototype:

```
boolean AEE_Key(AVKType key)
```

Parameters:

key	BREW keycode (AEEVCodes.h).

Return Value:

TRUE if the event is handled.

FALSE if the event is ignored.

Comments:

None

See Also:

AEE_KeyPress()

AEE_KeyRelease()

AEE_KeyHeld()

Return to the List of functions

AEE_KeyHeld()

Description:

This function sends a key held event to the AEE. It is up to the OEM layer to determine when a key is held.

Prototype:

```
boolean AEE_KeyHeld(AVKType key)
```

Parameters:

key BREW keycode (AEEVCodes.h).

Return Value:

TRUE if the event is handled.

FALSE if the event is ignored.

Comments:

None

See Also:

AEE_KeyPress()

AEE_KeyRelease()

AEE_Key()

Return to the List of functions

AEE_KeyPress()

Description:

This function sends a key press event to the AEE.

Prototype:

```
boolean AEE_KeyPress(AVKType key)
```

Parameters:

key BREW keycode (AEEVCodes.h).

Return Value:

TRUE if the key press is handled.

FALSE if the key press is ignored.

Comments:

Most BREW applications ignore this event. It is provided for games and other complicated applications.

See Also:

AEE_KeyRelease()

AEE_Key()

AEE_KeyHeld()

Return to the List of functions

AEE_KeyRelease()

Description:

This function sends a key release event to the AEE.

Prototype:

```
boolean AEE_KeyRelease(AVKType key)
```

Parameters:

key BREW keycode (AEEVCodes.h).

Return Value:

TRUE if the event is handled.

FALSE if the event is ignored.

Comments:

Most BREW applications ignore this event. It is provided for games and other complicated applications.

See Also:

AEE_KeyPress()

AEE_Key()

AEE_KeyHeld()

Return to the List of functions

AEE_LinkSysObject()

Description:

This function associates an AEESysObject with the currently running application, if any. It's designed to help clean up "system" resources when an application exits. It's intended use is for the AEESysObject to be embedded in the implementing interface object.

Prototype:

```
int AEE_LinkSysObject(AEESysObject *pso)
```

Parameters:

pso	the object to be associated with the application

Return Value:

SUCCESS, if the operaion was successful.

EALREADY, if pso->pac is already set, and pac is a valid application, call AEE_UnlinkSysObject first

Comments:

None

Side Effects:

pso->pac is updated to the application with which the object is associated

See Also:

AEESysObject

Return to the List of functions

AEE_NetEventOccurred()

Description:

This function is the network and socket callback function. it is called when there is any activity with the network/sockets. This is the callback function invoked by the OEMSocket layer. The address is passed to it during the initialization of the network layer.

Prototype:

```
void AEE_NetEventOccurred(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_RegisterForDataService()

Description:

This function allows the OEM layer to take advantage of the BREW notification capability to monitor the system for data service.

Prototype:

```
void AEE_RegisterForDataService
      (
      PFNNOTIFY pfn,
      void * pData,
      boolean bActive
      )
```

Parameters:

pfn	Function to call when the data service is determined.
pData	Callback argument for the data service.
bActive	Callback is only to be called when there is data service and the first BREW application has started.

Return Value:

None

Comments:

None

See Also:

PFNNOTIFY

Return to the List of functions

AEE_RegisterForValidTime()

Description:

This function allows the OEM layer to take advantage of the BREW notification capability to monitor the system for valid time.

Prototype:

```
void AEE_RegisterForValidTime
      (
      PFNNOTIFY pfn,
      void * pUser,
      boolean bActive
      )
```

Parameters:

pfn	Function to call when the valid time is determined.
pUser	Callback argument for the valid time.
bActive	Callback is only to be called when the time is valid and the first BREW application has started.

Return Value:

None

Comments:

None

See Also:

PFNNOTIFY

Return to the List of functions

AEE_Resume()

Description:

This function allows the OEM layer to restart the BREW application that was suspended using AEE_Suspend(). This is equivalent to calling ISHELL_SendEvent(EVT_APP_RESUME), which restarts the suspended application.

This function simplifies the reactivation of any BREW application that was suspended by the user interface (UI).

Prototype:

```
boolean AEE_Resume(void)
```

Parameters:

None

Return Value:

TRUE if the event is successfully processed.

FALSE if the event is not processed.

Comments:

None

See Also:

AEE_ResumeEx()

AEE_Suspend()

Return to the List of functions

AEE_ResumeEx()

Description:

This function is provided for the OEM to use when developing custom objects that require inter-thread notifications. It posts the AEECallback in the BREW system resume queue.

This is the same mechanism used by the BREW standard SoundPlayer interface. In that case, a callback is issued from another thread, the resume callback is posted, and the API is then called back in the UI thread's context.

Prototype:

```
void AEE_ResumeEx(AEECallback * pcb, uint16 wFlags, void * pa)
```

Parameters:

pcb	Pointer to the callback
wFlags	Flags for the callback: AEE_RESUME_CB_SYS: This flag tells the BREW layer to associate the callback with the system rather than the currently active application. This is done to support callbacks that may need to be called when applications are not running or across applications. AEE_RESUME_CB_PRIO: This flag instructs BREW to place this callback ahead of other callbacks.
pa	Pointer to the application context.

Return Value:

None

Comments:

None

See Also:

AEE_Resume()

Return to the List of functions

AEE_SetAppContext()

Description:

This function sets the application context.

Prototype:

```
void * AEE_SetAppContext(void * pc)
```

Parameters:

pc Pointer to the new application context.

Return Value:

Pointer to the previous application context.

Comments:

None

See Also:

AEE_GetAppContext()

Return to the List of functions

AEE_SetEventHandler()

Description:

This function allows an OEM layer to create and use AEE controls from outside AEE applets. The callback is called whenever the control or AEE issues an event.

Prototype:

```
void AEE_SetEventHandler(void * pData, AEEHANDLER pfn)
```

Parameters:

pData	Private data that is passed as the first parameter to the callback.
pfn	Private callback function that is to be called by the AEE.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_SetSysTimer()

Description:

This function allows the OEM layer 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 will expire at Current Time + <Milliseconds specified>.
- Any normal processing can be done in the callback. This includes drawing to the screen, writing to files, and so on.
- Timers do not repeat. The OEMs must reset the timer if they want a repeating timer.
- Specifying the same callback/data pointers automatically overrides a pending timer with the same callback/data pointers.

Prototype:

```
int AEE_SetSysTimer(int32 nMSecs, PFNNOTIFY pfn, void * pUser)
```

Parameters:

nMSecs	Timer expiration in milliseconds. The expiration will occur at Current Time + dwMSecs.
pfn	The callback that will be called when the timer expires.
pUser	The data pointer that will be passed as the only parameter to the callback.

Return Value:

EBADPARM if an invalid time or callback is specified.

ENOMEMORY if memory allocation fails.

0 (zero) if successful.

Comments:

None

See Also:

PFNNOTIFY

Return to the List of functions

AEE_SocketEventOccurred()

Description:

This function is the network and socket callback function. It is called when there is any activity with the network and sockets. This callback function is invoked by the processor's network layer. The address is passed to it during the initialization of the network layer.

Prototype:

```
void AEE_SocketEventOccurred(void)
```

Parameters:

None

Return Value:

None

Comments:

None

Return to the List of functions

AEE_Suspend()

Description:

This function allows the OEM layer to suspend the active BREW application. It is equivalent to calling ISHELL_SendEvent(EVT_APP_SUSPEND). The application can be restarted by calling AEE_Resume().

This function simplifies the suspension of any AEE activity during times when the user interface (UI) is active.

Prototype:

```
void AEE_Suspend(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

AEE_Resume()

Return to the List of functions

AEE_TimerExpired()

Description:

This function adds an event to the event queue that will cause AEE_DISPATCH to be called.

Prototype:

```
void AEE_TimerExpired(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

AEE_UnlinkSysObject()

Description:

This function de-associates an AEESysObject with an application.

Prototype:

```
int AEE_UnlinkSysObject(AEESysObject *pso)
```

Parameters:

pso	the object to be associated with the application

Return Value:

SUCCESS, if the operaion was successful.

EALREADY, if pso->pac is NULL, i.e. not associated with an app

Comments:

None

Side Effects:

pso->pac is set to NULL.

See Also:

AEESysObject

Return to the List of functions

OEM Address Book Interface

This section describes the Address Book Interface functions that the AEE uses to implement the Address Book functionality. Prior to 2.0 release these were independent function. They have been converted into an interface

List of functions

Functions in this interface include:

OEMAddr_EnumNextRec()
OEMAddr_EnumRecInit()
OEMAddr_GetCatCount()
OEMAddr_GetCatList()
OEMAddr_GetFieldInfo()
OEMAddr_GetFieldInfoCount()
OEMAddr_GetNumRecs()
OEMAddr_RecordAdd()
OEMAddr_RecordDelete()
OEMAddr_RecordGetByID()
OEMAddr_RecordUpdate()
OEMAddr_RemoveAllRecs()
OEMAddrBook_CommonExit()
OEMAddrBook_CommonInit()
OEMAddrBook_Exit()
OEMAddrBook_Init()

The remainder of this section provides details for each function.

OEMAddr_EnumNextRec()

Description:

This function returns the information about the next record based on the search criteria specified in most recent call to OEMAddr_EnumRecInit().

Prototype:

```
uint16 OEMAddr_EnumNextRec
      (
      AEEAddrCat * cat,
      AEEAddrField ** ppItems,
      int * nItemCount,
      int * pErr
      )
```

Parameters:

cat	On return, if the next record was found, contains the address category of that next record.
ppItems	On return, if the next record was found, contains the list of address fields found in that next record.
nItemCount	On return, if the next record was found, contains the number of address fields found in that next record.
*pErr	On return, contains the error code if an error occurred.

Return Value:

The recordID if the next record is successfully found. This function also fills up the incoming parameters with the contents of the newly found record.

AEE_ADDR_RECID_NULL if the end of the enumeration has been reached or no more records are found. This value must be returned.

Comments:

When the end of the enumeration has been reached, the index must not be reset to point to the beginning of the enumeration. All subsequent calls to this function must continue to return AEE_ADDR_RECID_NULL. The caller must call OEMAddr_EnumRecInit() to re-initialize the search criteria.

See Also:

OEMAddr_EnumRecInit()

Return to the List of functions

OEMAddr_EnumRecInit()

Description:

This function searches the address book for specific records, and also sequentially retrieves all of the records in the database. The function initializes the enumeration of records in the address book based on a specific search criteria. When enumeration has been initialized, the function OEMAddr_EnumNextRec() is used to iterate through the records that match this search criteria.

Prototype:

```
int OEMAddr_EnumRecInit
      (
      AEEAddrCat wCategory,
      AEEAddrFieldID wFieldID,
      void * pData,
      uint16 wDataSize
      )
```

Parameters:

wCategory	Category type to be matched. If set to AEE_ADDR_CAT_NONE, it is ignored.
wFieldID	AEEAddrFieldID to be matched. If set to AEE_ADDRFIELD_NONE, it is ignored. Typically, OEMs do not allow searching for records on this field ID (for example, searching for records based on EMAIL may not be allowed). In this case, return EFAILED and IADDRBOOK_EnumNextFieldsInfo().
pData	If non-null, the actual data that must be matched. If NULL, it is ignored. For example, if **wFieldID** is set to AEE_ADDRFIELD_NAME, **pData** contains the actual name to be matched.
wDataSize	Size of **pData**.

Return Value:

AEE_SUCCESS if enumeration is successfully initialized.

EFAILED if fails.

Comments:

This function can also be used to enumerate all records in the database by specifying AEE_ADDR_CAT_NONE for the category parameter and AEE_ADDRFIELD_NONE for the field parameter.

See Also:

OEMAddr_EnumNextRec()

Return to the List of functions

OEMAddr_GetCatCount()

Description:

This function returns the number of address categories supported by the address book. Examples of address categories are PERSONAL and BUSINESS. Each record in the address book can belong to a specific address category. If the concept of categories are not supported in the address book, this function must return 0 (zero).

Prototype:

```
int OEMAddr_GetCatCount(void)
```

Parameters:

None

Return Value:

Number of categories supported.

Comments:

It is valid to return 0 (zero) from this function if the address book does not support the concept of categories for each record.

See Also:

None

Return to the List of functions

OEMAddr_GetCatList()

Description:

This function returns information about all of the address categories supported by the address book in the device. The function is called only if OEMAddr_GetCatCount() returned a value other than 0 (zero).

Prototype:

```
int OEMAddr_GetCatList(AEEAddrCat *p, int nSize)
```

Parameters:

p	Pointer allocated by the caller that can hold information about the address categories.
nSize	Number of AEEAddrCat elements that can fit into the array pointed to by **p**.

Return Value:

AEE_SUCCESS if successful. Even if **nSize** is less than the total number of categories supported, this function must return AEE_SUCCESS as long as **nSize** is greater than 0 (zero).

EFAILED if fails.

Comments:

The categories must be converted from the OEM list to the AEE values before returning. A list of pre-defined AEEAddressCategories is in AEEAddrBook.h. You can also add your own categories.

See Also:

None

Return to the List of functions

OEMAddr_GetFieldInfo()

Description:

This function returns detailed information about each field type supported for the given category. This function is typically called after the OEMAddr_GetFieldInfoCount() function.

Prototype:

```
int OEMAddr_GetGetFieldInfo
      (
      AEEAddrCat c,
      AEEAddrFieldInfo * pf,
      int nSize
      )
```

Parameters:

c	Address category for which the field information is to be returned.
pf	Pointer to an array of AEEAddrFieldInfo structures (allocated by the caller) where information is to be returned by this function.
nSize	Number of **AEEAddrFieldInfo** elements that can fit into the array pointed to by **pf**.

Return Value:

AEE_SUCCESS if successful. Even if **nSize** is less than the total number of categories supported, this function must return AEE_SUCCESS as long as **nSize** is greater than 0 (zero).

EFAILED if fails.

Comments:

The AEEAddrFieldInfo structure contains detailed information about the fields, such as FieldID, and the maximum number of fields of this ID supported in each record. Detailed information about this structure is in AEEAddrBook.h and the *BREW API Reference Guide*.

See Also:

None

Return to the List of functions

OEMAddr_GetFieldInfoCount()

Description:

This function returns the number of types of fields supported for the given category. If the concept of categories is not supported, the function may return the total number of types of fields supported for each record in the address book. Examples of fields are NAME, WORK_NUM, FAX_NUM, URL, and ADDRESS.

Prototype:

```
int OEMAddr_GetFieldInfoCount(AEEAddrCat c)
```

Parameters:

c Address category whose number of supported field types is to be returned.

Return Value:

Number of types of fields supported for the given address category,

0 (zero) if category is not supported.

Comments:

None

See Also:

OEMAddr_GetFieldInfo()

Return to the List of functions

OEMAddr_GetNumRecs()

Description:

This function returns the total number of records found in the address book.

Prototype:

```
uint16 OEMAddr_GetNumRecs(void)
```

Parameters:

None

Return Value:

Count of the total number of records currently found in the address book.

Comments:

None

See Also:

None

Return to the List of functions

OEMAddr_RecordAdd()

Description:

This function adds a new record to the address book. The fields to be added to this record are passed as parameters to this function.

Prototype:

```
uint16 OEMAddr_RecordAdd
      (
      AEEAddrCat cat,
      AEEAddrField * pItems,
      int nItemCount,
      int * pErr
      )
```

Parameters:

cat	Address category to which this record belongs. If the concept categories is not supported, this parameter can be ignored.
pItems	Pointer to an array of items that must be added to the record. Each item contains information such as FieldID, DataType, Data, and DataLength. **NOTE:** For detailed information about this structure, see the AEEAddrBook.h or *BREW API Reference Guide*.
nItemCount	Number of fields in this record. It also indicates that the array pItems contains this number of fields.
pErr	If an error occurs, the error code must be placed in this pointer before returning from this function. You must check for this parameter being NULL before storing the error value in it.

Return Value:

The recordID if successful. This function must return the record ID of the newly added record.

AEE_ADDR_RECID_NULL if fails. The parameter ***pErr** must contain the error code. A list of error codes is in AEEError.h. This value must be returned.

Comments:

None

See Also:

None

Return to the List of functions

OEMAddr_RecordDelete()

Description:

This function deletes a specified record from the address book.

Prototype:

```
int OEMAddr_RecordDelete(uint16 recID)
```

Parameters:

recID ID of the record to be deleted from the address book.

Return Value:

AEE_SUCCESS if record is successfuly deleted.

EFAILED if fails.

Comments:

When deleting a record while enumerating through the list of addressbook records, care should be taken that the next enumeration returns the correct record.

See Also:

None

Return to the List of functions

OEMAddr_RecordGetByID()

Description:

This function retrieves the information about a specified record, and returns information about all of the fields in this record. This function does not delete the record from the address book.

Prototype:

```
int OEMAddr_RecordGetByID
      (
      uint16 recID,
      AEEAddrCat * cat,
      AEEAddrField ** ppItems,
      int * nItemCount,
      int * pErr
      )
```

Parameters:

recID	ID of the record whose fields are to be retrieved and returned
cat	On input, this is a valid pointer to AEEAddrCat. On return, this pointer points to the address category to which the record belongs.
ppItems	Pointer for passing info about the fields. While implementing this function, the OEMs must allocate memory for ***ppItems** using the function MALLOC(). This memory is freed by the caller (BREW).
nItemCount	On input, this is a valid pointer to an integer. On return, this pointer contains the count of the number of fields present in this record, and indicates that the array *ppItems contains this number of fields on return.
pErr	If any error occurs, the error code must be placed into this pointer before returning from this function. You must check for this parameter being NULL before storing the error value in it.

Return Value:

AEE_SUCCESS if the record information is successfully retrieved.

EFAILED if fails. The parameter ***pErr** must contain the exact error code.

Comments:

Memory for ***ppItems** must be allocated by the implementer of this function. This memory is released by the caller.

See Also:

None

Return to the List of functions

OEMAddr_RecordUpdate()

Description:

This function updates all of the fields in the specified record. It replaces all existing fields in that record with the fields being passed to this function.

Prototype:

```
int OEMAddr_RecordUpdate
      (
      uint16 recID,
      AEEAddrCat cat,
      AEEAddrField * pItems,
      int nItemCount,
      int * pErr
      )
```

Parameters:

recID	ID of the record whose fields are to be updated.
cat	Address category to which this record belongs. If the concept of categories is not supported, this parameter can be ignored.
pItems	Pointer to an array of items that are to be added to the record. Each item contains information such as FieldID, DataType, Data, and DataLength. **NOTE:** For detailed information about this structure, see the AEEAddrBook.h or *BREW API Reference Guide*.
nItemCount	Number of fields in this record. It also indicates that the array pItems contains this number of fields.
pErr	If an error occurs, the error code must be placed in this pointer before returning from this function. You must check for this parameter being NULL before storing the error value in it.

Return Value:

AEE_SUCCESS if the record is successfully deleted.

EFAILED if fails.

Comments:

This function is similar to OEMAddr_RecordAdd(); the main difference is that this function updates all of the fields in an existing record, while OEMAddr_RecordAdd() adds a new record to the address book.

See Also:

OEMAddr_RecordAdd()

Return to the List of functions

OEMAddr_RemoveAllRecs()

Description:

This function deletes all records from the address book.

Prototype:

```
int OEMAddr_RemoveAllRecs(void)
```

Parameters:

None

Return Value:

AEE_SUCCESS if all the records are successfully deleted.

EFAILED if fails.

EUNSUPPORTED if this function is not supported.

Comments:

Since this is a sensitive operation, you can decide not to support it. If this function is not supported, the value EUNSUPPORTED must be returned from this function.

See Also:

None

Return to the List of functions

OEMAddrBook_CommonExit()

Description:

This function is called when the BREW AddressBook interface is deleted, allowing OEMs to free the OEM layer object.

Prototype:

```
IOEMAddrBook *OEMAddrBook_CommonExit
      (
      AEECLSID ClsId, IOEMAddrBook *pme
      )
```

Parameters:

ClsId	Used to designate the OEM layer media used for address book storage.
pme	pointer to the IOEMAddrBook object to be deleted.

Return Value:

This function returns NULL.

Comments:

If any memory was allocated during the call to OEMAddrBook_CommonInit() this function should be used to free it. This function can also be used for any media/device cleanup that needs to occur when the object is released.

See Also:

OEMAddrBook_CommonInit()

Return to the List of functions

OEMAddrBook_CommonInit()

Description:

This function is called when the BREW AddressBook interface is created. It allocates memory for the OEM address book object and populates the object with the appropriate OEM functions depending on the input class ID.

This allows OEMs to customize the AddressBook OEM layer for specific storage media.

Prototype:

```
IOEMAddrBook *OEMAddrBook_CommonInit(AEECLSID ClsId);
```

Parameters:

ClsId	Used to designate the OEM layer media used for address book storage.

Return Value:

Returns a pointer to the IOEMAddrBook object for any valid class ID.

Otherwise, returns NULL.

Comments:

OEMs can add support for new media/devices and class IDs by adding the new class ID to the switch statement in this function, allocating memory for the object, and initializing the object with the appropriate OEM functions.

See Also:

None

Return to the List of functions

OEMAddrBook_Exit()

Description:

This function is called when the BREW AddressBook interface is deleted, allowing the address book to be cleaned up.

Prototype:

```
void OEMAddrBook_Exit(void)
```

Parameters:

None

Return Value:

None

Comments:

If any memory has been allocated during the address book operation, this is the time to free it.

See Also:

None

Return to the List of functions

OEMAddrBook_Init()

Description:

This function is called when the BREW AddressBook interface is created, allowing the address book to be initialized.

Prototype:

```
boolean OEMAddrBook_Init(void)
```

Parameters:

None

Return Value:

TRUE if initialization is successful.

FALSE if initialization fails.

Comments:

If this function returns FALSE, BREW does not allow the IAddrBook interface to be created, and therefore does not allow access to the OEM Address Book.

See Also:

None

Return to the List of functions

OEM Application Interface

This section describes the Application Interface functions that are required by the AEE. Reference implementations of some of the functions are provided. See the *OEM Porting Guide* for details on reference implementations.

List of functions

Functions in this interface include:

OEM_AuthorizeDownload()
OEM_CheckPrivacy()
OEM_GetItemStyle()
OEM_LockMem()
OEM_Notify()
OEM_SimpleBeep()
OEM_UnlockMem()

The remainder of this section provides details for each function.

OEM_AuthorizeDownload()

Description:

This function is called before each download. A callback must be sent before the IDOWNLOAD_Acquire() operation is completed. The IDownload class instance and the item/price IDs are used to query information about the item. The IDownload pointer is used to set the HTTP headers sent to the ADS. This is performed with the CSetHeaders method, which uses the following parameters:

- pUser: User data to be passed to the callback.
- iID: Item ID.
- iPrice: Item Price Handle.
- pItem: Pointer to information structure for the item.
- nErr: Error code. If non-zero, the download aborts and this error is reported.

Prototype:

```
void OEM_AuthorizeDownload
      (
      IDownload * pd,
      DLITEMID iID,
      DLPRICEID iPrice,
      DLItem * pItem,
      PFNCHECKDLCB pfn,
      void * pUser
      )
```

Parameter(s):

pd	Pointer to the IDownload instance.
iID	Item ID.
iPrice	Item price ID.
pItem	Pointer to the information structure for the item.
pfn	Callback function.
pUser	User data to be passed to the callback.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_CheckPrivacy()

Description:

This function allows the OEM/Carrier to specify the correct action for various outbound requests. The OEM/Carrier can modify this function to conform to the requirements for the Carrier, allowing the OEM to display a message or prompt to the user, such as “Contact the network.”

Prototype:

```
void OEM_CheckPrivacy
      (
      OEMPrivacyReqType t,
      AEECallback *pCB,
      int *pStatus;
      )
```

Parameters:

t	Privacy request type (PRT_POSITION or PRT_DIAL_VOICE).
pCB	Pointer to AEECallback structure. Refer to AEECallback structure
pStatus	Return value 0 (zero) if successful EFAILED if invalid request type

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_GetItemStyle()

This function retrieves information regarding the drawing style for stock objects. BREW can use a default set of values if none are provided by this function. To use the default values, return FALSE to this function.

The default values used by BREW are:

```
bBigX = di.cxScreen >= 120;
bBigY = di.cyScreen >= 120;

switch (nColorDepth) {
case 1:
   pNormal->ft = AEE_FT_NONE;
   pSel->ft    = AEE_FT_NONE;

   pNormal->roImage = AEE_RO_COPY;
   pSel->roImage    = AEE_RO_NOT;
   break;

case 2:
   pNormal->ft = AEE_FT_EMPTY;
   pSel->ft    = AEE_FT_BOX;

   pNormal->roImage = AEE_RO_COPY;
   pSel->roImage    = AEE_RO_COPY;
   break;

default:
   if (bBigX && bBigY) {
      pNormal->ft = AEE_FT_3D_EMPTY;
      pSel->ft    = AEE_FT_RAISED;
   } else {
      pNormal->ft = AEE_FT_EMPTY;
      pSel->ft    = AEE_FT_BOX;
   }
   pNormal->roImage = AEE_RO_MASK;
   pSel->roImage    = AEE_RO_MASK;
   break;
}

pSel->xOffset = pNormal->xOffset = bBigX ? 3 : 1;
pSel->yOffset = pNormal->yOffset = bBigY ? 3 : 1;
```

Prototype:

```
boolean OEM_GetItemStyle
      (
      AEEItemType t,
      AEEItemStyle * pNormal,
      AEEItemStyle * pSel
      )
```

Parameters:

t	Item type.

`pNormal`	Normal style
`pSel`	Selected style

Return Value:

None

Comments:

Return FALSE for BREW to pick the default values.

See Also:

None

Return to the List of functions

OEM_LockMem()

Description:

NOTE: This function should not be used without clearly understanding all rules associated with handled-based memory.

By default, all memory allocated via the BREW heap is locked and cannot be relocated. This function provides a means for the caller to specify that a recently allocated and unlocked block is now locked and cannot be moved.

Initial Allocation:

```
pme->m_ptr = OEM_Malloc(100);
OEM_UnlockMem(&pme->m_ptr);
```

Use:

```
OEM_LockMem(&pme->m_ptr);
```

Perform Work...

```
OEM_UnlockMem(&pme->m_ptr);
```

As indicated, this function accepts a pointer to a handle of allocated memory. The function will then mark the block as unavailable to be moved in memory. This function also increments a "lock count" on the memory block. This allows the caller to pair OEM_LockMem and OEM_UnlockMem inside sub-routines that may be called while the memory is locked. In such cases, the memory will not actually be "unlocked" until the reference count reaches 0.

Prototype:

int OEM_LockMem(void ** ppHandle);

Parameters:

None.

Return Value:

0 - Incremented lock count of the memory

0 - Invalid ppHandle

Comments:

None

See Also:

OEM_UnlockMem()

Return to the List of functions

OEM_Notify()

Description:

This function is called by BREW to notify the OEM about critical system functions. These situations include the following:

OEMNTF_APP_START

A query to determine if the specified application can be started. The class ID of the application is specified in the **dwParam**. The OEM should not refuse to start apps unless they are in very critical conditions on the device (i.e. incoming call, etc.). In such cases, there is no automatic way to start this app at a later time.

OEMNTF_ACTIVATE

Notification to the OEM that the BREW layer is activating an application. Subsequent application start notifications that occur while a previous BREW application is running will not be preceded by this notification.

OEMNTF_IDLE

Notification to the OEM that the BREW layer is closing the last active application in the stack of currently active BREW applications. This situation does not mean that all BREW applications are closed, but rather that control is returning to the native UI.

OEMNTF_RESET

A request by BREW to reset the device.

OEMNTF_CLOSE_FILE

BREW is requesting that a file be closed.

OEMNTF_SHOW_CALL_DIALOGS

Indicates whether the OEM should show call dialogs. If **dwParam** is TRUE, the OEM should show these dialogs. If it is FALSE, the OEM should not show them. This call is typically made only by the network layer to inhibit/re-enable the display of call status dialogs during PPP sessions.

OEMNTF_GET_CONTEXT

This call is made by BREW when the OEM starts a BREW app. This notification is required because in these cases OEMNTF_ACTIVATE is not called. The dwParam is of type OEMAppState.

OEMNTF_APP_EVENT

This notification is sent to the OEM layer for any app event <= EVT_APP_LAST_EVENT. This can be used by OEMs to track when an app is being started, suspended, resumed or stopped. The passed structure contains all of the elements of the event passed to the app.

NOTE: No OEM modification of the parameters is supported. The OEM should NOT alter the parameters. The app context of the target app has been asserted when this notification is made. Any calls to BREW will appear to come from the app. Moreover, access to system functions, etc. will be limited based upon the rights of the app. The dwParam is of type OEMAppEvent. This structure is as follows:

```
typedef struct _OEMAppEvent
      {
         AEECLSID cls: ClassID of the app to which the event is being
      sent.
         AEEEvent evt: Event Code of the event being sent to the app
         uint16   wp:  wParam of the event being sent to the app
         uint32   dwp: dwParam of the event being sent to the app.
      } OEMAppEvent;
```

Prototype:

```
int OEM_Notify(OEMNotifyEvent evt, uint32 dwParam)
```

Parameters:

evt	The event code.
dwParam	Context sensitive data.

Return Value:

0 (zero) if successful.

Comments:

None

See Also:

None

Return to the List of functions

OEM_SimpleBeep()

Description:

This function plays a standard OEM beep tone and vibration given the duration, and returns TRUE if successful.

Prototype:

```
boolean OEM_SimpleBeep(BeepType nBeepType, boolean bLoud)
```

Parameters:

nBeepType	Beep type that can be one of the following: BEEP_OFF: stop playback of the current beep or vibration BEEP_ALERT: alert beep tone BEEP_REMINDER: reminder beep tone BEEP_MSG: message beep tone BEEP_ERROR: error beep tone BEEP_VIBRATE_ALERT: alert vibration BEEP_VIBRATE_REMIND: reminder vibration
bLoud	Boolean flag that sets the playback volume high or low.

Return Value:

TRUE if successfuly played or stopped the tone or vibration.

FALSE otherwise.

Comments:

None

See Also:

None

Return to the List of functions

OEM_UnlockMem()

Description:

NOTE: This function should not be used without clearly understanding all rules associated with handled-based memory.

By default, all memory allocated via the BREW heap is locked and cannot be relocated. This function provides a means for the caller to specify that a recently allocated memory block can be moved. It is called as follows:

Initial Allocation:

```
pme->m_ptr = OEM_Malloc(100);
OEM_UnlockMem(&pme->m_ptr);
```

Use:

```
OEM_LockMem(&pme->m_ptr);
```

Perform Work...

```
OEM_UnlockMem(&pme->m_ptr);
```

As indicated, this function accepts a pointer to a handle of allocated memory. The function will then mark the block as available to be moved in memory. This function monitors a "lock count" on the memory block. This allows the caller to pair OEM_LockMem and OEM_UnlockMem inside sub-routines that may be called while the memory is locked. In such cases, the memory will not actually be "unlocked" until the reference count reaches 0.

Prototype:

```
int OEM_UnlockMem(void ** ppHandle);
```

Parameters:

None.

Return Value:

0 - Decremented lock count of the memory

Negative Values:

EALREADY - Memory is already unlocked and associated with another sentinal.

EMEMPTR - Invalid ppHandle

EFAILED - Association is invalid. Breakpoint thrown on SDK simulator.

Comments:

None

See Also:

OEM_LockMem()

Return to the List of functions

OEMBTSDP Interface

This module specifies the functions required for BT SDP support in BREW. There are no requirements for supporting calls from other DMSS tasks. It is assumed that only BREW will be making calls to this interface.

List of functions

Functions in this interface include:

OEMBTSDP_CancelDiscovery()
OEMBTSDP_CloseLib()
OEMBTSDP_DiscoverDevices()
OEMBTSDP_GetDeviceName()
OEMBTSDP_GetServerChannel()
OEMBTSDP_Init()
OEMBTSDP_OpenLib()
OEMBTSDP_Shutdown()

The remainder of this section provides details for each function.

OEMBTSDP_CancelDiscovery()

Description:

This function cancels the outstanding discovery request. The OEM layer should not generate any more events related to device discovery.

Prototype:

```
void OEMBTSDP_CancelDiscovery(int32 libhandle)
```

Parameters:

libhandle	Library handle for the BT SDP instance.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMBTSDP_CloseLib()

Description:

This function can be called by BREW at some point after OEMBTSDP_OpenLib() to close a BT SDP library instance

Prototype:

```
extern void OEMBTSDP_CloseLib(int32 libhandle)
```

Parameters:

libhandle	Valid library handle greater than 0 (zero).

Return Value:

None

Comments:

The OEM should not send any events after the close to the Using the Notification function for this instance.

See Also:

OEMBTSDP_OpenLib()

Return to the List of functions

OEMBTSDP_DiscoverDevices()

Description:

This functions tells the OEM layer to start the discovery process. It should generate events on getting the response from the neighboring devices.

Prototype:

```
int32 OEMBTSDP_DiscoverDevices
      (
      int32 libhandle,
      OEMBT_service_class_enum_type svcclass,
      int32 maxResps
      )
```

Parameters:

libhandle	Library Handle for the BT SDP Instance.
svcclass	Service class of the BT device.
maxResps	Max number of responses accepted.

Return Value:

OEMBTSDP_DISCOVERY_INPROGRESS if discovery is already in progress.

0 if successful.

Comments:

None

See Also:

None

Return to the List of functions

OEMBTSDP_GetDeviceName()

Description:

This function requests the OEM layer to send the device name request to the specified device.

Prototype:

```
extern int32 OEMBTSDP_GetDeviceName
      (
      int32 libhandle,
      OEMBT_Addr * bdaddr
      )
```

Parameters:

libhandle	Library handle for the BT SDP instance.
bdaddr	BT device address.

Return Value:

OEMBTSDP_DISCOVERY_INPROGRESS if a discovery is already in progress.

0 if successful.

-1 if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEMBTSDP_GetServerChannel()

Description:

This function requests the OEM layer to send server channel request to the specified device and service class

Prototype:

```
int32 OEMBTSDP_GetServerChannel
      (
      int32 libhandle,
      OEMBT_service_class_enum_type service_class
      )
```

Parameters:

libhandle	Library handle for the BT SDP instance.
bdaddr	BT device address.
service_class	Service class of the device.

Return Value:

OEMBTSDP_DISCOVERY_INPROGRESS if a discovery is already in progress.

0 if successful.

-1 if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEMBTSDP_Init()

Description:

This function is called by BREW to indicate that it is ready to accept notifications, and to specify the function to be called to deliver notifications. It should return 1. Non-functional stubs for this API return 0 to indicate that BT I/O is unsupported.

The OEM BT layer uses the function pointer passed by BREW to notify BREW of events. When this notification function is called, BREW is responsible for taking care of thread safety issues.

Prototype:

```
int16 OEMSDP_Init(PFNBTSDPNOTIFY pfnNotify)
```

Parameters:

pfnNotify	Function to call with notifications.

Return Value:

1 if successful (BT SDP is supported).

0 if fails (BT SDP unsupported).

Comments:

None

See Also:

OEMBTSDP_Shutdown()

Return to the List of functions

OEMBTSDP_OpenLib()

Description:

This function can be called by BREW at some point after OEMBTSDP_Init() to open a BT SDP Library Instance.

Prototype:

```
extern int32 OEMBTSDP_OpenLib()
```

Parameters:

None

Return Value:

Valid library handle greater than 0 (zero).

<0 in case of error.

Comments:

None

See Also:

OEMBTSDP_CloseLib()

Return to the List of functions

OEMBTSDP_Shutdown()

Description:

This function can be called by BREW at some point after OEMBTSIO_Init() to indicate that the previously-registered notify callback should not be called anymore.

Prototype:

```
void OEMBTSDP_Shutdown(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

OEMBTSDP_Init()

Return to the List of functions

OEMBTSIO Interface

This section describes the functions required for BT IO support in BREW. There are no requirements for supporting calls from other DMSS tasks. It is assumed that only BREW will be making calls to this interface.

This interface currently describes one mode of interaction with one type of device: applet Initiated.

List of functions

Functions in this interface include:

OEMBTSIO_Close()
OEMBTSIO_DataAvailable()
OEMBTSIO_Init()
OEMBTSIO_Open()
OEMBTSIO_ProcessEvents()
OEMBTSIO_Read()
OEMBTSIO_Write()

The remainder of this section provides details for each function.

OEMBTSIO_Close()

Description:

This function can be called by BREW at some point after OEMBTSIO_Open() to close the specified port.

Prototype:

```
int32 OEMBTSIO_Close(int32 portHandle)
```

Parameters:

portHandle	Returned by OEMBTSIO_Open().

Return Value:

0(zero) if successful.

OEMBTSIO_INVALID_HANDLE if it is an invalid handle.

Comments:

None

See Also:

OEMBTSIO_Open()

Return to the List of functions

OEMBTSIO_DataAvailable()

Description:

This function determines if there is data available to be read from the BT port.

Prototype:

```
int32 OEMBTSIO_DataAvailable(uint32 portHandle)
```

Parameters:

portHandle Returned by OEMBTSIO_Open().

Return Value:

1 if data is available

0 (zero) if no data is available.

Comments:

None

See Also:

OEMBTSIO_Open()

Return to the List of functions

OEMBTSIO_Init()

Description:

This function is called by BREW to indicate that it is ready to accept notifications, and to specify the function to be called to deliver notifications. It should return 1. Non-functional stubs for this API return 0 to indicate that BT IO is unsupported.

The function pointer passed by BREW should be used by the OEM BT layer to notify BREW of events. When this notification function is called, BREW is responsible for taking care of thread safety issues.

Prototype:

```
int32 OEMBTSIO_Init(PFNBTSIONOTIFY pfnNotify)
```

Parameters:

pfnNotify	Function to call with notifications.

Return Value:

1 if successful (BT IO is supported).

0 (zero) if fails (BT IO is unsupported).

Comments:

None

See Also:

OEMBTSIO_Close()

Return to the List of functions

OEMBTSIO_Open()

Description:

This function can be called by BREW at some point after OEMBTSIO_Init() to open a BT Port This will not otherwise affect the state of the BT driver or the receive and transmit buffers.

Prototype:

```
int32 OEMBTSIO_Open(OEMBTConnectionInfo * pCntInfo)
```

Parameters:

pCntInfo	Pointer to the connection information.

Return Value:

Porthandle identified by an integer.

OEMBTSIO_INVALID_HANDLE if fails.

Comments:

None

See Also:

OEMBTSIO_Close()

Return to the List of functions

OEMBTSIO_ProcessEvents()

Description:

This function processes the events related to Bluetooth Stack.

Prototype:

```
void OEMBTSIO_ProcessEvents()
```

Parameters:

None

Return Value:

None

Comments:

This function should be called in response to the Wakeup Callback

See Also:

OEMBTSIO_Init()

Return to the List of functions

OEMBTSIO_Read()

Description:

This function copies data from the receive buffer to the **pcBuf[ ]** buffer supplied by the caller. As much data as is available should be copied, never exceeding **nLen** bytes and never exceeding the number of bytes available in the receive buffer. Data copied to **pcBuf[ ]** should be removed from the receive buffer.

Prototype:

```
int32 OEMBTSIO_Read(char * pcBuf, int32 nLen, uint32 nPort)
```

Parameters:

pcBuf	Pointer to the character buffer.
nLen	Length of the buffer in bytes.
nPort	Port handle returned by OEMBTSIO_Open().

Return Value:

The number of bytes removed from the receive buffer.

If the receive buffer is empty, no bytes should be copied and this function should return 0 (zero).

If the port is in error, -1 should be returned.

Comments:

None

See Also:

None

Return to the List of functions

OEMBTSIO_Write()

Description:

This function should append **nLen** bytes from the specified buffer (**pcBuf**) to the end of the transmit buffer. If the buffer cannot accomodate **nLen** bytes, as many bytes as can fit should be appended.

Prototype:

```
int32 OEMBTSIO_Write(const char * pcBuf, int32 nLen, uint32 nPort)
```

Parameters:

pcBuf	Pointer to the character buffer.
nLen	Length of the buffer in bytes.
nPort	Port handle returned by OEMBTSIO_Open().

Return Value:

The number of bytes appended to the transmit buffer.

If the receive buffer is full, no bytes should be copied and this function should return 0 (zero).

If the port is error, -1 should be returned.

Comments:

None

See Also:

None

Return to the List of functions

OEM Configuration Interface

This section describes the Configuration Interface functions that are required by the AEE.

List of functions

Functions in this interface include:

OEM_GetAddrBookPath()
OEM_GetAppPath()
OEM_GetConfig()
OEM_GetDeviceInfo()
OEM_GetDeviceInfoEx()
OEM_GetLogoPath()
OEM_GetMIFPath()
OEM_GetPath()
OEM_GetRingerPath()
OEM_GetSharedPath()
OEM_SetConfig()

The remainder of this section provides details for each function.

OEM_GetAddrBookPath()

Description:

This function is called by the AEE to determine the path for the address book application and files. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetAddrBookPath(void)
```

Parameters:

None

Return Value:

The **const char** pointer to the path.

Comments:

None

See Also:

OEM_GetPath()

Return to the List of functions

OEM_GetAppPath()

Description:

This function is called by the AEE to determine the path for the applet directories. The path is the directory in which the AppManager looks for the applets. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetAppPath(void)
```

Parameters:

None

Return Value:

The **const char *** pointer to the path.

Comments:

None

See Also:

OEM_GetPath()

Return to the List of functions

OEM_GetConfig()

Description:

This function retrieves the device configuration information related to the download services.

Prototype:

```
int OEM_GetConfig(AEEConfigItem i, void * pBuff, int nSize)
```

Parameters:

i	Item that needs to be retrieved. It can be one of the Configuation Parameters
pBuff	Buffer in which the new values are stored.
nSize	Size of the buffer.

See the Configuation Parameters list for the values of the **i** parameter:

Return Value:

0 (zero) if successful.

EBADPARM if size or parameter is not valid

EUNSUPPORTED if the config item is not supported

Other implementation-specific error codes

Comments:

None

See Also:

OEM_SetConfig()

Configuation Parameters

Return to the List of functions

OEM_GetDeviceInfo()

Description:

This function retrieves the current device's physical and hardware characteristics.

Prototype:

```
void OEM_GetDeviceInfo(AEEDeviceInfo * pi)
```

Parameters:

pi Retrieved buffer where AEEDeviceInfo is stored. See the *BREW API Reference Guide*.

Return Value:

None

Comments:

None

See Also:

AEEDeviceInfo

Return to the List of functions

OEM_GetDeviceInfoEx()

Description:

This method is used to get specific information about the device. This function takes an ID that specifies what information is needed. The buffer contains the corresponding information on return.

Prototype:

```
int OEM_GetDeviceInfoEx
      (
      AEEDeviceItem nItem,
      void *pBuff,
      int *pnSize
      );
```

Parameters:

nItem	Specifies the Item whose info is needed. Please see documentation of AEEDeviceItem for the supported Device Items.
pBuff	Contains the corresponding information on return
pnSize	Contains the size of pBuff. On return, it contains the size filled. This parameter maybe NULL for certain device Items.

Return Value:

SUCCESS, if successful

EBADPARM, if bad parameters are passed in

EUNSUPPORTED, if this item is not supported

Comments:

None

See Also:

AEEDeviceItem

Return to the List of functions

OEM_GetLogoPath()

Description:

This function is called by the AEE to determine the path for the LOGO files. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetLogoPath(void)
```

Parameters:

None

Return Value:

The **const char** pointer to the path.

Comments:

None

See Also:

OEM_GetPath()

Return to the List of functions

OEM_GetMIFPath()

Description:

This function is called by the AEE to determine the path for the MIFs (Module Information Files). The path is the directory in which the AppManager looks for MIFs. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetMIFPath(void)
```

Parameters:

None

Return Value:

Returns **const char** pointer to path

Comments:

None

See Also:

OEM_GetPath()

Return to the List of functions

OEM_GetPath()

Description:

This function is called by the AEE to determine the path for the directory names used by BREW. Following are the directories:

- APPS: the directory where all the MIF files and applet directories are stored.
- SHARED: the directory where all the shared files are stored.
- RINGER: the directory where all the ringer files are stored.
- ADDRBOOK: the directory where all of the address book files are stored.
- MIF: the directory where all of the MIF files are stored.
- LOGO: the directory where all of the LOGO files are stored.

The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetPath(uint16 wType)
```

Parameters:

wType	Any of the six directories used by the AEE: APPS, ADDRBOOK, MIF, RINGER, SHARED, or LOGO.

Return Value:

The **const char *** pointer to the path.

Comments:

None

See Also:

OEM_GetAddrBookPath()

OEM_GetAppPath()

OEM_GetLogoPath()

OEM_GetMIFPath()

OEM_GetRingerPath()

OEM_GetSharedPath()

Return to the List of functions

OEM_GetRingerPath()

Description:

This function is called by the AEE to determine the path for the ringer files. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetRingerPath(void)
```

Parameters:

None

Return Value:

The **const char *** pointer to the path.

Comments:

None

See Also:

None

Return to the List of functions

OEM_GetSharedPath()

Description:

This function is called by the AEE to determine the path for the shared directory, which all applets with the shared directory access privilege share. The path must not have a trailing directory separation character.

Prototype:

```
const char * OEM_GetSharedPath(void)
```

Parameters:

None

Return Value:

The **const char *** pointer to the path.

Comments:

None

See Also:

None

Return to the List of functions

OEM_SetConfig()

Description:

It sets new handset configuration information related to the download services.

Prototype:

```
int OEM_SetConfig(AEEConfigItem i, void * pBuff, int nSize);
```

Parameters:

i	Item that needs to be retrieved. It can be one of the Configuation Parameters
pBuff	Buffer in which the new values are stored.
nSize	Size of the buffer.

See the Configuation Parameters list for the values of the **i** parameter:

Return Value:

0 (zero) if successful.

Comments:

None

See Also:

OEM_GetConfig()

Configuation Parameters

Return to the List of functions

OEM Cyclic Redundancy Check Interface

This section describes the function for the Cyclic Redundancy Check (CRC) Interface.

List of functions

Functions in this interface include:

OEMCRC_16_step()

The remainder of this section provides details for each function.

OEMCRC_16_step()

Description:

This function calculates a step-by-step 16-bit CRC over the specified disjunct data. This is more commonly referred to as CCITT-16. Use it to produce a CRC and check a CRC. The CRC value passed in is used to continue the CRC calculation from a previous call, allowing this routine to be used to CRC discontinuous data.

Prototype:

```
uint16 OEMCRC_16_step(uint16 seed, uint8 * buf_ptr, uint16 num_bytes)
```

Parameters:

seed	Initial state of the accumulation register.
buf_ptr	Pointer to the buffer containing the bytes to the CRC.
num_bytes	Number of bytes in the buffer to calculate the CRC over.

Return Value:

The calculated CCITT-16 CRC.

Comments:

None

See Also:

None

Return to the List of functions

OEM Database Interface

This section describes the Database Interface functions that the AEE database interface uses. OEMs may use the reference implementation provided, or modify and reimplement this module. The modified/reimplemented functions must comply with this interface specification.

List of functions

Functions in this interface include:

OEM_DBClose()
OEM_DBCreate()
OEM_DBDelete()
OEM_DBFree()
OEM_DBInit()
OEM_DBMakeReadOnly()
OEM_DBOpen()
OEM_DBRecordAdd()
OEM_DBRecordCount()
OEM_DBRecordDelete()
OEM_DBRecordGet()
OEM_DBRecordNext()
OEM_DBRecordUpdate()

The remainder of this section provides details for each function.

OEM_DBClose()

Description:

This function closes the database specified by **pDBContext**. As part of closing, the memory associated with this database is freed.

Prototype:

```
void OEM_DBClose(OEMCONTEXT pDBContext, AEE_DBError * pDBErr)
```

Parameters:

pDBContext	Handle of the database to be closed.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

None

Comments:

None

See Also:

OEM_DBOpen()

Return to the List of functions

OEM_DBCreate()

Description:

This function creates a new database and returns a handle to the created database. If **wMinRecCount** and **wMinRecSize** are specified, the function also reserves memory for this database.

Prototype:

```
OEMCONTEXT OEM_DBCreate
      (
      const char * szDBName,
      word wMinRecCount,
      word wMinRecSize,
      AEE_DBError * pDBErr
      )
```

Parameters:

szDBName	Name of the database to be created.
wMinRecCount	Minimum number of records that this database must support. It is used in conjunction with **wMinRecSize** to determine the total memory to be reserved for the database. If this parameter is 0 (zero), it is ignored.
wMinRecSize	Minimum size of the record. This is used in conjunction with the **wminRecCount** to determine the total memory to be reserved for the database. If this parameter is 0 (zero), it is ignored.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

OEMCONTEXT if successful, this is used as a handle to the newly created database.

NULL if fails. If **pDBErr** is non-null, it contains the actual error code on return.

Comments:

If **szDBName** is NULL or an empty string, set the error code to AEE_DB_ERR_NOT_EXIST and return NULL.

See Also:

OEM_DBOpen()

OEM_DBDelete()

Return to the List of functions

OEM_DBDelete()

Description:

This function deletes the database specified by **szDBName**.

Prototype:

```
void OEM_DBdelete(const char * szDBName, AEE_DBError * pDBErr)
```

Parameters:

szDBName	Name of the database to be deleted
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

None

Comments:

If **szDBName** is NULL or an empty string, set the error code to AEE_DB_ERR_NOT_EXIST and return NULL.

See Also:

OEM_DBCreate()

Return to the List of functions

OEM_DBFree()

Description:

This function frees the memory allocated for the buffer returned by OEM_DBRecordGet.

Prototype:

```
void OEM_DBFree
      (
      OEMCONTEXT pDBContext,
      byte * pbyRecBuf,
      AEE_DBError * pDBErr
      )
```

Parameters:

pDBContext	Handle of the database.
pbyRecBuf	Pointer to the previously allocated record buffer that is to be freed.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_DBInit()

Description:

This function initializes the database subsystem.

Prototype:

int OEM_DBInit(void)

Parameter(s):

None

Return Value:

If initialization was successfully completed, returns zero.

In all other cases, returns any non-zero value.

Comments:

Routine currently does nothing, but is called by OEM_Init

See Also:

None

Return to the List of functions

OEM_DBMakeReadOnly()

Description:

This function makes the specified database file read only. The contents of the index file are appended to the end of the database, and the database is marked as read only. No add, update, or delete operations are subsequently allowed on the database.

Prototype:

```
void OEM_DBMakeReadOnly(const char * szDBName, AEE_DBError * pDBErr)
```

Parameters:

szDBName	Name of the database to be made read only.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

None

Comments:

If **szDBName** is NULL or an empty string, set the error code to AEE_DB_ERR_NOT_EXIST and return NULL.

See Also:

None

Return to the List of functions

OEM_DBOpen()

Description:

This function opens a database and returns a handle to the opened database. If the database does not exist, the function returns NULL. It does not create the database.

Prototype:

```
OEMCONTEXT OEM_DBOpen(const char * szDBName, AEE_DBError * pDBErr)
```

Parameters:

szDBName	Name of the database to be opened
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

OEMCONTEXT if successful, this is used as the handle to the opened database.

NULL if fails. If pDBErr is non-null, the actual error code is placed inside pDBErr.

Comments:

The database **szDBName** must exist. Use OEM_DBCreate() to create a database and open it.

If **szDBName** is NULL or an empty string, set the error code to AEE_DB_ERR_NOT_EXIST and return NULL.

See Also:

OEM_DBCreate()

OEM_DBClose()

Return to the List of functions

OEM_DBRecordAdd()

Description:

This function adds a new record to the specified database. It returns the ID of the newly added record.

Prototype:

```
int OEM_DBRecordAdd
      (
      OEMCONTEXT pdb,
      const byte * pbBuf,
      word wBufSize,
      AEE_DBError * pDBErr
      )
```

Parameters:

pdb	Database handle.
pbBuf	Pointer to the buffer containing data to be stored in the new record.
wBufSize	Size of the buffer pointed to by pbBuf.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

ID of the newly added record, if successful.

OEM_REC_ID_NULL if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_DBRecordCount()

Description:

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

Prototype:

```
word OEM_DBRecordCount(OEMCONTEXT pDBContext, AEE_DBError * pDBErr)
```

Parameters:

pDBContext	Handle of the database whose record count is required.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

Number of records in the database, if successful.

0 (zero) if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_DBRecordDelete()

Description:

This function deletes the specified record from the database.

Prototype:

```
void OEM_DBRecordDelete
      (
      OEMCONTEXT pDBContext,
      word wRecId,
      AEE_DBError * pDBErr
      )
```

Parameters:

pDBContext	Handle of the database.
wRecId	ID of the record.
pDBErr	Place holder to contain error code information on return. If NULL no error code is returned.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_DBRecordGet()

Description:

This function retrieves a specified record from the database. For the given record ID (**wRecId**), the function retrieves the record information and the data associated with that record. The record information is returned by the parameter **pRecInfo** passed to this function. The data associated with that record is returned as a **byte***. The memory for the byte* is allocated from the heap. It is the caller's responsibility to free the memory later.

The function does not remove the record from the database. It returns a copy of the information stored in that record.

Prototype:

```
byte* OEM_DBRecordGet
      (
      OEMCONTEXT pDBContext,
      word wRecId,
      AEE_DBRecInfo * pRecInfo,
      AEE_DBError * pDBErr
      )
```

Parameters:

pDBContext	Handle of the database whose record is to be retrieved.
wRecId	ID of the record to be retrieved from the database.
pRecInfo	Information about the retrieved record is returned to the caller using the AEE_DBRecInfo structure.
pDBErr	Place holder to contain error code information on return. If NULL no error code is returned.

Return Value:

A pointer to the data stored in the specified record, if successful.

NULL if fails.

Comments:

The record returned is a copy of the actual data stored. The memory for this copy is allocated from the heap. A subsequent get call frees this memory and reallocates the requisite amount from the heap for that operation. Therefore, after each get call, the caller must copy the contents of the buffer returned to their own buffer and call OEM_DBFree().

See Also:

None

Return to the List of functions

OEM_DBRecordNext()

Description:

This function retrieves the ID of the record next to the given record ID. If the given record ID is OEM_DB_RECID_NULL, the function returns the ID of the first record in the given database. It returns OEM_DB_RECID_NULL if **wCurRecId** is the maximum record ID in the database.

Prototype:

```
word OEM_DBRecordNext
      (
      OEMCONTEXT pDBContext,
      word wCurRecid,
      AEE_DBError * pDBErr
      )
```

Parameters:

pDBContext	Handle of the database
wCurRecId	Specifies the ID of the current record. The record next to this record is retrieved from the database.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

ID of the record next to the given record ID, if successful.

OEM_DB_RECID_NULL if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_DBRecordUpdate()

Description:

This function updates the contents of the given record ID.

Prototype:

```
void OEM_DBRecordUpdate
      (
      OEMCONTEXT pdb,
      word wRecId,
      const byte * pbBuf,
      word wBufSize,
      AEE_DBError * pDBErr
      )
```

Parameters:

pdb	Database handle.
wRecId	ID of the record whose contents are to be updated.
pbBuf	Pointer to the buffer containing updated data to be stored in the given record.
wBufSize	Size of the buffer pointed to by pbBuf.
pDBErr	Place holder to contain error code information on return. If NULL, no error code is returned.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM Debug Interface

This section describes the functions in the Debug Interface.

List of functions

Functions in this interface include:

OEMDebug_Printf()
OEMDebug_VPrintf()

The remainder of this section provides details for each function.

OEMDebug_Printf()

Description:

This function prints text through a debug mechanism. It is very similar to printf(), but returns nothing.

Prototype:

```
void OEMDebug_Printf(const char * pszFormat, ...)
```

Parameter(s):

pszFormat	A printf-like format string.
...	Arguments for printf.

Return Value:

None

Comments:

This function uses MSG_FATAL macros to interface with DMSS debug messages. Unfortunately, the MSG macros do not copy the data and do not give an indication when they are done with it.

The function keeps DBG_NUM_STORES data buffers. If messages are dropped, garbled, or mixed together, increase the number of buffers.

Side Effects:

Sends messages through msg_put.

See Also:

OEMDebug_VPrintf()

Return to the List of functions

OEMDebug_VPrintf()

Description:

This function prints text through a debug mechanism. It is very similar to printf(), but returns nothing.

Prototype:

```
void OEMDebug_VPrintf(const char * pszFormat, va_list ap)
```

Parameter(s):

pszFormat	A printf-like format string.
ap	List of arguments.

Return Value:

None

Comments:

This function uses OEMDebug_Printf() to print a string. If the string is too long, memory corruption will occur.

See Also:

OEMDebug_Printf()

Return to the List of functions

OEM Display Interface

NOTE: This module provides all the basic display routines that AEE files use. OEMs must either implement or replace the implementation of these functions.

This section describes the basic Display Interface functions that the AEE files use.

List of functions

Functions in this interface include:

IOEMDISP_Backlight()
IOEMDISP_GetDefaultColor()
IOEMDISP_GetDeviceBitmap()
IOEMDISP_GetPaletteEntry()
IOEMDISP_GetSymbol()
IOEMDISP_GetSystemFont()
IOEMDISP_MapPalette()
OEMDisp_New()
IOEMDISP_SetAnnunciators()
IOEMDISP_SetPaletteEntry()
IOEMDISP_Update()

The remainder of this section provides details for each function.

IOEMDISP_Backlight()

Description:

The function turns the device backlight on or off.

Prototype:

```
int IOEMDISP_Backlight(IOEMDisp *pMe, boolean bOn)
```

Parameters:

pMe	Pointer to the OEM Display Interfacel.
bOn	Flag that determines whether to turn the backlight on or off.

Return Value:

SUCCESS is returned if the function performed correctly.

EUNSUPPORTED if the operation is not supported

Other implementation-specific error codes

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_GetDefaultColor()

Description:

This function is used to query the default system colors.

Prototype:

```
int IOEMDISP_GetDefaultColor
      (
      IOEMDisp *pMe,
      AEEClrItem clr,
      RGBVAL *pRGB
      )
```

Parameters:

pMe	[in]	Pointer to the OEM Display Interface.
clr	[in]	Item for which to obtain color.
pRGB	[out]	RGB value of the corresponding color.

Return Value:

SUCCESS is returned if the function performed correctly.

Other implementation-specific error codes

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_GetDeviceBitmap()

Description:

This function retrieves an interface to the device (screen) bitmap.

Prototype:

```
int IOEMDISP_GetDeviceBitmap(IOEMDisp * pMe, IBitmap ** ppIBitmap)
```

Parameters:

pMe	[in]	Pointer to the OEM Display Interface.
ppIBitmap	[out]	Pointer to the interface of device bitmap.

Return Value:

SUCCESS if the function performed correctly.

ENOMEMORY if there was not enough memory for the operation.

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_GetPaletteEntry()

Description:

This function gets an entry in the device's palette table. If the device does not support a dynamic palette, the function returns EUNSUPPORTED.

Prototype:

```
int IOEMDisp_GetPaletteEntry
      (
      IOEMDisp *pMe,
      RGBVAL *pRGB,
      unsigned int index
      )
```

Parameters:

pMe	[in]	Pointer to the OEM Display Interface.
pRGB	[out]	Value of the palette entry.
index	[in]	Index of palette entry to retrieve.

Return Value:

SUCCESS if the function performed correctly.

EUNSUPPORTED if the device does not have a dynamic palette.

Comments:

None

See Also:

IOEMDISP_SetPaletteEntry()

IOEMDISP_MapPalette()

Return to the List of functions

IOEMDISP_GetSymbol()

Description:

This function returns the AECHAR value corresponding to the specified symbol value.
NOTE: This function is deprecated in BREW 2.1 and should return EUNSUPPORTED.

Prototype:

```
int OEMDISP_GetSymbol
      (
      IOEMDisp *pMe,
      AECHAR *pChar,
      AEESymbol sym,
      AEEFont font
      )
```

Parameters:

pMe	[in]	Pointer to the OEM Display Interface.
pChar	[out]	Character value associated with the specified symbol.
sym	[in]	Requested symbol.
font	[in]	Requested font.

Return Value:

SUCCESS is returned if the function performed correctly.

EFAILED if AECHAR pointer is NULL

EUNSUPPORTED is returned if function is not implemented.

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_GetSystemFont()

Description:

The function retrieves an interface to the font specified. A new instance of the font should be created for each call to this function.

Prototype:

```
int IOEMDISP_GetSystemFont
      (
      IOEMDisp * pMe,
      AEEFont font,
      IFont ** pFont
      )
```

Parameters:

pMe	[in]	Pointer to IOEMDisp interface.
font	[in]	Requested font.
pFont	[out]	Pointer to interface of font.

Return Value:

SUCCESS is returned if function performed correctly.

EUNSUPPORTED if the specified font is not supported.

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_MapPalette()

Description:

This function sets multiple entries in the device's palette table. It only sets a contiguous set of entries, starting from index 0 (zero). If the device does not support a dynamic palette, this function returns EUNSUPPORTED.

Prototype:

```
int IOEMDisp_MapPalette
      (
      IOEMDisp *pMe,
      unsigned int cntRGB,
      RGBVAL *pRGB
      )
```

Parameters:

pMe	Pointer to the OEM Display Interface.
cntRGB	Number of entries to set.
pRGB	Array of colors that are to be used in the palette.

Return Value:

SUCCESS if the function performed correctly.

EUNSUPPORTED if the device does not have a dynamic palette.

Comments:

None

See Also:

IOEMDISP_SetPaletteEntry()

IOEMDISP_GetPaletteEntry()

Return to the List of functions

OEMDisp_New()

Description:

This function creates a new instance of the IOEMDisp interface.

Prototype:

```
int OEMDisp_New(IShell * ps, AEECLSID cls, void ** ppif)
```

Parameters:

ps	[in]	Pointer to the IShell interface.
cls	[in]	Class ID of the new IOEMDisp interface.
ppif	[out]	Pointer to the new OEM Display Interface.

Return Value:

SUCCESS is returned if the function performed correctly.

EBADPARM if the parameters are invalid

ENOMEMORY if there is not enough memory

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_SetAnnunciators()

Description:

This function turns annunciators on or off. Two bitmasks are passed as parameters: **wMask** and **wVal**.The bits set in **wMask** select the corresponding annunciators. For each of the selected annunciators, the bits in **wVal** indicate whether the corresponding annunciator needs to be turned on or off.

Prototype:

```
int IOEMDISP_SetAnnunciators
      (
      IOEMDisp *pMe,
      unsigned int wVal,
      unsigned int wMask
      )
```

Parameters:

pMe	Pointer to the OEM Display Interface.
wVal	Annunciator bitmask values.
wMask	Annunciator bitmask masks.

Return Value:

SUCCESS is returned if the function performed correctly.

EUNSUPPORTED if the operation is not supported.

Comments:

None

See Also:

None

Return to the List of functions

IOEMDISP_SetPaletteEntry()

Description:

This function sets an entry in the device's palette table. If the device does not support a dynamic palette, this function returns EUNSUPPORTED.

Prototype:

```
int IOEMDisp_SetPaletteEntry
      (
      IOEMDisp * pMe,
      unsigned int index,
      RGBVAL rgb
      )
```

Parameters:

pMe	Pointer to the OEM Display Interface.
index	Index of palette entry to change.
rgb	Color to put in the palette.

Return Value:

SUCCESS if the function performed correctly.

EUNSUPPORTED if the device does not have a dynamic palette.

Comments:

None

See Also:

IOEMDISP_GetPaletteEntry()

IOEMDISP_MapPalette()

Return to the List of functions

IOEMDISP_Update()

Description:

This function updates the graphic memory (display) using the contents of the shadow buffer. The function can do either of the following:

- Deferred refresh where the update waits for any ongoing draw operations to complete.
- Forced refresh where the update happens immediately.

Prototype:

```
int IOEMDISP_Update(IOEMDisp *pMe, boolean bDefer)
```

Parameters:

pMe	Pointer to the OEM Display Interface.
bDefer	Flag that determines whether to do a deferred update or a forced update.

Return Value:

SUCCESS is returned if the function performed correctly.

Other implementation-specific error codes.

Comments:

None

See Also:

None

Return to the List of functions

OEM File System Interface

This section describes the File System interface functions that the AEE uses. The interface provides a basic foundation for simple file system operations.

List of functions

Functions in this interface include:

OEMFS_Close()
OEMFS_EnumNext()
OEMFS_EnumStart()
OEMFS_EnumStop()
OEMFS_GetDirInfo()
OEMFS_GetFileAttributes()
OEMFS_GetLastError()
OEMFS_GetOpenFileAttributes()
OEMFS_Mkdir()
OEMFS_Open()
OEMFS_Read()
OEMFS_Remove()
OEMFS_Rename()
OEMFS_Rmdir()
OEMFS_Seek()
OEMFS_SpaceAvail()
OEMFS_SpaceUsed()
OEMFS_Tell()
OEMFS_Test()
OEMFS_Truncate()
OEMFS_Write()

The remainder of this section provides details for each function.

OEMFS_Close()

Description:

This function closes the file identified by **pFileHandle**. The function takes the file handle returned by OEMFS_Open() and frees all resources associated with it. Subsequent operations on the file handle fail.

Prototype:

```
int OEMFS_Close(void * pFileHandle)
```

Parameters:

pFileHandle File handle returned from OEMFS_Open().

Return Value:

SUCCESS if the operation was successful

Valid BREW-defined error code, otherwise.

EMEMPTR: "pFileHandle" does not point to a valid handle

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

Side Effects:

Any OEM-specific memory structures associated with the file are closed.

See Also:

OEMFS_Open()

Return to the List of functions

OEMFS_EnumNext()

Description:

This function uses the file or directory enumeration control object referenced by **pIterator** to return file information for the next file or directory specified in OEMFS_EnumStart(). The function returns not only the name, but all of the other parameters of AEEFileInfo, including flags, creation date, and file size.

Upon success, the OEMFSEnum object will contain the values associated with the enumerated file or directory.

Prototype:

```
int OEMFS_EnumNext
      (
      OEMFSEnum *pcxt
      )
```

Parameters:

pcxt	Input/Output structure for the enumeration operation.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

SUCCESS error code if no more items are to be enumerated. The function fills **pInfo** with zeros.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

OEMFS_EnumStart()

OEMFS_EnumStop()

OEMFS_GetLastError()

Return to the List of functions

OEMFS_EnumStart()

Description:

This function initializes a file or directory enumeration control object for use by subsequent calls to OEMFS_EnumNext(). Enumeration can be of either files or directories in the parent directory **szDir**. To enumerate files, the **isDir** parameter is zero. To enumerate directories, the value is non-zero.

Prototype:

```
OEMFSEnum * OEMFS_EnumStart
      (
      const char*szDir,
      charisDir
      )
```

Parameters:

szDir	Directory to be enumerated.
isDir	Flag to indicate enumeration of files or directories.

Return Value:

Returns a pointer to the structure that will be populated with the results of all future OEMFS_EnumNext() operations.

Comments:

BREW will reference members of the OEMFSEnum only during the time that it is valid (between OEMFS_EnumStart() and OEMFS_EnumStop())

This function should set an internal error value that can be retrieved with OEMFS_GetLastError().

Common error codes:

EFILENOEXISTS: If an element of the directory path does not exist

EBADFILENAME: If "szDir" is NULL, empty or longer than AEE_MAX_FILE_NAME

Side Effects:

This function allocates memory to hold the structure.

See Also:

OEMFS_EnumNext()

OEMFS_EnumStop()

Return to the List of functions

OEMFS_EnumStop()

Description:

This function frees any resources associated with an enumeration.

Prototype:

```
int OEMFS_EnumStop
      (
      OEMFSEnum *pcxt
      )
```

Parameters:

pcxt	Enumeration control object returned from OEMFS_EnumStart int OEMFS_EnumStop(void * pIterator)

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

EMEMPTR: if "pIterator" is not valid

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

OEMFS_EnumStart()

OEMFS_EnumNext()

Return to the List of functions

OEMFS_GetDirInfo()

Description:

Get the following information about the given directory:

- Number of files in this directory (including sub-directories)
- Total space occupied by this directory (including sub-directories)

Prototype:

```
int OEMFS_GetDirInfo
      (
      IFileMgr *pfm,
      const char* pszDir,
      uint16 *pwCount,
      uint32 *pdwSize
      );
```

Parameters:

pfm	Pointer to IFileMgr interface
pszDir	Directory whose info is needed. The AEE function AEE_BuildPath() must be used to build the complete path specified by pszDir.
pwCount	If non Null on input, *pwCount must contain the total number of files in this directory tree on return
pdwSize	If non NULL on onput, *pdwSize must contain the total space occuped by this directory tree on return

Return Value

AEE_SUCCESS, if successful

EFAILED, if failed

Comments:

None

See Also:

None

Return to the List of functions

OEMFS_GetFileAttributes()

Description:

This function returns the following file attributes of the file specified by **szFilename**:

- Flags (Hidden, System, Directory)
- Creation date in seconds from the beginning of the GPS epoch
- File size
- Filename

The attributes matched are exactly the same as those currently returned to the programmer by the upper level API.

Prototype:

```
int OEMFS_GetFileAttributes
      (
      const char * szFilename,
      AEEFileInfo * pInfo
      )
```

Parameters:

szFilename	Name of the open file
pInfo	Pointer to a structure used to return the file attributes

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_GetLastError()

Description:

This function gets the last error produced by the OEMFS subsystem. If the last operation was successful, this function returns SUCCESS.

Prototype:

```
int OEMFS_GetLastError(void)
```

Parameters:

None

Return Value:

SUCCESS if successful.

Valid BREW-defined error code if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEMFS_GetOpenFileAttributes()

Description:

This function returns the following file attributes of the currently open file specified by **pFileHandle**:

- Flags (Hidden, System, Directory)
- Creation date in seconds from the beginning of the GPS epoch
- File size
- Filename

Prototype:

```
int OEMFS_GetOpenFileAttributes
      (
      void* pFileHandle,
      const char* szFilename,
      AEEFileInfo *pInfo
      )
```

Parameters:

pFileHandle	File handle returned by OEMFS_Open().
szFilename	Name of the open file.
pInfo	Pointer to structure used to return data.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Mkdir()

Description:

This function creates a new user file directory named **dirname**. This directory is created as long as the new directory's parent directories exist.

Prototype:

```
int OEMFS_Mkdir(const char * dirname)
```

Parameters:

dirname	Name of new directory.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Open()

Description:

This function opens the file specified by **szFilename** for the type of access specified by **nMode**. The function allocates and initializes a data structure that it returns. The current file position is set to be the beginning of the file.

Prototype:

```
void* OEMFS_Open(const char * szFilename, AEEOpenFileMode nMode)
```

Parameters:

szFilename	Name of file to open.
nMode	File open mode, which ttakes the following values: _OFM_READ: File is opened for read only and will not create a file. _OFM_READWRITE: File is opened in readwrite mode and will not create a file. _OFM_APPEND: Same as _OFM_READWRITE but sets the file pointer to the end of the file. . _OFM_CREATE: Creates a new file in read/write mode. An error is generated, and no file opened, if the file already exists.

Return Value:

Pointer identifying the open file. The AEE does not attempt to de-reference the pointer, but instead passes it as a parameter to all functions that act upon the open file (such as read, write, and truncate).

NULL if there is an error.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

OEMFS_Close()

Return to the List of functions

OEMFS_Read()

Description:

This function reads **nBytes** bytes from the file identified by **pFileHandle** into **buffer** starting at the current file pointer position associated with **pFileHandle**. The number of bytes actually read is returned. If an error occurs with the operation, 0 (zero) is returned.

Prototype:

```
uint32 OEMFS_Read(void * pFileHandle, void * buffer, uint32 nBytes)
```

Parameters:

pFileHandle	Handle of file to read, returned by OEMFS_Open().
buffer	Buffer with bytes to read.
nBytes	Number of bytes to read.

Return Value:

Number of bytes read from the **buffer**.

0 (zero) if there is an error, including an End-Of-File (EOF) error.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

Side Effects:

The current file pointer position is incremented so that it points immediately after the last byte returned.

See Also:

OEMFS_Write()

Return to the List of functions

OEMFS_Remove()

Description:

This function removes the file identified by **Filename**.

Prototype:

```
int OEMFS_Remove(const char * Filename)
```

Parameters:

Filename	Name of file to remove

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Rename()

Description:

This function renames the file identified by **old_filename**, the old filename, to **new_filename**, the new filename.

Prototype:

```
int OEMFS_Rename(const char * old_filename, const char * new_filename)
```

Parameters:

old_filename	Current name of the file.
new_filename	New name of the file.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Rmdir()

Description:

This function removes the user file directory specified by **dirname**.

Prototype:

```
int OEMFS_Rmdir(const char * dirname)
```

Parameters:

dirname	Directory to remove.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Seek()

Description:

This function sets the current file pointer position of the file associated with **pFileHandle** from a starting point **sType**, offset by a number of bytes (positive or negative) specified by **nOffset**. The sType parameter can be one of three values:

- _SEEK_START (offset is based on the beginning of the file)
- _SEEK_END (offset is based on the end of the file)
- _SEEK_CURRENT (offset from the current position)

If the new seek pointer is beyond the end of the file, the file is enlarged to accomodate the seek.

Prototype:

```
int OEMFS_Seek
      (
      void * pFileHandle,
      AEEFileSeekType sType,
      int32 nOffset
      )
```

Parameters:

pFileHandle	File handle returned by OEMFS_Open().
sType	Type of seek starting point.
nOffset	Offset from starting point.

Return Value:

SUCCESS if the operation was successful

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

Side Effects:

Sets the current file position associated with **pFileHandle** to the specified position.

See Also:

None

Return to the List of functions

OEMFS_SpaceAvail()

Description:

This function returns the amount of available file system space (in bytes).

Prototype:

```
uint32 OEMFS_SpaceAvail(void)
```

Parameters:

None

Return Value:

Number of bytes of file system space currently available.

Comments:

The file service handling task must have been started already. This function does not set or return an error.

See Also:

None

Return to the List of functions

OEMFS_SpaceUsed()

Description:

This function returns the amount of file system space (in bytes) in use. The function call is synchronous and does not involve the file service handling task command queues.

Prototype:

```
uint32 OEMFS_SpaceUsed(void)
```

Parameters:

None

Return Value:

Number of bytes of file system space currently in use.

Comments:

The file service handling task must have been started already. This function does not set or return an error.

See Also:

None

Return to the List of functions

OEMFS_Tell()

Description:

This function returns the current file pointer position of the file associated with **pFileHandle**.

Prototype:

```
int OEMFS_Tell(void * pFileHandle)
```

Parameters:

pFileHandle	File handle returned from OEMFS_Open().

Return Value:

Offset from beginning of the file if successful.

-1 if an error occurs with the operation.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Test()

Description:

This function tests for the existence of a file or a directory. It first checks if the directory exists, and then checks the specified name.

Prototype:

```
int OEMFS_Test(const char * filename)
```

Parameters:

filename	File or directory to check for existence.

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Truncate()

Description:

This function truncates the file identified by **pFileHandle** to the position specified by **nPos**. The file must not be open for read only operations. The offset must be less than the total length of the file.

Prototype:

```
int OEMFS_Truncate(void * pFileHandle, uint32 nPos)
```

Parameters:

pFileHandle	File handle.
nPos	File truncate position (new file size).

Return Value:

SUCCESS if the operation was successful.

Valid BREW-defined error code, otherwise.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

See Also:

None

Return to the List of functions

OEMFS_Write()

Description:

This function writes **nBytes** bytes from **buffer** into the file identified by **pFileHandle** starting at the current file pointer position associated with **pFileHandle**. The number of bytes actually written is returned following operation completion.

Prototype:

```
uint32 OEMFS_Write
      (
      void * pFileHandle,
      const void * buffer,
      uint32 nBytes
      )
```

Parameters:

pFileHandle	Handle of file to write to, returned by OEMFS_Open().
buffer	Buffer with bytes to write.
nBytes	Number of bytes to write.

Return Value:

Number of bytes written to the **buffer**.

0 (zero) if there is an error.

Comments:

The file service handling task must have been started already. This function sets an internal error value that can be retrieved with OEMFS_GetLastError().

Side Effects:

The current file pointer position is incremented so that it points immediately after the last byte returned.

See Also:

OEMFS_Read()

Return to the List of functions

OEM Heap Interface

This section describes the basic heap memory routines that the AEE files use to provide memory management functionality.

List of functions

Functions in this interface include:

OEM_CheckMemAvail()
OEM_Free()
OEM_GetRAMFree()
OEM_InitHeap()
OEM_Malloc()
OEM_Realloc()

The remainder of this section provides details for each function.

OEM_CheckMemAvail()

Description:

This function checks to see if a memory block of the given size can be allocated. The function does not allocate memory, it simply returns TRUE or FALSE to indicate whether it is possible to allocate a block of the given size.

Prototype:

```
boolean OEM_CheckMemAvail(uint32 dwSize)
```

Parameters:

dwSize	Size of the block whose allocation is to be verified.

Return Value:

TRUE if a memory block of the given size can be allocated.

FALSE 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 are available.

See Also:

None

Return to the List of functions

OEM_Free()

Description:

This function frees an allocated memory buffer.

Prototype:

```
void OEM_Free(void * pBuff)
```

Parameter(s):

pBuff	Pointer to buffer that is to be freed.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_GetRAMFree()

Description:

This function returns the number of free bytes in the heap. It conditionally fills the values of the total heap size and the largest block that can be allocated.

Prototype:

```
uint32 OEM_GetRAMFree(uint32 * pdwTotal, uint32 * pdwMax)
```

Parameter(s):

pdwTotal	Pointer to the value to set with the total space in the file system.
pdwMax	Pointer to the value to set with the maximum size block that can be allocated.

Return Value:

Number of free bytes in the heap.

Comments:

None

See Also:

None

Return to the List of functions

OEM_InitHeap()

Description:

This function initializes the heap sub-allocator.

Prototype:

```
void OEM_InitHeap(void * pMem, uint32 dwSize)
```

Parameter(s):

pMem	Memory buffer.
dwSize	Size in bytes.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_Malloc()

Description:

This function allocates a buffer from the heap.

Prototype:

```
void * OEM_Malloc(uint32 dwNewSize)
```

Parameter(s):

dwNewSize	Size, in bytes, of the buffer to be allocated from the heap.

Return Value:

Pointer to the allocated buffer if successful.

NULL if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_Realloc()

Description:

This function allocates or reallocates a buffer from the heap.

Prototype:

```
void * OEM_Realloc(void * pBuff, uint32 dwNewSize)
```

Parameter(s):

pBuff	Buffer or NULL.
dwNewSize	New size of the buffer.

Return Value:

Pointer to the rellocated buffer if successful.

NULL if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEMLogger Interface

Description:

BREW provides a standardized and extensible data logging interface, which allows a BREW application developer to log data using a number of different transport mechanisms.

Below are the primary logging transport implementations. A BREW application developer selects one by creating an ILogger instance with one of the following class IDs:

Class ID	Description
AEECLSID_LOGGER_FILE	Sends log items to a file.
AEECLSID_LOGGER_WIN	Sends log items to the Emulator output window.

Each implementation is responsible for handling and writing to a specific transport but the data being sent is transport independent.

The header file AEELoggerTypes.h provides definitions for the logging data types common to both BREW's ILogger interface and the PC side log parser in a client/server type of architecture.

The file implementation outputs data to the output file in the following BREW packet format:

The Windows implementation of the ILogger interface writes all outgoing logs to the BREW output window using the following format:

bkt:xx typ:xx clD:xx iID:xx FILENAME LINENUMBER MESSAGE ARGS

in which

bkt	Log bucket
typ	Log type
cID	ClassID of the currently running BREW application
iID	User-defined instance ID
FILENAME	Optional file name where log was sent
LINENUMBER	Optional line number where log was sent
MESSAGE	User defined text message
ARGS	Optional arguments using OEMLogger_PutMsg()

When compiling a release version of a BREW application, the constant AEE_LOG_DISABLE may be defined, which, using the preprocessor, removes all OEMLogger interface logging functions, except the instance creation and getting and setting parameters processes. This constant must be defined before a new BREW application includes AEELogger.h.

The contents of log data is determined by the type element of the BREW Log header. Three standard log types are predefined by BREW, but the BREW application developer can also define as many custom log types as required. The three standard BREW-defined log types are as follows:

Type	Description	Data contains
AEE_LOG_TYPE_TEXT	ASCII text message	If you use this log type, the data contains nSize bytes of ASCII text.
AEE_LOG_TYPE_BIN_MSG	AEELogTypeBinMsg	If you use this log type, the data contains one AEE LogTypeBinMsg structure.
AEE_LOG_TYPE_BIN_BLK	Block of arbitrary binary data	If you use this log type, the data contains nSize bytes of arbitrary binary data.

Log items are sent and filtered in one of 255 distinct, general purpose buckets. These log buckets are filtered by the developer at run time using ILOGGER_SetParam() and ILOGGER_GetParam() or on the PC side, using a post processor.

The structure AEELogTypeBinMsg contains the following elements:

Element	Description
Header	b7,b6 – bits reserved b5,b4 – number of args b3 bit – file name present b2,b1,b0 – message level
Line	Line number in application code where this log item was sent
args[MAX_LOG_TYPE_BIN_MSG_ARGS]	Contains zero or more 32 bit integer values
pszMsg[MAX_LOG_TYPE_BIN_MSG_TEXT_SIZE]	pszMsg contains two consecutive NULL terminated strings: the first is the file name where the log message was sent and the second is an ASCII text message

List of Header files to be included

The following header file is required:

OEMLogger.h

List of functions

Functions in this interface include:

OEMLogger_Printf()
OEMLogger_PutItem()
OEMLogger_PutMsg()
OEMLoggerDMSS_GetParam()
OEMLoggerDMSS_PutRecord()
OEMLoggerDMSS_SetParam()
OEMLoggerFile_GetParam()
OEMLoggerFile_PutRecord()
OEMLoggerFile_SetParam()
OEMLoggerWin_GetParam()
OEMLoggerWin_PutRecord()
OEMLoggerWin_SetParam()

The remainder of this section provides details for each function.

OEMLogger_Printf()

Description:

This function is called to send a prioritized formatted ASCII text message. Since this routine is a MACRO that allows variable arguments it must be called as follows:

```
OEMLogger_Printf( pMe->m_pILogger,
   ( pMe->m_pILogger,
      USER_BUCKET1,
      __FILE__,
      (uint16)__LINE__,
      "msg",
      args )
   );
```

Notice that the second argument is actually multiple arguments in parentheses, and args can be multiple comma separated values

Prototype:

```
int OEMLogger_Printf
      (
      ILogger *pMe,
      AEELogBucketType bucket,
      const char *pszFileName,
      uint16 nLineNum,
      const char *pszFormat,
      ... );
```

Parameters:

pMe	Pointer to the OEMLogger Interface object
bucket	Bucket to place item
pszFileName	Name of file calling this function
nLineNum	Line number in file where it was called
pszFormat	ASCII text string similar to a printf format string
...	Format string arguments

Return Value:

SUCCESS Log send successfully

EBADPARM Invalid pointer to pszFormat

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocated required memory

EFAILED Log not sent

-- The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

Return to the List of functions

OEMLogger_PutItem()

Description:

This function is called to send a prioritized user defined binary message.

To define a user log item type:

1. Select a user item number and define a meaningful name to it, For Example:

```
#define MY_APPS_LOG_ITEM_TYPE AEE_LOG_TYPE_USER_1
```

2. Define a structure that corresponds to you're new type, Example:

```
typedef struct{
      uint8 foo1;
      uint32 foo2;
      uint8 fooString[ STRING_SIZE ];
      } myAppsItem;
```

3. Enable the PC software that will be reading the logging output to recognize the log item type AEE_LOG_TYPE_USER_1 (which in this case is MY_APPS_LOT_ITEM_TYPE)

4. Call OEMLogger_PutItem() with MY_APPS_LOT_ITEM_TYPE, a pointer to an instance of myAppsItem, and the size of myAppsItem.

Prototype:

```
int OEMLogger_PutItem( ILogger *pMe,
      AEELogBucketType bucket,
      AEELogItemType type,
      uint16 nSize,
      uint8 *pItem )
```

Parameters:

pMe	Pointer to the OEMLogger Interface object
bucket	Bucket to place item
type	User defined item type
nSize	Size of type in bytes
pItem	Pointer to instance of type

Return Value:

SUCCESS Log send successfully

EBADPARM Invalid pointer to pItem or size equal to zero

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocated required memory

EFAILED Log not sent

The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType
AEELogItemType

Return to the List of functions

OEMLogger_PutMsg()

Description:

This function is called to send a prioritized predefined binary message and allows fast logging due to the limited formatting required and the fixed size of the outgoing log message. The outgoing binary message's data of type structure AEELogBinMsgType, which is defined in AEELoggerTypes.h.

Prototype:

```
int OEMLogger_PutMsg( ILogger *pMe,
      AEELogBucketType bucket,
      const char *pszFileName,
      uint16 nLineNum,
      const char *pszMsg,
      uint8 nNumArgs,
      uint32 args[ MAX_LOG_TYPE_BIN_MSG_ARGS ] )
```

Parameters:

pMe	Pointer to the OEMLogger Interface object
bucket	Bucket to place item
pszFileName	ASCII NULL terminated name of file calling this function
nLineNum	Line number in file where it was called
pszMsg	ASCII NULL terminated text message
nNumArgs	length of the args array
args	array containing uint32 arguments

Return Value:

SUCCESS Log send successfully

EBADPARM Invalid pointer to pszMsg or nNumArgs too large

EUNSUPPORTED Log item filtered

ENOMEMORY Unable to allocated required memory

EFAILED Log not sent

The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

AEELogBinMsgType

Return to the List of functions

OEMLoggerDMSS_GetParam()

Description:

This function is called to get the configuration of the OEM logging interface.

Prototype:

```
int OEMLoggerDMSS_GetParam
      (
      ILogger *po,
      AEELogParamType pType,
      void* pParam
      )
```

Parameters:

pMe	Pointer to the ILogger object
pType	Parameter to modify
pParam	Pointer to settings parameter to fill

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEMLoggerDMSS_PutRecord()

Description:

This function is called by ILOGGER_Printf(), ILOGGER_PutItem(), or ILOGGER_PutMsg() to output a log record to the DMSS Diag task.

Prototype:

int ILOGGER_PutRecord

```
(
ILogger *po,
AEELogBucketType bucket,
AEELogRecord *pRcd
)
```

Parameters:

pMe: Pointer to the ILOGGER object

bucket: Bucket to place item

pItem: Pointer to data with BREW header at beginning to write to log

Return Value:

SUCCESS Item data written successfully

EBADPARM Invalid pointer to pItem

EFAILED General failure

-- The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

Return to the List of functions

OEMLoggerDMSS_SetParam()

Description:

This function is called to set performance and behavior of the logging interface. Supported parameters depends on the current implementation's support,

Prototype:

```
int OEMLoggerDMSS_SetParam
      (
      ILogger *po,
      AEELogParamType pType,
      uint32 param,
      void* pParam )
```

Parameters:

pMe	Pointer to the ILogger object
pType	Parameter to modify
param	New settings parameter
pParam	Pointer to new settings parameter

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EFAILED: General failure

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEMLoggerFile_GetParam()

Description:

This function is called to get the configuration fo the ILogger interface. Supported parameters depends on the current implementation's support,

Prototype:

```
int OEMLoggerFile_GetParam
      (
      ILogger *po,
      AEELogParamType pType,
      void* pParam
      )
```

Parameters:

pMe	Pointer to the ILogger object
pType	Parameter to modify
pParam	Pointer to be filled with settings parameter

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEMLoggerFile_PutRecord()

Description:

This function is called by ILOGGER_Printf(), ILOGGER_PutItem(), or ILOGGER_PutMsg() to output a log record to the output log file.

Prototype:

```
int ILOGGER_PutRecord
      (
      ILogger *po,
      AEELogBucketType bucket,
      AEELogRecord *pRcd
      )
```

Parameters:

pMe	Pointer to the ILOGGER object
bucket	Bucket to place item
pRcd	Pointer to data with BREW header at beginning to write to log

Return Value:

SUCCESS Item data written successfully

EBADPARM Invalid pointer to pItem

EFAILED General failure

The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

Return to the List of functions

OEMLoggerFile_SetParam()

Description:

This function is called to set performance and behavior of the logging interface.Supported parameters depends on the current implementation's support,

Prototype:

```
int OEMLoggerFile_SetParam
      (
      ILogger *po,
      AEELogParamType pType,
      uint32 param,
      void* pParam
      )
```

Parameters:

pMe	Pointer to the ILogger object
pType	Parameter to modify
param	New settings parameter
pParam	Pointer to new settings parameter

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEMLoggerWin_GetParam()

Description:

This function is called to get the configuration fo the ILogger interface.Supported parameters depends on the current implementation's support,

Prototype:

```
int OEMLoggerWin_GetParam
      (
      ILogger *po,
      AEELogParamType pType,
      void* pParam
      )
```

Parameters:

pMe: Pointer to the ILogger object

pType: Parameter to modify

pParam: Pointer to be filled with settings parameter

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEMLoggerWin_PutRecord()

Description:

This function is called by ILOGGER_Printf(), ILOGGER_PutItem(), or ILOGGER_PutMsg() to output a log record to the BREW Emulator output window.

This function only supports text output and therefore only supports log types AEE_LOG_TYPE_TEXT and AEE_LOG_TYPE_BIN_MSG.

Prototype:

```
int OEMLoggerWin_PutRecord
      (
      ILogger *po,
      AEELogBucketType bucket,
      AEELogRecord *pRcd
      )
```

Parameters:

pMe	Pointer to the ILOGGER object
bucket	Bucket to place item
pItem	Pointer to data with BREW header at beginning to write to log

Return Value:

SUCCESS Item data written successfully

EBADPARM Invalid pointer to pItem

EFAILED General failure

The following log codes only apply to file logging

EFSFULL Not enough space in log file for this packet

EFILENOEXISTS Output log file is closed

Comments:

None

See Also:

AEELogBucketType

Return to the List of functions

OEMLoggerWin_SetParam()

Description:

This function is called to set performance and behavior of the logging interface. Supported parameters depends on the current implementation's support,

Prototype:

```
int OEMLoggerWin_SetParam
      (
      ILogger *po,
      AEELogParamType pType,
      uint32 param,
      void* pParam
      )
```

Parameters:

pMe	Pointer to the ILogger object
pType	Parameter to modify
param	New settings parameter
pParam	Pointer to new settings parameter

Return Value:

EMEMPTR: Invalid pParam pointer if pParam is required for this LogParamType

SUCCESS: LogParamType handled successfully

EUNSUPPORTED: The log parameter is not supported

Comments:

None

See Also:

AEELogParamType

Return to the List of functions

OEM MD5 Interface

This section describes the functions in the OEM MD5 interface.

List of functions

Functions in this interface include:

OEMMD5_Final()
OEMMD5_Init()
OEMMD5_Update()

The remainder of this section provides details for each function.

OEMMD5_Final()

Description:

This function ends an MD5 message-digest operation, writing the message digest and changing the context to 0 (zero).

Prototype:

```
void OEMMD5_Final(uint8 digest[16], OEMMD5_CTX * context)
```

Parameters:

digest	Message digest.
context	Pointer to the MD5 context.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMMD5_Init()

Description:

This function begins an MD5 operation, writing a new context.

Prototype:

```
void OEMMD5_Init(OEMMD5_CTX * context)
```

Parameters:

context	Pointer to a context that will be initialized.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMMD5_Update()

Description:

This function continues an MD5 message-digest operation, processing another message block and updating the context.

Prototype:

```
void OEMMD5_Update
      (
      OEMMD5_CTX * context,
      uint8 * input,
      uint32 inputLen
      )
```

Parameters:

context	MD5 context.
input	Input block.
inputLen	Length of the input block.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM Net Interface

The OEM Net interface provides the underlying networking services required for sockets to operate. Primarily this consists of functions for managing PPP, with a few additional utility functions. If an operation is unable to complete immediately, it should return an error code of AEE_NET_EWOULDBLOCK. When the operation completes, call AEE_NetEventOccurred to notify BREW.

The following functions are optional and may return EUNSUPPORTED:

OEMNet_GetPPPAuth()
OEMNet_GetRLP3Cfg()
OEMNet_GetUrgent()
OEMNet_SetPPPAuth()
OEMNet_SetRLP3Cfg()

List of functions

Functions in this interface include:

OEMNet_CloseNetlib()
OEMNet_GetPPPAuth()
OEMNet_GetRLP3Cfg()
OEMNet_GetUrgent()
OEMNet_MyIPAddr()
OEMNet_NameServers()
OEMNet_OpenNetlib()
OEMNet_PPPClose()
OEMNet_PPPOpen()
OEMNet_PPPSleep()
OEMNet_PPPState()
OEMNet_SetPPPAuth()
OEMNet_SetRLP3Cfg()

The remainder of this section provides details for each function.

OEMNet_CloseNetlib()

Description:

This function closes the network library for the application. All sockets must have been closed for the application before closing the library. If this is the last remaining application, the network subsytem (PPP/traffic channel) must also be brought down before closing the library. The function is called from the context of the socket client's task.

Prototype:

```
int16 OEMNet_CloseNetlib(void)
```

Parameters:

None

Return Value:

AEE_NET_SUCCESS, if successful

AEE designated error codes indicating reason for failure, if otherwise

Comments:

None

See Also:

OEMNet_OpenNetlib()

Return to the List of functions

OEMNet_GetPPPAuth()

Description:

This function allows the caller to retrieve the configured PPP authentication settings, if this is relevant to the network implementation.

Prototype:

```
int16 OEMNet_GetPPPAuth(char *pszAuth, int *pnLen)
```

Parameters:

pszAuth	[out]	The buffer into which the credentials are to be copied, in the form of 2 concatenated, null terminated strings, for example: “userid@vzw.com\000password\000”
pnLen	[in/out]	The size of pszAuth. If pszAuth is NULL, pnLen is ignored on input, and set to the number of bytes required to hold authentication information on output.

Return Value:

SUCCESS, if the credentials were retrieved

EUNSUPPORTED, if PPP authentication cannot be retrieved

EBADPARM, if pszAuth is not-NULL and pnLen is less than or equal to 0 (zero).

Comments:

None

See Also:

OEMNet_SetPPPAuth()

Return to the List of functions

OEMNet_GetRLP3Cfg()

Description:

This function allows the caller to discover the configured RLP settings, if the OEM's network layer is implemented using RLP.

Prototype:

```
int16 OEMNet_GetRLP3Cfg(int16 nOptName,AEERLP3Cfg *prlp3)
```

Parameters:

nOptName	One of 3 values	
	INET_OPT_DEF_RLP3	retrieve default RLP3 settings
	INET_OPT_CUR_RLP3	retrieve current RLP3 settings
	INET_OPT_NEG_RLP3	retrieve negotiated RLP3 settings
prlp3	[out] filled with relevant settings	

Return Value:

SUCCESS, if the settings were retrieved

EUNSUPPORTED, if RLP isn't employed or this API is otherwise unsupported

AEE designated error codes indicating reason for failure, if otherwise

Comments:

None

See Also:

OEMNet_SetRLP3Cfg()

AEERLP3Cfg

Return to the List of functions

OEMNet_GetUrgent()

Description:

Determines whether urgent sendto option is supported and the payload limit, if any. This function is synchronous, and therefore should not callback any notification function.

Prototype:

```
int16 OEMNet_GetUrgent
      (
        AEEUDPUrgent* pUrgent
      )
```

Parameters:

pUrgent	pointer to AEEUDPUrgent struct

Return Value:

On success, returns AEE_NET_SUCCESS.

On error, returns one of the AEE designated error codes indicating reason for failure.

Comments:

None

See Also:

AEEUDPUrgent

Return to the List of functions

OEMNet_MyIPAddr()

Description:

This function returns the IP address of the active session.

Prototype:

```
int16 OEMNet_MyIPAddr(INAddr * addr)
```

Parameters:

addr	Pointer to the buffer used to hold the IP address, in network byte order.

Return Value:

AEE_NET_SUCCESS, if successful.

Comments:

Always returns success because failure conditions not currently defined.

See Also:

None

Return to the List of functions

OEMNet_NameServers()

Description:

This function allows the caller to discover the configured name server addresses. The addresses may come from PPP setup or from phone configuration, or both. If it's both, the PPP addresses are listed first.

Prototype:

```
int16 OEMNet_NameServers(INAddr *ainaAddrs, int *pnNumAddrs);
```

Parameters:

ainaAddrs	[in/out] a caller-allocated array of INAddrs, filled by INetMgr with the answer to the question: "which nameservers?" If NULL, pnNumAddrs is filled with the number of addresses available .
pnNumAddrs	[in/out] caller sets this to array size of ainaAddrs, set by INetMgr to the number filled(if ainaAddrs is non-null)/available

Return Value:

SUCCESS, if the addresses are found, filled

Comments:

pnNumAddrs may be set to 0, if there is no servers configured in NVRam and there is no PPP setup available the addresses returned in ainaAddrs must be in network byte-order

See Also:

INAddr

Return to the List of functions

OEMNet_OpenNetlib()

Description:

This function opens the network library and assigns the application ID. It sets the application-defined callback functions to be called when the library and socket calls make progress. The function is called from the context of the socket client's task.

Prototype:

```
int16 OEMNet_OpenNetlib(void)
```

Parameters:

None

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is an error.

Comments:

None

See Also:

OEMNet_CloseNetlib()

Return to the List of functions

OEMNet_PPPClose()

Description:

This function initiates termination to bring down any network connections started with OEMNet_PPPOpen(). This function is asynchronous and must call AEE_NetEventOccurred() upon completion of the close operation.

Prototype:

```
int16 OEMNet_PPPClose(void)
```

Parameters:

None

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is an error.

Comments:

None

See Also:

OEMNet_PPPOpen()

OEMNet_PPPState()

Return to the List of functions

OEMNet_PPPOpen()

Description:

This function starts the network subsystem (data service and PPP) and establishes a network connection to the internet. After the network is established, this function must call AEE_NetEventOccurred() to indicate to the libraries that the connection is ready for use.

Prototype:

```
int16 OEMNet_PPPOpen(void)
```

Parameters:

None

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is an error.

Comments:

None

Side Effects:

Initiates call origination and PPP negotiation.

See Also:

OEMNet_PPPClose()

OEMNet_PPPState()

Return to the List of functions

OEMNet_PPPSleep()

Description:

This function releases data channel resources, but preserve PPP state basically: go to CDMA2000 dormant mode

Prototype:

```
int16 OEMNet_PPPSleep(void)
```

Parameters:

None

Return Value:

SUCCESS, if PPP is ASLEEP

AEE_NET_EWOULDBLOCK, if dormancy is kicked off or in progress

AEE_NET_EINVAL, if PPP is WAKING or CLOSED or the network hasn't been initialized with OpenNetLib()

AEE_NET_EOPNOTSUPP, if dormancy can't be supported

Comments:

The OEM network state should transition to NET_PPP_SLEEPING or NET_PPP_ASLEEP synchronously, and AEE_NetEventOccurred() must be called.

See Also:

None

Return to the List of functions

OEMNet_PPPState()

Description:

This function returns the state of the PPP connection.

Prototype:

```
NetState OEMNet_PPPState (void)
```

Parameters:

None

Return Value:

State of PPP connection, if successful.

NET_INVALID_STATE, if fails.

Comments:

None

See Also:

NetState

OEMNet_PPPOpen()

OEMNet_CloseNetlib()

Return to the List of functions

OEMNet_SetPPPAuth()

Description:

This function allows the caller to modify the configured PPP authentication settings, if this is relevant to the network implementation.

Prototype:

```
int16 OEMNet_SetPPPAuth(const char *cpszAuth)
```

Parameters:

cpszAuth	The new credentials, in the form of 2 concatenated, null terminated strings, for example: “userid@vzw.com\000password\000”

Return Value:

SUCCESS, if the credentials were set

EUNSUPPORTED, if PPP authentication cannot be configured

Comments:

None

See Also:

OEMNet_GetPPPAuth()

Return to the List of functions

OEMNet_SetRLP3Cfg()

Description:

This function allows the caller to modify the configured RLP settings, if the OEM's network layer is implemented using RLP.

Prototype:

```
int16 OEMNet_SetRLP3Cfg
      (
      int16 nOptName,
      AEERLP3Cfg *prlp3
      )
```

Parameters:

nOptName	One of 2 values
	INET_OPT_DEF_RLP3: set default RLP3 settings.
	INET_OPT_CUR_RLP3: set current RLP3 settings.
prlp3	the new settings

Return Value:

SUCCESS, if the settings were configured

EUNSUPPORTED, if RLP isn't employed or this API is otherwise unsupported

AEE designated error code, if there is an error.

Comments:

None

See Also:

OEMNet_GetRLP3Cfg()

AEERLP3Cfg

Return to the List of functions

OEM Registry Interface

This section describes the OEM Registry-related function.

List of functions

Functions in this interface include:

OEMRegistry_DetectType()

The remainder of this section provides details for each function.

OEMRegistry_DetectType()

Description:

Given data in a buffer or the name of an object, this function detects the MIME type. This function is typically used to get the handler associated with the data type. For example, if the data represents standard MIDI format, then this function returns the MIME "audio/mid". Using the MIME type, you can query Shell registry to obtain the handler (Class ID) of type AEECLSID_MEDIA.

Prototype:

```
int OEMRegistry_DetectType
     (
     const void * cpBuf,
     uint32 * pdwSize,
     const char * cpszName,
     const char ** pcpszMIME
     );
```

Parameters:

cpBuf	[in]	Buffer containing the data whose type needs to be determined
pdwSize	[in/out]	On input - Size of data in pBuf, unless pBuf is NULL, then ignored On output - number of additional data bytes needed to perform type detection
cpszName	[in]	Name of the object whose type needs to be determined (may be null, if unknown).
pcpszMIME	[out]	MIME string returned to caller, on return, filled with a pointer to a constant string (do not free)

Return Value:

SUCCESS: Data type is detected and MIME is returned

ENOTYPE: There is no type associated with this data

EBADPARM: Wrong input data (parameter(s))

ENEEDMORE: Need more data to perform type detection. *pdwSize contains the the required number of additional bytes.

EUNSUPPORTED: Type detection for the specified input is not supported

Comments:

pBuf takes precedence over pszName. If both of them are specified, then first pBuf is used for type detection followed by pszName.

If the function returns ENEEDMORE, then *pdwSize is filled with the required additional bytes to carry out the operation. Call this function again with (original dwSize + *pdwSize) bytes.

To determine the maximum number of bytes required to enable type detection, you can call

```
if (ENEEDMORE == ISHELL_DetectType(ps, NULL, &dwReqSize, NULL,
NULL))
{
// dwReqSize contains the max bytes needed for type detection
}
```

IMPORTANT NOTE TO OEMs:

(1) Do not modify the existing type detection code.

(2) Add your new type detection functions and you may order them accordingly.

See Also:

ISHELL_DetectType()

ISHELL_GetHandler()

ISHELL_CreateInstance()

Return to the List of functions

OEM Operating System Interface

This section describes the functions in the OEM Operating System Interface.

List of functions

Functions in this interface include:

OEMOS_ActiveTaskID()
OEMOS_BrewHighPriority()
OEMOS_BrewNormalPriority()
OEMOS_CancelDispatch()
OEMOS_GetLocalTime()
OEMOS_GetTimeMS()
OEMOS_GetUptime()
OEMOS_LocalTimeOffset()
OEMOS_SetTimer()
OEMOS_SignalDispatch()
OEMOS_Sleep()

The remainder of this section provides details for each function.

OEMOS_ActiveTaskID()

Description:

This function returns the ID of the currently running task.

Prototype:

```
uint32 OEMOS_ActiveTaskID(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_BrewHighPriority()

Description:

This function raises BREW's task priority so that certain time-limited operations (such as signature verification) will be performed more quickly.

Prototype:

```
void OEMOS_BrewHighPriority(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_BrewNormalPriority()

Description:

This function return BREW's task priority to its normal level.

Prototype:

```
void OEMOS_BrewNormalPriority(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_CancelDispatch()

Description:

If there is an event in the queue to call AEE_DISPATCH, this function sets its enable field to FALSE so that AEE_DISPATCH will not be called. If such an event does not exist, one is created. This is OK because the event will have its enable set to FALSE so it will be ignored by the event handler.

Prototype:

```
void OEMOS_CancelDispatch(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_GetLocalTime()

Description:

This function returns the current time in seconds since 1/6/1980.

Prototype:

```
uint32 OEMOS_GetLocalTime(void)
```

Parameters:

None

Return Value:

The current time in seconds since 1/6/1980.

Comments:

The time returned by this function can change when the device acquires time from a network. Therefore, do not assume that each call to this function will return a greater value.

Even though this function returns the current time in seconds, the accuracy of that time is determined by the time resolution of the underlying hardware/software platform.

See Also:

OEMOS_GetTimeMS()

OEMOS_GetUptime()

Return to the List of functions

OEMOS_GetTimeMS()

Description:

This function returns the number of milliseconds since midnight.

Prototype:

```
uint32 OEMOS_GetTimeMS(void)
```

Parameters:

None

Return Value:

The number of milliseconds since midnight.

Comments:

The time returned by this function can change when the device acquires time from a network. Therefore, do not assume that each call to this function will return a greater value.

Even though this function returns the current time in milliseconds, the accuracy of that time is determined by the time resolution of the underlying hardware/software platform.

See Also:

OEMOS_GetLocalTime()

OEMOS_GetUptime()

Return to the List of functions

OEMOS_GetUptime()

Description:

This function returns the time in milliseconds since the device started.

Prototype:

```
uint32 OEMOS_GetUptime(void)
```

Parameters:

None

Return Value:

The time in milliseconds since the device started.

Comments:

If a device is turned on for approximately 50 days, this value can roll over and restart at zero.

See Also:

OEMOS_GetLocalTime()

OEMOS_GetTimeMS()

Return to the List of functions

OEMOS_LocalTimeOffset()

Description:

This function returns the local time zone offset from UTC, in seconds. It optionally returns a flag indicating that daylight savings time is active. if it is active, the value of the local time zone offset already takes the shift into account, and the flag controls the display of a time zone name.

The returned value is added to UTC to give the local time, or subtracted from the local time to give the UTC time.

Prototype:

```
int32 OEMOS_LocalTimeOffset(boolean * DaylightSavings)
```

Parameters:

DaylightSavings	Pointer to boolean, which is set to TRUE if daylight savings time is active.

Return Value:

The local time zone offset from UTC in seconds.

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_SetTimer()

Description:

This function sets the master OEM timer to **nMSecs** milliseconds. After **nMSecs**, the OEM code will call the AEE_TimerExpired() function. A call to OEMOS_SetTimer while another timer is still pending will cancel the previous timer before setting the new one. A call to OEMOS_SetTimer with an **nMSecs** value of 0 will cancel the pending timer, if one is active.

Prototype:

```
void OEMOS_SetTimer(uint32 nMSecs)
```

Parameters:

nMSecs	Number of milliseconds to set the master OEM timer.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_SignalDispatch()

Description:

This function adds an event to the event queue that will cause AEE_DISPATCH to be called.

Prototype:

```
void OEMOS_SignalDispatch(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEMOS_Sleep()

Description:

This function delays execution of subsequent code for **nMSecs** milliseconds. It will block for **nMSecs**.

Prototype:

```
void OEMOS_Sleep(uint32 nMSecs)
```

Parameters:

nMSecs	Number of milliseconds to sleep.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM Random Number Generator Interface

This interface provides all of the basic routines for Random Number Generation.

List of functions

Functions in this interface include:

OEMRan_GetNonPseudoRandomBytes()
OEMRan_Next()
OEMRan_Seed()

The remainder of this section provides details for each function.

OEMRan_GetNonPseudoRandomBytes()

Description:

Return 20 bytes of crypto quality random data. This routine need not return new random data more than about once every 100ms. If you don't have a source of pure random numbers, fill the buffer with zeros and return.

Prototype:

```
void OEMRan_GetNonPseudoRandomBytes(byte *pbRand, int *pcbLen);
```

Parameters:

pbRand	buffer to fill with random data
pcbLen	length of buffer on input, length of random data on output

Return Value:

None

Comments:

Brew internally produces cryptographic quality random numbers using key stroke and network timing and randomness in the memory manager. However if an additional very high quality source of randomness is available it can be fed in to the pool here. A good example would be noise taken of the phone antenna. A bad example would be randomness from key strokes or time.

If you have more good random data, return it 20 bytes at a time when called here. If the pool you have is larger than 20 bytes, used SHA-1 or MD5 to reduce it to 20 bytes. If you don't have good random data this should return a buffer of zeros.

No concern should be given here for accumulating or stirring the random pool. This is all handled internally.

See Also:

None

Return to the List of functions

OEMRan_Next()

Description:

This function returns the next number in the sequence.

Prototype:

```
uint32 OEMRan_Next(void)
```

Parameters:

None

Return Value:

The next number in the sequence.

Comments:

None

See Also:

None

Return to the List of functions

OEMRan_Seed()

Description:

This function seeds the random number generator.

Prototype:

```
void OEMRan_Seed(uint32 seed)
```

Parameters:

seed	Seed value for the random number generator.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM SMS Interface

This section describes the SMS Interface functions that the AEE files use to provide Short Message Service functionality to BREW applications.

List of Functions

Functions in this interface include:

OEM_extract_SMS_text()
OEM_format_SMS_msg()
OEM_format_SMS_text()
OEM_uasms_config_listeners()

The remainder of this section provides details for each function.

OEM_extract_SMS_text()

Description:

This function extracts text from the SMS message. This routine is necessary because OEMs in some markets choose to decode into an alternate format that is more appropriate for the type of SMS supported.

Prototype:

```
AEESMSTextMsg * OEM_extract_SMS_text
      (
      const uasms_user_data_type * pm,
      byte * pDest,
      int nSize
      )
```

Parameters:

pm	Input SMS message (user data).
pDest	Buffer to hold the text extracted from the SMS message.
nSize	Size of the destination buffer.

Return Value:

Final formatted SMS text if successful.

NULL if fails.

Comments:

None

See Also:

None

Return to the List of Functions

OEM_format_SMS_msg()

Description:

This function extracts **AEESMSMsg** from the SMS message. This routine is necessary because OEMs in some markets choose to decode into an alternate format that is more appropriate for the type of SMS supported.

Prototype:

```
void OEM_format_SMS_msg
      (
      uasms_teleservice_e_type ts,
      const uasms_client_bd_type * pData,
      AEESMSMsg * pm
      )
```

Parameters:

ts	Teleservice type.
pData	Input SMS data.
pm	Formatted SMS message.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of Functions

OEM_format_SMS_text()

Description:

This function formats SMS text given the buffer, length, and encoding.

Prototype:

```
AEESMSTextMsg * OEM_format_SMS_text
      (
      byte * pMsgData,
      int nMsgLen,
      uasms_user_data_encoding_e_type encoding,
      byte * pDest,
      int nSize
      )
```

Parameters:

pMsgData	Input SMS text string.
nMsgLen	Input SMS text string length.
encoding	Encoding type.
pDest	Buffer to hold the formatted SMS text.
nSize	Size of the destination buffer.

Return Value:

Final formatted SMS text if successful.

NULL if fails.

Comments:

None

See Also:

None

Return to the List of Functions

OEM_uasms_config_listeners()

Description:

This function registers OEM SMS notification functions to the AEE.

Prototype:

```
void OEM_uasms_config_listeners
      (
      uasms_message_listener_type pfnMsg,
      uasms_status_listener_type pfnStatus,
      uasms_event_listener_type pfnEvent
      )
```

Parameters:

pfnMsg	Message listener function pointer.
pfnStatus	Status listener function pointer.
pfnEvent	Event listener function pointer.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of Functions

OEM Socket Interface

The OEM Socket interface provides standard internet socket support. The semantics are modeled after BSD style sockets, but are strictly non-blocking. If an operation is unable to complete immediately, it should return an error code of AEE_NET_EWOULDBLOCK. BREW will then use OEMSocket_AsyncSelect() to indicate interest in an event for a particular socket. When the event occurs, call AEE_SocketEventOccurred, and BREW will in turn use OEMSocket_GetNextEvent() to determine which specific event has occurred for that socket.

The following functions are encouraged, but are optional and may return an error code of AEE_NET_EOPNOTSUPP:

OEMSocket_Accept()
OEMSocket_Listen()
OEMSocket_Shutdown()

List of functions

Functions in this interface include:

OEMSocket_Accept()
OEMSocket_AsyncSelect()
OEMSocket_Bind()
OEMSocket_Close()
OEMSocket_Connect()
OEMSocket_GetNextEvent()
OEMSocket_GetPeerName()
OEMSocket_GetSockName()
OEMSocket_Listen()
OEMSocket_Open()
OEMSocket_Read()
OEMSocket_Readv()
OEMSocket_RecvFrom()
OEMSocket_SendTo()
OEMSocket_Shutdown()
OEMSocket_Write()
OEMSocket_Writev()

The remainder of this section provides details for each function.

OEMSocket_Accept()

Description:

The accept function is used on listening sockets to respond when AEE_NET_READ_EVENT is asserted. The first backlog queued connection is removed from the queue, and bound to a new socket (as if you called OEMSocket_Open). The newly created socket is in the connected state. The listening socket is unaffected and the queue size is maintained (i.e. there is no need to call listen again.)

Prototype:

```
OEMCONTEXT OEMSocket_Accept
      (
      OEMCONTEXT sockd,
      int16 *err
      );
```

Parameters:

sockd	listening socket descriptor
err	error code (returned by operation)

Return Value:

On successful creation of a socket, this function returns socket descriptor which is OEM defined.

On error, returns AEE_NET_ERROR. Error specifics are returned via the err parameter.

Comments:

None

See Also:

OEMSocket_Listen()

Return to the List of functions

OEMSocket_AsyncSelect()

Description:

This function enables the events to be notified about through the asynchronous notification mechanism. The application specifies a bitmask of events in which it is interested, for which it will receive asynchronous notification by its application callback function.

This function also performs a real-time check to determine if any of the events have already occurred, and if so, it invokes the application callback.

Events OEMs need to support are:

AEE_NET_READ_EVENT: Socket is now available to read or connect

AEE_NET_WRITE_EVENT: Socket is now available for writing

AEE_NET_CLOSE_EVENT: Socket is being closed

Prototype:

```
int16 OEMSocket_AsyncSelect(OEMCONTEXT sockd, int32 interest_mask)
```

Parameters:

sockd	Socket descriptor.
mask	Bitmask of events to set (see above).

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is a failure.

Comments:

None

Side Effects:

The application will be notified using the callback function.

See Also:

None

Return to the List of functions

OEMSocket_Bind()

Description:

For all client sockets, this function attaches a local address and port value to the socket. If the call is not explicitly issued, the socket implicitly binds during calls to OEMSocket_Connect() or OEMSocket_SendTo().

NOTE: This function does not support binding a local IP address, but only a local port number.

The local IP address is assigned automatically by the sockets library. The function must receive (as a parameter) a valid socket descriptor, implying a previous successful call to OEMSocket_Open().

This function is synchronous, and therefore should not callback any notification functions.

Prototype:

```
int16 OEMSocket_Bind(OEMCONTEXT sockd, INAddr addr, INPort port)
```

Parameters:

sockd	Socket descriptor.
addr	Local address in network byte order.
port	Local port in network byte order.

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is a failure.

Comments:

None

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_Close()

Description:

This function performs a non-blocking close of a socket, and performs all necessary clean-up of data structures and frees the socket for re-use. For TCP, it initiates the active close for connection termination. After the TCP connection is complete, the socket resources may optionally not be freed. In this case, this function should return AEE_NET_ERROR and set **err** to AEE_NET_EWOULDBLOCK. The AEE libraries receive notification through OEMSocket_AsyncSelect() and call OEMSocket_Close() again to free the resources.

This function can be synchronous (returning anything other than ***err** set to AEE_NET_EWOULDBLOCK), or asynchronous as described above. For asynchronous incantations, AEE_SocketEventOccurred() should be called only if the AEE_NET_CLOSE_EVENT was registered with OEMSocket_AsyncSelect().

Prototype:

```
int16 OEMSocket_Close(OEMCONTEXT sockd)
```

Parameters:

sockd	Socket descriptor.

Return Value:

AEE_NET_SUCCESS, if successful.

AEE designated error code, if there is a failure. If the socket cannot be closed right away, the OEM may return an AEE_NET_EWOULDBLOCK error, indicating that it should be called at a later time (through an indication from OEMSocket_AsyncSelect().

Comments:

None

Side Effects:

Initiates active close for TCP connections.

See Also:

OEMSocket_AsyncSelect()

Return to the List of functions

OEMSocket_Connect()

Description:

For TCP, this function attempts to establish the TCP connection. Upon successful connection, it calls the socket callback function

This function is asynchronous and should call AEE_NetEventOccurred() if the connection attempt is completed or aborted, and the original error value was AEE_NET_SUCCESS.

Prototype:

```
int16 OEMSocket_Connect(OEMCONTEXT sockd, INAddr addr, INPort port)
```

Parameters:

sockd	Socket descriptor.
addr	Destination address in network byte order.
port	Destination port in network byte order.

Return Value:

AEE_NET_SUCCESS, if arguments are valid, and the connection process could be started. Thus, a return value of AEE_NET_SUCCESS does not indicate that the socket could be connected.

AEE designated error code, if there is a failure or an error can be detected immediately

Comments:

None

Side Effects:

This function starts the connection process for a socket. It may automatically call bind() on that socket.

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_GetNextEvent()

Description:

This function performs a real-time check to determine if any of the events of interest specified with the mask in OEMSocket_AsyncSelect() have occurred. It also clears any bits in the event mask that have occurred. The application must re-enable these events through a subsequent call to OEMSocket_AsyncSelect(). It may pass a pointer to a single socket descriptor to determine if any events have occurred for that socket. Alternatively, the application may set this pointer's referenced value to NULL (0).

NOTE: Do not confuse the referenced value of NULL (0) with a NULL pointer. NULL (0) is a pointer referencing an address with a value of 0 (zero), in which case the function returns values for the next available socket.

The next available socket's descriptor is to be placed in the socket descriptor pointer, and the function will return. If no sockets are available (no events have occurred across all sockets for that application) the pointer value remains NULL (original value passed in), and the function returns 0, indicating that no events have occurred.

Prototype:

```
int32 OEMSocket_GetNextEvent(OEMCONTEXT * sockd, int16 * err)
```

Parameters:

sockd	Socket descriptor.
err	Error code returned by operation.

Return Value:

Returns an event mask of the events that were asserted. A value of zero indicates that no events have occurred.

On passing a pointer whose value is NULL into the function for the socket descriptor (not to be confused with a NULL pointer), this function places the next available socket descriptor in *sockd and returns the event mask for that socket. If no sockets are available (no events have occurred across all sockets for that application) the pointer value remains NULL (original value passed in), and the function returns zero indicating that no events have occurred. On error, returns AEE_NET_ERROR.

Comments:

None

See Also:

OEMSocket_AsyncSelect()

Return to the List of functions

OEMSocket_GetPeerName()

Description:

This function returns the IP address and port of a connected peer.The address and port are in network byte order. This function is synchronous, and therefore must not call any notification functions.

Prototype:

```
int16 OEMSocket_GetPeerName
      (
      OEMCONTEXT sockd,
      INAddr *addr,
      INPort *port
      );
```

Parameters:

sockd	[in]	Socket descriptor
addr	[out]	IP address
port	[out]	Port number

Return Value:

On success, returns AEE_NET_SUCCESS. On error, returns one of the AEE designated error codes indicating reason for failure.

Comments:

None

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_GetSockName()

Description:

Returns the local IP address and port of a socket. The address and port will be in network byte order.

This function is synchronous, and therefore must not call any notification functions.

Prototype:

```
int16 OEMSocket_GetSockName
      (
      OEMCONTEXT sockd,
      INAddr *addr,
      INPort *port
      );
```

Parameters:

sockd	[in]	socket descriptor
addr	[out]	IP address
port	[out]	port number

Return Value:

On success, returns AEE_NET_SUCCESS.

On error, returns one of the AEE designated error codes indicating reason for failure.

Comments:

None

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_Listen()

Description:

Performs a passive open for connections, such that incoming connections may be subsequently accepted. The socket must be a TCP socket that has been bound to a local port. The backlog parameter indicates the maximum length for the queue of pending connections. If backlog is larger than the system maximum, it will be silently reduced to the system maximum.

Prototype:

```
int16 OEMSocket_Listen
      (
      OEMCONTEXT sockd,
      int16 backlog
      );
```

Parameters:

sockd	socket descriptor
backlog	maximum number of pending connections

Return Value:

On success, returns AEE_NET_SUCCESS.

On error, returns one of the AEE designated error codes indicating reason for failure.

Comments:

None

See Also:

OEMSocket_Accept()

Return to the List of functions

OEMSocket_Open()

Description:

This function creates a TCP or UDP socket and related data structures, and returns a reference to that socket.

Supported Types:

The OEM must support the SOCK_STREAM (TCP) and SOCK_DGRAM (UDP) data types. This function must be called to obtain a valid socket descriptor for use with all other socket-related functions. Before any socket functions can be used (such as I/O, asynchronous notification, and so on), this call must have successfully returned a valid socket descriptor.

This function is synchronous, and therefore should not callback any notification function.

Prototype:

```
OEMCONTEXT OEMSocket_Open(NetSocket type, int16 * err)
```

Parameters:

type	Socket type (see above).
err	Error code (returned by operation).

Return Value:

On successful creation of a socket, this function returns socket file descriptor that is a signed value greater than or equal to 0 (zero).

On error, returns AEE_NET_ERROR. Error specifics are returned via the err parameter.

Comments:

None

See Also:

None

Return to the List of functions

OEMSocket_Read()

Description:

This function reads the specified number of bytes into the buffer from the TCP transport. If the socket is connected but there is no data to read, the function should return AEE_NET_ERROR and set ***err** to AEE_NET_EWOULDBLOCK.

Prototype:

```
int32 OEMSocket_Read
      (
      OEMCONTEXT sockd,
      byte * buffer,
      uint32 nbytes,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
buffer	User buffer to which to copy data.
nbytes	Maximum number of bytes to be read from socket.
err	Error code (returned by operation).

Return Value:

On success, returns the number of bytes read, which could be less than the number of bytes specified.

On error, returns AEE_NET_ERROR, including when ***err** is AEE_NET_EWOULDBLOCK.

NOTE: A return of 0 (zero) indicates that an End-of-File (EOF) condition has occurred.

Comments:

None

See Also:

None

Return to the List of functions

OEMSocket_Readv()

Description:

This function provides the scatter read variant of OEMSocket_Read(), which allows the application to read into non-contiguous buffers. It reads the specified number of bytes into the buffer from the TCP transport.

Prototype:

```
int32 OEMSocket_Readv
      (
      OEMCONTEXT sockd,
      SockIOBlock iov[],
      uint16 iovcount,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
iov	Array of data buffers to which to copy data.
iovcount	Number of array items.
err	Error code (returned by operation).

Return Value:

On success, returns the number of bytes read, which could be less than the number of bytes specified.

On error, returns AEE_NET_ERROR.

NOTE: A return of 0 (zero) indicates that an End-of-File (EOF) condition has occurred.

Comments:

None

See Also:

SockIOBlock

Return to the List of functions

OEMSocket_RecvFrom()

Description:

This function reads **nbytes** bytes in the buffer from the UDP transport. It fills in the address and port pointers with values from who sent the data.

Prototype:

```
int32 OEMSocket_RecvFrom
      (
      OEMCONTEXT sockd,
      byte * buffer,
      uint32 nbytes,
      uint16 flags,
      INAddr * addr,
      INPort * port,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
buffer	User buffer into which to copy data.
nbytes	Number of bytes to be read.
flags	
addr	IP address, in network byte order.
port	Port number, in network byte order.
err	Error condition value.

Return Value:

Number of bytes read. Can be less than the number of bytes specified.

AEE_NET_ERROR, if there is an error.

Comments:

None

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_SendTo()

Description:

This function sends **nbytes** bytes in the buffer over the UDP transport.

Prototype:

```
int32 OEMSocket_SendTo
      (
      OEMCONTEXT sockd,
      const byte * buffer,
      uint32 nbytes,
      uint16 flags,
      INAddr addr,
      INPort port,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
buffer	User buffer from which to copy the data.
nbytes	Number of bytes to be written.
flags	send flags
addr	IP address, in network byte order.
port	Port number, in network byte order.
err	Error condition value.

Return Value:

Number of bytes written. Can be less than the number of bytes specified.

AEE_NET_ERROR, if there is an error.

Comments:

Currently supported flags are documented in AEENet.h.

For cdma2000, the URGENT flag corresponds to use of Short Data Burst (SDB) over the reversed enhanced access channel (R-EACH). The WAKEUP flag requests traffic channel origination immediately after the SDB attempt. This is necessary because the cdma2000 standard currently requires origination to take priority over any other access attempt, and thus the short data burst would either be prematurely aborted or undesirably delayed.

See Also:

INAddr

INPort

Return to the List of functions

OEMSocket_Shutdown()

Description:

Causes all or part of a full-duplex connection to be terminated gracefully.

If how is AEE_SHUTDOWN_RD, no more reads will be allowed. If how is AEE_SHUTDOWN_WR, no more writes will be allowed (AKA half-close). If how is AEE_SHUTDOWN_RDWR, both read and write will be disallowed.

Prototype:

```
int16 OEMSocket_Shutdown
      (
      OEMCONTEXT sockd,
      int32 how
      );
```

Parameters:

sockd	socket descriptor
how	dictates which portion(s) of the connection to shutdown

Return Value:

On success, returns AEE_NET_SUCCESS.

On error, returns one of the AEE designated error codes (including AEE_NET_WOULDBLOCK) indicating reason for failure.

Comments:

None

See Also:

None

Return to the List of functions

OEMSocket_Write()

Description:

This function sends a specified number of bytes in the buffer over the TCP transport.

Prototype:

```
int32 OEMSocket_Write
      (
      OEMCONTEXT sockd,
      const byte * buffer,
      uint32 nbytes,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
buffer	User buffer from which to copy data.
nbytes	Number of bytes to be written to socket.
err	Error condition value.

Return Value:

Number of bytes written. Can be less than the number of bytes specified.

AEE_NET_ERROR, if there is an error; places one of the following error condition values in **err**:

DS_EBADF: Invalid socket descriptor is specified.

DS_ENOTCONN: Socket not connected.

DS_ECONNRESET: TCP connection reset by server.

DS_ECONNABORTED: TCP connection aborted due to timeout or other failure.

DS_EIPADDRCHANGED: IP address changed, causing TCP connection reset.

DS_EPIPE: Broken pipe.

DS_EADDRREQ: Destination address required;

DS_ENETDOWN: Network subsystem unavailable.

DS_EFAULT: Application buffer no valid part of address space.

DS_EWOULDBLOCK: Operation would block.

Comments:

None

See Also:

None

Return to the List of functions

OEMSocket_Writev()

Description:

This function provides the gather write variant of the OEMSocket_Write() function, which allows the application to write from non-contiguous buffers. It sends a specified number of bytes in the buffer over the TCP socket.

Prototype:

```
int32 OEMSocket_Writev
      (
      OEMCONTEXT sockd,
      const SockIOBlock iov[],
      uint16 iovcount,
      int16 * err
      )
```

Parameters:

sockd	Socket descriptor.
iov	Array of data buffers from which to copy data.
iovcount	Number of array items.
err	Error condition value.

Return Value:

Written number of bytes. Can be less than the specified number of bytes.

AEE_NET_ERROR, if there is an error; places one of the following error condition values in **err**:

DS_EBADF: Invalid socket descriptor is specified.

DS_ENOTCONN: Socket not connected.

DS_ECONNRESET: TCP connection reset by server.

DS_ECONNABORTED: TCP connection aborted due to timeout or other failure.

DS_EIPADDRCHANGED: IP address changed, causing TCP connection reset.

DS_EPIPE: Broken pipe.

DS_EADDRREQ: Destination address required.

DS_ENETDOWN: Network subsystem unavailable.

DS_EFAULT: Application buffer no valid part of address space.

DS_EWOULDBLOCK: Operation would block.

Comments:

None

See Also:

SockIOBlock

Return to the List of functions

OEM Sound Interface

This section describes the Sound Interface functions that the AEE uses to provide a simple way to play multi-tones, generate vibration, and set the device volume.

List of functions

Functions in this interface include:

OEMSound_DeleteInstance()
OEMSound_GetLevels()
OEMSound_GetVolume()
OEMSound_Init()
OEMSound_NewInstance()
OEMSound_PlayFreqTone()
OEMSound_PlayTone()
OEMSound_PlayToneList()
OEMSound_SetDevice()
OEMSound_SetVolume()
OEMSound_StopTone()
OEMSound_StopVibrate()
OEMSound_Vibrate()

The remainder of this section provides details for each function.

OEMSound_DeleteInstance()

Description:

This function decreases ref count of underlying audio device and gives an opprtunity to restore the default settings when ref count goes to zero.

Prototype:

```
int OEMSound_NewInstance(AEESoundInfo * psi);
```

Parameters:

psi [in]: Sound info of the ISound object

Return Value:

SUCCESS if successful.

Error code otherwise.

Comments:

This function is called every time an instance of ISound is deleted.

See Also:

OEMSound_NewInstance()

Return to the List of functions

OEMSound_GetLevels()

Description:

This function returns the number of volume levels supported for the device/class pair. It is called when a user issues ISOUND_GetVolume() or ISOUND_SetVolume().

Prototype:

```
void OEMSound_GetLevels(AEESoundInfo * psi, void * pUser)
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_LevelCB is called with the result.

See Also:

None

Return to the List of functions

OEMSound_GetVolume()

Description:

This function issues a command to get the volume level of device/class pair.

Prototype:

```
void OEMSound_GetVolume(AEESoundInfo * psi, void * pUser)
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_VolumeCB is called with the result.

See Also:

None

Return to the List of functions

OEMSound_Init()

Description:

Maps device sound layer enumerations to AEE_SOUND enumerations.

Prototype:

```
void OEMSound_Init( void );
```

Parameters:

None

Return Value:

None

Comments:

This function is called only once during BREW initialization.

See Also:

None

Return to the List of functions

OEMSound_NewInstance()

Description:

This function increases ref count of underlying audio device. It returns current AEESoundInfo.

Prototype:

```
int OEMSound_NewInstance(AEESoundInfo * psi);
```

Parameters:

psi	[out]	Current sound info

Return Value:

SUCCESS if successful.

Error code otherwise.

Comments:

This function is called every time an instance of ISound is created.

See Also:

OEMSound_DeleteInstance()

Return to the List of functions

OEMSound_PlayFreqTone()

Description:

This function issues a command to play a specified pair of tone frequencies.

Prototype:

```
void OEMSound_PlayFreqTone
      (
      AEESoundInfo * psi,
      uint16 wHiFreq,
      uint16 wLoFreq,
      uint16 wDuration,
      void * pUser
      )
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
wHiFrteeq	High frequency of the specified pair of tone frequencies.
wLoFreq	Low frequency of the specified pair of tone frequencies.
wDuration	Duration of the tone play.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_PlayTone()

Description:

This function issues a command to play a specified tone.

Prototype:

```
void OEMSound_PlayTone
      (
      AEESoundInfo * psi,
      AEESoundToneData toneData,
      void * pUser
      )
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
toneData	Tone and duration to be played.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_PlayToneList()

Description:

This function issues a command to play a list of tones.

Prototype:

```
void OEMSound_PlayToneList
      (
      AEESoundInfo * psi,
      AEESoundToneData * pToneData,
      uint16 wDataLen,
      void * pUser
      )
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
pToneData	An array of tones and durations.
wDataLen	Number of tones and durations.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_SetDevice()

Description:

This function issues a command to set the sound output device.

Prototype:

```
void OEMSound_SetDevice(AEESoundInfo * psi, void * pUser)
```

Parameters:

psi	New sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_SetVolume()

Description:

This function issues a command to set the volume of a sound device/class pair.

Prototype:

```
void OEMSound_SetVolume
      (
      AEESoundInfo * psi,
      uint16 wLevel,
      void * pUser
      )
```

Parameters:

psi	Sound device information. See the *BREW API Reference Guide* for the definition of AEESoundInfo.
wLevel	New volume for the device.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_VolumeCB is called with the result.

See Also:

None

Return to the List of functions

OEMSound_StopTone()

Description:

This function issues a command to stop playing a single tone or playlist.

Prototype:

```
void OEMSound_StopTone(boolean bPlayList, void * pUser)
```

Parameters:

bPlayList	Flag that stops playing a tone list playback.
pUser	User data that is passed back to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_StopVibrate()

Description:

This function stops the current vibration. If the feature is not supported, it does not do anything.

Prototype:

```
void OEMSound_StopVibrate(void * pUser)
```

Parameters:

pUser	User data that is passed to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEMSound_Vibrate()

Description:

This function causes the device to vibrate for the specified amount to time. If the feature is not supported, it does not do anything.

Prototype:

```
void OEMSound_Vibrate(uint16 wDuration, void * pUser)
```

Parameters:

wDuration	Duration of vibration in milliseconds.
pUser	User data that is passed to the caller through the status callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESound_StatusCB is called with the status.

See Also:

None

Return to the List of functions

OEM SoundPlayer Interface

This section describes the multimedia SoundPlayer Interface functions that the AEE uses to provide controls for a basic sound player.

List of functions

Functions in this interface include:

OEMSoundPlayer_FastForward()
OEMSoundPlayer_GetTotalTime()
OEMSoundPlayer_Pause()
OEMSoundPlayer_Play()
OEMSoundPlayer_PlayRinger()
OEMSoundPlayer_Resume()
OEMSoundPlayer_Rewind()
OEMSoundPlayer_Stop()
OEMSoundPlayer_Tempo()
OEMSoundPlayer_Tune()

The remainder of this section provides details for each function.

OEMSoundPlayer_FastForward()

Description:

This function issues a command to fast forward an audio the indicated number of milliseconds.

Prototype:

```
void OEMSoundPlayer_FastForward(uint32 dwTime, void * pUser)
```

Parameters:

dwTime	Number of milliseconds to fast forward.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also trigger to call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_FFORWARD.

See Also:

OEMSoundPlayer_Pause()

OEMSoundPlayer_Resume()

OEMSoundPlayer_Rewind()

Return to the List of functions

OEMSoundPlayer_GetTotalTime()

Description:

This function issues a command to calculate the time of the indicated audio file.

Prototype:

```
void OEMSoundPlayer_GetTotalTime
      (
      AEESoundPlayerInput * pInfo,
      void * pUser
      )
```

Parameters:

pInfo	SoundPlayer source data.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_TimeCB is called with the command status and time information if the command was successfully performed. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

See Also:

None

Return to the List of functions

OEMSoundPlayer_Pause()

Description:

This function issues a command to pause an audio playback.

Prototype:

```
void OEMSoundPlayer_Pause(void * pUser)
```

Parameters:

pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_PAUSE.

See Also:

OEMSoundPlayer_Resume()

Return to the List of functions

OEMSoundPlayer_Play()

Description:

This function issues a command to play an audio file.

Prototype:

```
void OEMSoundPlayer_Play(AEESoundPlayerInput * pInfo, void * pUser)
```

Parameters:

pInfo	SoundPlayer source data.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_PlayCB is called with the current audio play. There must be at least one callback when this function executed with one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

If AEE_SOUNDPLAYER_SUCCESS is sent back, one of the following status codes must be sent back at some time before the next play:

AEE_SOUNDPLAYER_DONE

AEE_SOUNDPLAYER_ABORTED

For other operations, the current play also triggers a callback to inform the client of the play status.

See Also:

OEMSoundPlayer_FastForward()

OEMSoundPlayer_Pause()

OEMSoundPlayer_Resume()

OEMSoundPlayer_Rewind()

OEMSoundPlayer_Stop()

Return to the List of functions

OEMSoundPlayer_PlayRinger()

Description:

This function issues a command to play a MIDI ringer.

Prototype:

```
void OEMSoundPlayer_PlayRinger
      (
      AEESoundPlayerInfo * pInfo,
      uint16 wRepeatTimer,
      void * pUser
      )
```

Parameters:

pInfo	SoundPlayer source data.
wRepeatTimer	Time, in milliseconds, of the silence between the playbacks of the MIDI file. Playback is not repeated if this is set to 0 (zero).
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_PlayCB is called with the current audio play. There must be at least one callback when this function executed with one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

If AEE_SOUNDPLAYER_SUCCESS is sent back, one of the following status codes must be sent back at some time before the next play:

AEE_SOUNDPLAYER_DONE

AEE_SOUNDPLAYER_ABORTED

See Also:

OEMSoundPlayer_Stop()

Return to the List of functions

OEMSoundPlayer_Resume()

Description:

This function issues a command to resume MIDI or WebAudio playback.

Prototype:

```
void OEMSoundPlayer_Resume(void * pUser)
```

Parameters:

pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_RESUME.

See Also:

OEMSoundPlayer_Pause()

Return to the List of functions

OEMSoundPlayer_Rewind()

Description:

This function issues a command to rewind an audio playback the indicated number of milliseconds.

Prototype:

```
void OEMSoundPlayer_Rewind(uint32 dwTime, void * pUser)
```

Parameters:

dwTime	Number of milliseconds to rewind.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_REWIND.

See Also:

OEMSoundPlayer_FastForward()

OEMSoundPlayer_Pause()

OEMSoundPlayer_Resume()

Return to the List of functions

OEMSoundPlayer_Stop()

Description:

This function issues a command to stop an audio playback.

Prototype:

```
void OEMSoundPlayer_Stop(void * pUser)
```

Parameters:

pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_ABORTED.

See Also:

OEMSoundPlayer_Play()

Return to the List of functions

OEMSoundPlayer_Tempo()

Description:

This function issues a command to adjust the audio playback tempo.

Prototype:

```
void OEMSoundPlayer_Tempo(uint32 dwTempoFactor, void * pUser)
```

Parameters:

dwTempoFactor	New tempo value.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_TEMPO.

See Also:

None

Return to the List of functions

OEMSoundPlayer_Tune()

Description:

This function issues a command to adjust the an audio playback tune.

Prototype:

```
void OEMSoundPlayer_Tune(int32 dwTuneFactor, void * pUser)
```

Parameters:

dwTuneFactor	New tune value.
pUser	Client data to be sent back with the callback function.

Return Value:

None

Comments:

None

Side Effects:

AEESoundPlayer_StatusCB is called with the command status. It can be one of the following status codes:

AEE_SOUNDPLAYER_SUCCESS

AEE_SOUNDPLAYER_FAILURE

It must also call AEESoundPlayer_PlayCB with AEE_SOUNDPLAYER_TUNE.

See Also:

None

Return to the List of functions

OEM String Interface

This section describes the String Interface functions that the AEE uses to perform formatting and printing operations on strings.

List of functions

Functions in this interface include:

OEM_FloatToWStr()
OEM_GetCHType()
OEM_UTF8ToWStr()
OEM_vxprintf()
OEM_WStrLower()
OEM_WStrToFloat()
OEM_WStrToUTF8()
OEM_WStrUpper()

The remainder of this section provides details for each function.

OEM_FloatToWStr()

Description:

This function converts a floating point to a string.

Prototype:

```
boolean OEM_FloatToWStr(double v, AECHAR * psz, int nSize)
```

Parameters:

v	Floating point value.
psz	Destination string.
nSize	Size of destination string.

Return Value:

TRUE, if successful.

FALSE, if fails (if **psz** is NULL or **nSize** is zero or lesser).

Comments:

None

See Also:

None

Return to the List of functions

OEM_GetCHType()

Description:

This function returns the type (such as numeric or alpha) of a wide character.

Prototype:

```
TChType OEM_GetCHType(AECHAR ch)
```

Parameters:

ch Input character.

Return Value:

Type of character, if successful.

SC_UNKNOWN, if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_UTF8ToWStr()

Description:

This function converts a UTF8 string to a wide string.

Prototype:

```
boolean OEM_UTF8ToWStr
      (
      const byte * pSrc,
      int nLen,
      AECHAR * pDst,
      int nSize
      )
```

Parameters:

pSrc	Input string.
nLen	Length of input string.
pDst	Destination string.
nSize	Size in bytes of destination.

Return Value:

TRUE, if successful.

FALSE, if fails (if **pSrc** or pDst is NULL; if **nSize** is zero or lesser).

Comments:

None

See Also:

None

Return to the List of functions

OEM_vxprintf()

Description:

This function prints a formatted string to a buffer, or determines how much room to allocate for a formatted string. This size includes the NULL terminator.

Prototype:

```
int32 OEM_vxprintf
      (
      void * buf,
      uint32 f,
      const char * format,
      VA_LIST list
      )
```

Parameters:

buf	Buffer to write to. Use NULL to find out the size required for the string.
f	Maximum size of the buffer (0x7FFFFFFF). Use 0 (zero) with NULL **buf** to determine the size required for the string. The rest are reserved for flags.
format	String containing formatting.
list	Optional list of arguments based on formatting.

Return Value:

Number of bytes stored in **buf**.

Or, number of bytes required for the formatted string including the NULL terminator.

-1, if fails.

Comments:

None

See Also:

None

Return to the List of functions

OEM_WStrLower()

Description:

This function converts all upper case characters in a wide string to lower case.

Prototype:

```
void OEM_WStrLower(AECHAR * pszDest)
```

Parameters:

pszDest	Source/destination string.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_WStrToFloat()

Description:

This function converts a string to a floating point value.

Prototype:

```
double OEM_WStrToFloat(const AECHAR *psz)
```

Parameters:

psz	Input string.

Return Value:

Floating point, if successful.

0 (zero), if fails (if **psz** is NULL).

Comments:

None

See Also:

None

Return to the List of functions

OEM_WStrToUTF8()

Description:

This function converts a wide string to UTF8.

Prototype:

```
boolean OEM_WStrToUTF8
      (
      const AECHAR * pSrc,
      int nLen,
      byte * pDst,
      int nSize
      )
```

Parameters:

pSrc	Input string.
nLen	Length of input string.
pDst	Destination string.
nSize	Size in bytes of destination.

Return Value:

TRUE, if successful.

FALSE, if fails (**pSrc** or **pDst** is NULL; **nSize** is zero or lesser).

Comments:

None

See Also:

None

Return to the List of functions

OEM_WStrUpper()

Description:

This function converts all lower case characters in a wide string to upper case.

Prototype:

```
void OEM_WStrUpper(AECHAR * pszDest)
```

Parameters:

pszDest	Source/destination string.

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM Text Interface

This section describes the Text Interface input functions that the AEE uses. OEMs can modify the reference implementation to add more text input modes for foreign languges or other input methods.

List of functions

Functions in this interface include:

OEM_TextAddChar()
OEM_TextCreate()
OEM_TextDelete()
OEM_TextDraw()
OEM_TextEnumMode()
OEM_TextEnumModesInit()
OEM_TextGet()
OEM_TextGetCurrentMode()
OEM_TextGetCursorPos()
OEM_TextGetMaxChars()
OEM_TextGetModeString()
OEM_TextGetProperties()
OEM_TextGetRect()
OEM_TextGetSel()
OEM_TextKeyPress()
OEM_TextQueryModes()
OEM_TextQuerySymbols()
OEM_TextSet()
OEM_TextSetCursorPos()
OEM_TextSetEdit()
OEM_TextSetMaxChars()
OEM_TextSetProperties()
OEM_TextSetRect()
OEM_TextSetSel()
OEM_TextUpdate()

The remainder of this section provides details for each function.

OEM_TextAddChar()

Description:

This function adds or overwrites a character at the current cursor location in the specified text control.

Prototype:

```
void OEM_TextAddChar
      (
      OEMCONTEXT hTextField,
      AECHAR ch,
      boolean bOverStrike
      )
```

Parameters:

hTextField	Handle for the text control object.
ch	Wide character to be added to the text control.
bOverStrike	Overwrites the text at the cursor location.

Return Value:

None

Comments:

Overstrike is meaningful only if there is an insertion point rather than a selection and the insertion point is not at the very end of the text. If **hTextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextCreate()

Description:

This function creates a dynamic text control object. It uses the given rectangle **pRect** to create the text control. The **pIShell** and **pIDisplay** pointers are saved in the newly created context to be used by the text control for notification, drawing, and so forth.

Prototype:

```
OEMCONTEXT OEM_TextCreate
      (
      const IShell* pIShell,
      const IDisplay* pIDisplay,
      const AEERect * pRect
      )
```

Parameters:

pIShell	Pointer to the IShell interface object.
pIDisplay	Pointer to the IDisplay interface object.
pRect	Pointer to the rectangle specifying the bounds and location of the text control to be created.

Return Value:

OEMCONTEXT that can be used as the handle to the newly created text control, if successful.

NULL, if fails.

Comments:

If **pIShell**, **pIDisplay**, or **pRect** is NULL, the function fails.

See Also:

None

Return to the List of functions

OEM_TextDelete()

Description:

This function deletes a dynamic text control object. The text control must have been created successfully using OEM_TextCreate(). This function also frees memory and any other resources associated with this text control.

Prototype:

```
void OEM_TextDelete(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object to be deleted.

Return Value:

None

Comments:

If **hTextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextDraw()

Description:

This function draws the text associated with a given text control object on the screen. It also draws the associated items (such as Scroll Bar, Border, and Cursor) if necessary and if they are supported.

Prototype:

```
void OEM_TextDraw(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

None

Comments:

When the TP_PASSWORD property is set, please display a text buffer of **** in place of actual characters. You must maintain your original buffer of actual text.

When in multitap mode, please allow the selection to appear while the user presses the key. After the selection is committed to text, show only the * character.

If **hTextField** is NULL, the function perform no task and returns no errors.

See Also:

None

Return to the List of functions

OEM_TextEnumMode()

Description:

This function gets the next text enumeration mode.

Prototype:

```
boolean OEM_TextEnumMode(AEETextMode * pMode)
```

Parameters:

pMode	[OUT]	Pointer to the next text mode.

Return Value:

TRUE, if the next mode is valid.

FALSE, if already at the end of the list.

Comments:

If **pMode** is NULL, the function returns FALSE.

See Also:

None

Return to the List of functions

OEM_TextEnumModesInit()

Description:

This function initializes the enumeration mode. It does not have an associated handle for the text control object.

Prototype:

```
void OEM_TextEnumModesInit(void)
```

Parameters:

None

Return Value:

None

Comments:

None

See Also:

None

Return to the List of functions

OEM_TextGet()

Description:

This function gets the text associated with a given text control object, and returns a pointer to the text.

Prototype:

```
AECHAR* OEM_TextGet(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

Pointer to the text string in the text control, if successful.

NULL, if fails.

Comments:

If **hTextField** is NULL, the function returns NULL.

See Also:

None

Return to the List of functions

OEM_TextGetCurrentMode()

Description:

This function returns the current text mode of the text control specified by **hTextField**.

Prototype:

```
AEETextInputMode OEM_TextGetCurrentMode(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

Current text mode for the text control specified.

Comments:

None

See Also:

None

Return to the List of functions

OEM_TextGetCursorPos()

Description:

This function gets the absolute position of the cursor.

Prototype:

```
int32 OEM_TextGetCursorPos(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

The 0 based position of the cursor. For example, if you have the Text Hi and the cursor is given as |:

- |Hi would return 0.
- H|i would return 1.
- Hi| would return 2.

Comments:

None

See Also:

None

Return to the List of functions

OEM_TextGetMaxChars()

Description:

This function gets the maximum number of characters that can be added to the specified text control.

Prototype:

```
uint16 OEM_TextGetMaxChars(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

Maximum number of characters for the text control specified by **hTextField**.

Comments:

If **hTextField** is NULL, the function returns 0 (zero).

See Also:

None

Return to the List of functions

OEM_TextGetModeString()

Description:

This function returns the wide string corresponding to the current mode of the text control specified by **hTextField**. The mode strings are Multitap, Numbers, and Symbols.

Prototype:

```
void OEM_TextGetModeString
      (
      OEMCONTEXT hTextField,
      AECHAR* szBuf,
      uint16 len
      )
```

Parameters:

hTextField	[IN]	Handle for the text control object.
szBuf	[IN]	String corresponding to the mode of the text control.
len	[OUT]	Length of the mode string buffer.

Return Value:

None

Comments:

If **hTextField** or **szBuf** is NULL, or if **len** is one or less, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextGetProperties()

Description:

This function returns the properties of the text control, such as frame type, multiline, or rapid entry (like T9).

Prototype:

```
uint32 OEM_TextGetProperties(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

Property of the text control.

Comments:

Important properties are:

TP_MULTILINE, if set,the text control object is multiple line control.

TP_FRAME, if set, the text control object has a frame.

TP_RAPID_MODE, if set, the text control object is in rapid mode.

TP_PASSWORD, if set, the text control displays * characters in place of real characters.

It is safe to ignore the following:

TP_NODRAW, if set, the text control object does not draw itself

TP_NOUPDATE, if set, the text control object does not call IDISPLAY_Update when it is not needed

When using TP_PASSWORD in multitap mode, please allow the selection to appear while the user presses the key. After the selection is committed to text, show only the * character.

If **hTextField** is NULL, the function returns 0 (zero).

See Also:

None

Return to the List of functions

OEM_TextGetRect()

Description:

This function returns the rectangle corresponding to the bounds of this text control.

Prototype:

```
void OEM_TextGetRect(OEMCONTEXT hTextField, AEERect *pOutRect)
```

Parameters:

hTextField	Handle for the text control object.
pOutRect	Rectangle corresponding to the bounds of the text control.

Return Value:

None

Comments:

If **hTextField** or **pOutRect** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextGetSel()

Description:

This function gets the start and end locations for the selected text.

Prototype:

```
void OEM_TextGetSel
      (
      OEMCONTEXT hTextField,
      int * piSelStart,
      int * piSelEnd
      )
```

Parameters:

hTextField	[IN]	Handle for the text control object.
pSelStart	[OUT]	Start location of the text selection.
pSelEnd	[OUT]	Ending location of the text selection.

Return Value:

None

Comments:

If **htextField** is NULL and **piSelStart** is non-NULL, the first entry is set to 0 (zero).

If **htextField** is NULL and **piSelEnd** is non-NULL, the first entry is set to 0 (zero).

See Also:

None

Return to the List of functions

OEM_TextKeyPress()

Description:

This function handles the key press events in a text control. When a key is pressed while a text control is active, this function is invoked, passing information relating to the key that has been pressed. The OEM layer must handle the key event and process it appropriately.

Prototype:

```
boolean OEM_TextKeyPress
      (
      OEMCONTEXT hTextField,
      AEEEvent eCode,
      uint32 dwKeyCode,
      uint32 dwKeySyms
      )
```

Parameters:

hTextField	Handle for the text control object
eCode	Event code for the key event
dwKeyCode	Key code of the key that has been pressed
dwKeySyms	Not used.

Return Value:

The current text mode for the specified text control.

Comments:

None

See Also:

None

Return to the List of functions

OEM_TextQueryModes()

Description:

This function is invoked by the AEE to query the different text modes (such as T9 and MULTITAP) supported by the OEM layer. The AEE uses this information for two purposes:

- To populate the pop-up menu containing selections for the different modes supported.
- To notify the text control, using OEM_TextSetEdit(), of the mode selected by the user.

The OEM layer must populate the given data structure and return from this function.

Prototype:

```
int OEM_TextQueryModes(AEETextMode ** ppTextMode)
```

Parameters:

ppTextMode	On return, this contains a valid pointer to an array of AEE_TextMode containing information about the different modes supported by the OEM layer. The OEM layer must use the standard identifier OEM_TEXT_MODE_ SYMBOLS for symbols mode. Memory for this pointer must be allocated by the OEM.

Return Value:

Number of text modes supported by the OEM.

Comments:

The following is a brief description of how text modes are supported.

- The AEE platform invokes the OEM function OEM_TextQueryModes() to get information on the different text modes supported by the OEM.
- The information obtained above is used to populate the menu containing selection strings for the different modes.
- When the user selects a particular mode, the function OEM_TextSetEdit() is invoked and is passed the ID of the mode that has been selected. If the user has not changed the mode, the ID is set to OEM_TEXT_MODE_DEFAULT, informing the OEM layer to use the currently selected mode.
- The OEM layer must use the standard ID OEM_TEXT_MODE_SYMBOLS for supporting the symbol mode. All other IDs must be based out of OEM_TEXT_MODE_USER.

See Also:

None

Return to the List of functions

OEM_TextQuerySymbols()

Description:

This function gets the buffer **pszOut** with the symbols. The length of the buffer is specified by **size**.

Prototype:

```
uint16 OEM_TextQuerySymbols(AECHAR * pszOut, uint16 size)
```

Parameters:

pszOut	Symbols buffer.
size	Size of the buffer.

Return Value:

Number of symbols put in the query buffer.

If **pszOut** is NULL or if **size** is less than the number of OEM symbols, this function returns 0 (zero).

Comments:

If **pszOut** is NULL, the function returns 0 (zero).

See Also:

None

Return to the List of functions

OEM_TextSet()

Description:

This function sets the text of a given text control object. The text control must have been created successfully using OEM_TextCreate(). Once the text has been set, OEM_TextDraw() must be called to update the screen with the new text.

Prototype:

```
boolean OEM_TextSet
      (
      OEMCONTEXT hTextField,
      const AECHAR *pszText,
      int nChars)
```

Parameters:

hTextField	Handle for the text control object.
pszText	Text string to be set into the text control.
nChars	Number of characters to set.

Return Value:

None

Comments:

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextSetCursorPos()

Description:

This function gets the absolute position of the cursor.

Prototype:

```
int32 OEM_TextSetCursorPos(OEMCONTEXT hTextField, int32 nOffset)
```

Parameters:

hTextField	Handle for the text control object.
nOffset	Absolute offset where the cursor is to be moved.

Return Value:

None

Comments:

This function should move the cursor to the 0-based position of the cursor.

If nOffset is > the length of the text, the cursor should be placed after the text.

If nOffset is <= 0, the cursor should be placed at the beginning of the text.

For example, if you have the Text Hi and | represents the cursor:

nOffset = 0 |Hi
nOffset = -1 |Hi
nOffset = 1 H|i
nOffset = 2 Hi|
nOffset = 100 Hi|

See Also:

None

Return to the List of functions

OEM_TextSetEdit()

Description:

This function informs the text control whenever it goes in or out of focus. Typically, when the text control is in focus, the border and cursor are activated; when the text control goes out of focus, these items are de-activated. This function also informs the text control of the current text mode.

Prototype:

```
vvoid OEM_TextSetEdit
      (
      OEMCONTEXT hTextField,
      boolean bIsEditable,
      AEETextInputMode wmode
      )
```

Parameters:

hTextField	Handle for the text control object.
bIsEditable	Flag to indicate if the text control object is in focus (that is, it is editable).
wmode	Text input mode.

Return Value:

None

Comments:

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextSetMaxChars()

Description:

This function sets the maximum number of characters that can be added to the specified text control.

Prototype:

```
void OEM_TextSetMaxChars(OEMCONTEXT hTextField, uint16 wMaxChars)
```

Parameters:

hTextField	Handle for the text control object.
wMaxChars	New maximum number of characters in this text control.

Return Value:

None

Comments:

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextSetProperties()

Description:

This function sets the properties of the text control, such as frame type, multiline, rapid entry (such as T9), or a combination of properties.

Prototype:

```
void OEM_TextSetProperties(OEMCONTEXT hTextField, uint32 dwProperties)
```

Parameters:

hTextField	Handle for the text control object.
dwProperties	Properties (TP_FRAME, TP_MULTILINE, TP_RAPID_MODE, or a combination).

Return Value:

None

Comments:

Important properties are:

TP_MULTILINE, if set, text control object is multiple line control

TP_FRAME, if set, text control object has a frame

TP_RAPID_MODE, if set, text control object is in rapid mode

TP_PASSWORD, if set, text control displays * characters in place of real characters

It is safe to ignore the following properties:

TP_NODRAW, if set, text control object does not draw itself

TP_NOUPDATE, if set, text control object does not call IDIPLAY_Update when not needed

When using TP_PASSWORD in multitap mode, please allow the selection to appear while the user presses the key. After the selection is committed to text, show only the * character.

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextSetRect()

Description:

This function returns the rectangle corresponding to the bounds of this text control.

Prototype:

```
void OEM_TextSetRect(OEMCONTEXT hTextField, const AEERect *pInRect)
```

Parameters:

hTextField	Handle for the text control object.
pInRect	New bounds for the text control.

Return Value:

None

Comments:

If **htextField** or **pInRect** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextSetSel()

Description:

This function sets the start and end locations for the text selection.

Prototype:

```
void OEM_TextSetSel(OEMCONTEXT hTextField, int iSelStart, int iSelEnd)
```

Parameters:

hTextField	Handle for the text control object.
iSelStart	Start location of the text selection.
iSelEnd	Ending location of the text selection.

Return Value:

None

Comments:

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

OEM_TextUpdate()

Description:

This function draws the text associated with a given text control object on the screen if the text control is modified.

Prototype:

```
void OEM_TextUpdate(OEMCONTEXT hTextField)
```

Parameters:

hTextField	Handle for the text control object.

Return Value:

None

Comments:

If **htextField** is NULL, the function simply returns.

See Also:

None

Return to the List of functions

Data Types

This section describes the data types used by the BREW OEM API functions. These data types define the format and content of the data that is passed by applications to the BREW functions and received by the applications. Type definitions for the BREW data structures are contained in the BREW header files. Some data structures are specific to a particular BREW interface and are contained in the header files for those interfaces. Other data structures are used by more than one interface and are found in the files AEE.h and AEEError.h. The description of each BREW function contains links to the descriptions of all relevant data structures.

BREW data structures are of three main types:

- **Structures and Unions:** Some BREW functions take pointers to structures as input parameters. To use such a function, populate an instance of a structure and pass a pointer to the instance when calling the function. Other BREW functions return pointers to structures as output. This section describes each field in each of the BREW structures.
- **Enumerated Types:** Some BREW variables and structure members take values from a finite set defined by the C typedef enum construct. For example, the font types supported by text-drawing functions are specified with an enumerated-type definition. This section describes each value of each enumerated type.
- **Constant Definitions:** The BREW functions use 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. 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. This section describes each set of related constants.

List of data structures

Data structures in this interface include:

CMediaQCP
Configuation Parameters
CtlAddItem
CtlValChange
FileAttrib
FileInfo
GSMSMSEncodingType
GSMSMSMsg
GSMSMSMsgType
GSMSMSRawMsg
GSMSMSStatusType
GSMSMSStorageType
I3D_Events
IDC_COMMAND_RESERVED
IDIB
INAddr
INPort
ITransform Properties
NativeColor
NetSocket
NetState
OEMAppEvent
oemLogType
PFNCBCANCEL
PFNDLTEXT
PFNMEDIANOTIFY
Q12 Fixed Point Format
Q14 Fixed Point Format
Q16 Fixed Point Format
Q3D File Format
AEEObjectHandle
PFNPOSITIONCB
PFNRINGEREVENT
PFNSIONOTIFY
PhoneState
RGBVAL
TAPIStatus
Tile Properties
Tile Map Properties

The remainder of this section provides details for each function.

AECHAR

Description:

AECHAR is BREW defined data type for wide strings.

Definition:

```
typedef uint16 AECHAR;
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEE Events

Description:

The defined AEE events that can be received by an applet or control. For each event, the **wParam** and **dwParam** parameters, if any, that are passed to the applet or control are given.

Definition:

The following tables list the event codes and key codes supported by BREW.

Key Codes are received with EVT_KEY, EVT_KEY_PRESS, EVT_KEY_RELEASE, and EVT_KEY_HELD events.

Event codes

Event code	Description	Parameters
EVT_ALARM	Alarm event.	wParam = Alarm Code, dwParam = 0.
EVT_APP_BROWSE_FILE	Called after EVT_APP_START.	
EVT_APP_BROWSE_URL	Called after EVT_APP_START	dwParam = (const AECHAR * pURL).
EVT_APP_CONFIG	Alternate application start. Configuration screen shown.	wParam = 0, dwParam = 0.
EVT_APP_HIDDEN_CONFIG	Alternate application start. Configuration screen hidden.	wParam = 0, dwParam = 0.
EVT_APP_NO_CLOSE	Application should not be closed	
EVT_APP_NO_SLEEP	Application is working - called after long periods of non-idle application	
EVT_APP_RESUME	Application resume.	wParam = 0, dwParam = 0.
EVT_APP_START	Application start.	wParam = 0, dwParam = const char *(arguments).
EVT_APP_STOP	Application stop.	wParam = 0, dwParam = boolean *(flag to indicate if application wants to close now or later. Default is now.
EVT_APP_SUSPEND	Application suspend.	wParam = 0, dwParam = 0.

Event code	Description	Parameters
EVT_BUSY	Sent to application to determine if the application can be suspended or stopped. Application must return TRUE if it does not want to be suspended or stopped. Typically, applications return FALSE.	wParam = 0, dwParam = 0.
EVT_CB_COPY	Copy request -	**dwParam** = (const char *) const char * indicating the preferred format, NULL for copy all
EVT_CB_CUT	Cut request -	**dwParam** = (const char *) const char * indicating the preferred format, NULL for cut all
EVT_CB_PASTE	Paste request -	no parameters
EVT_CHAR	Character event.	wParam = AECHAR code of character, dwParam = bitflags for modifier keys.
EVT_COMMAND	Application custom controls event.	wParam = user command ID, dwParam = user data.
EVT_COPYRIGHT_END	Dialog event: Copyright dialog ended.	wParam = DialogID, dwParam = IDialog *.
EVT_CTL_ADD_ITEM	Message interface to add item.	wParam = 0, dwParam = CtlAddItem *.
EVT_CTL_CHANGING	Change control event.	wParam = 0, dwParam = CtlValChange*.
EVT_CTL_MENU_OPEN	Sent by ITextCtl before menu is activated.	wParam = 0, dwParam = IMenuCtl *.
EVT_CTL_SEL_CHANGED	Sent by IMenuCtl when selection has changed	wParam - selection ID, dwParam == IMenuCtl *
EVT_CTL_SET_TEXT	Set text control event.	wParam = ID, dwParam = if ID is not zero, then resource file, else text.
EVT_CTL_SET_TITLE	Set control title event.	wParam = ID, dwParam = if ID is not zero, then resource file, else text.
EVT_CTL_SKMENU_PAGE_FULL	Sent by IMenuCtl when SK menu page is full	dwParam == IMenuCtl *
EVT_CTL_TAB	Application tab event.	wParam = 0-left, 1-right, dwParam = pointer to the control.
EVT_CTL_TEXT_MODE CHANGED	Sent by ITextCtl when input mode was changed	

Event code	Description	Parameters
EVT_DIALOG_END	Dialog event: Dialog completed normally.	wParam = DialogID, dwParam = IDialog *.The dwPram has 1 for a "Yes" response and 2 for "No" response from user.
EVT_DIALOG_INIT	Dialog Event: Controls created, pre-init values, flags, and other items.	wParam = Dialog ID, dwParam = IDialog *.
EVT_DIALOG_START	Dialog event: Dialog opening.	wParam = DialogID, dwParam = IDialog *.
EVT_FLIP	Device-specific event: Sent to application when flip-type (clam-shell) device is opened or closed.	wParam = TRUE if open, FALSE if closed; dwParam = 0.
EVT_KEY	Key handling event.	wParam = key code (see Key codes table), dwParam = bitflags for modifier keys.
EVT_KEY_HELD	Key held event. The hold time is device-specific. Not supported in Emulator.	wParam = key code (see Key codes table), dwParam = bitflags for modifier keys.
EVT_KEY_PRESS	Keypress event.	wParam = key code (see Key codes table), dwParam = bitflags for modifier keys.
EVT_KEY_RELEASE	Key release event.	wParam = key code (see Key codes table), dwParam = bitflags for modifier keys.
EVT_KEYGUARD	Device-specific event: Sent to application when device keypad is locked.	wParam = TRUE if keyguard is on, FALSE otherwise; dwParam = 0.
EVT_LOCKED	Device-specific event: Sent to application when device user interface is locked.	wParam = TRUE if locked, FALSE otherwise; dwParam = 0.
EVT_MOD_LIST_CHANGED	List of modules changed. May be sent while application suspended!	
EVT_NOTIFY	BREW-generated notification or application-registered notification event.	wParam = 0, dwParam = AEENotify *.
EVT_UPDATECHAR	Character update event.	wParam = AECHAR code of character, dwParam = bitflags for modifier keys.
EVT_USER	Start of application/user-defined events.	Private to application.

Members:

None

Comments

The user-defined events start from EVT_USER.

See Also:

None

Return to the List of data structures

AEE ITextCtl Properties

Description:

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	(Deprecated)
TP_RAPID_MODE	Supports Rapid Entry and uses as default
TP_NODRAW	Disables all drawing by the control
TP_NOUPDATE	Disables wasteful IDISPLAY_Update calls
TP_PASSWORD	Displays ***, manages correct buffer chars
TP_INTEGRALHEIGHT	If set this forces the rectangle of the TextCtl to be of an even height with respect to the character height. Basically there will be no left over space and the text will fit naturally into the text control. Rather than showing 1.5 lines of text it will show either 1 or 2. It will round to the nearest line height and snap to it.
TP_FIXSETRECT	Actual height more closely represents requested height.

Members:

None

Comments

None

See Also:

ITEXTCTL_SetProperties()

ITEXTCTL_GetProperties()

Return to the List of data structures

AEE Static Properties

Description:

The properties defined for IStatic Interface.

Definition:

ST_CENTERTEXT	Center Text
ST_CENTERTITLE	Center Title
ST_NOSCROLL	Do not scroll text
ST_TEXTALLOC	Text allocated on heap - dialog takes responsibility of freeing it.
ST_TITLEALLOC	Title allocated on heap - dialog takes responsibility of freeing it.
ST_MIDDLETEXT	Text is drawn in the middle of the screen
ST_UNDERLINE	Underline the title
ST_ICONTEXT	Text is IImage *
ST_ASCII	Text is single-byte
ST_ENABLETAB	Generate EVT_CTL_TAB when at top or bottom
ST_ENABLE_HLGHT	Highlights the static, if it has focus

Members:

None

Comments

None

See Also:

ISTATIC_SetProperties()

ISTATIC_GetProperties()

Return to the List of data structures

AEE_ADDR_RECID_NULL

Description:

This constant defines a NULL record ID in the BREW Address Book interface.

Definition:

```
#define AEE_ADDR_RECID_NULL 0xffff
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEE3DColor

Description:

This STRUCT defines the color format used in I3D API.

Definition:

```
typedef struct {
      uint8 r,g,b,a;
      } AEE3DColor;
```

Members:

r: RED value.

g: GREEN value.

b: BLUE value.

a: ALPHA value.

Comments:

Alpha blending is not supported in this release. The default color scheme is 5-6-5 for (r, g,b).

See Also:

None

Return to the List of data structures

AEE3DCoordinateTransformType

Description:

NOTE: This data item is currently not supported.

This ENUM defines the coordinate transformation type.

Definition:

```
typedef enum
      {
      AEE3D_COORDINATE_TRANSFORM_NONE =0,
      AEE3D_COORDINATE_TRANSFORM_SCREENMAP,
      AEE3D_COORDINATE_TRANSFORM_PROJECTION,
      AEE3D_COORDINATE_TRANSFORM_MODELVIEW
      }AEE3DCoordinateTransformType;
```

Members:

AEE3D_CORDINATE_TRANSFORM_NONE: Vertices are given in screen coordinates. No transformation is necessary.

AEE3D_CORDINATE_TRANSFORM_SCREENMAP: Screen mapping is made.

AEE3D_CORDINATE_TRANSFORM_PROJECTION: Screen mapping is made after projection.

AEE3D_CORDINATE_TRANSFORM_MODELVIEW: Vertices go through the entire transformation pipeline: model-view, projection, and screen mapping.

Comments:

None

See Also:

I3D_GetCoordTransformMode()

I3D_SetCoordTransformMode()

Return to the List of data structures

AEE3DCullingType

Description:

This ENUM defines which triangles should be discarded before they are rendered. By default, triangles with vertices arranged in counterclockwise rotation will be visible. A counterclockwise rotation indicates front-facing. A clockwise rotation is considered back facing.

Definition:

```
typedef enum
      {
      AEE3D_CULLING_BACK_FACING = -1,
      AEE3D_CULLING_FRONT_FACING = 1
      } AEE3DCullingType;
```

Members:

AEE3D_CULLING_BACK_FACING: All triangles with vertices arranged in clockwise rotation will be visible. And triangles that are back facing will be discarded when they are drawn.

AEE3D_CULLING_FRONT_FACING: All triangles with vertices arranged in counter-clockwise rotation will be visible. And triangles that are front facing will be discarded when they are drawn.

Comments:

None

See Also:

I3D_SetCullingMode()

I3D_GetCullingMode()

Return to the List of data structures

AEE3DEventNotify

Description:

This STRUCT defines a 3D event notifier.

Definition:

```
typedef struct _AEE3DEventNotify
      {
         I3D*           p3D;
         int16          nEventType;
         int16          nErrorCode;
         void*          pData;
      } AEE3DEventNotify;
```

Members:

p3D	Pointer to I3D instance associated with the event.
InEventType	Type of event that occurred.
InErrorCode	The error code.
pData	Pointer to event data. Could be dropped frame, status, and so on.

Comments:

Your registered event notifier will be passed this event notify type.

See Also:

I3D_RegisterEventNotify()

I3D_StartFrame()

Return to the List of data structures

AEE3DLight

Description:

This STRUCT defines a 3D light type.

Definition:

```
typedef struct _AEE3DLight {
      AEE3DLightType type;
      AEE3DColor color;
      AEE3DPoint direction;
      } AEE3DLight;
```

Members:

type: The type of light to use.

color: The color of the light.

direction: The direction of the light.

Comments:

None

See Also:

I3D_SetLight()

I3D_SetLightingMode()

AEE3DLightType

Return to the List of data structures

AEE3DLightingMode

Description:

This ENUM defines constants that determine which lighting mode to use.

Definition:

```
typedef enum
      {
      AEE3D_LIGHT_MODE_DIFFUSED = 0,
      AEE3D_LIGHT_MODE_COLOR_DIFFUSED,
      AEE3D_LIGHT_MODE_DIFFUSED_COLOR_SPECULAR,
      AEE3D_LIGHT_MODE_COLOR_DIFFUSED_COLOR_SPECULAR
      AEE3D_LIGHT_MODE_NONE
      }AEE3DLightingMode;
```

Members:

AEE3D_LIGHT_MODE_DIFFUSED: lighting will be white diffused only.

AEE3D_LIGHT_MODE_COLOR_DIFFUSED: lighting will be color diffused.

AEE3D_LIGHT_MODE_DIFFUSED_COLOR_SPECULAR: lighting will be white diffused with color specular light. Both lights are in the same direction.

AEE3D_LIGHT_MODE_COLOR_DIFFUSED_COLOR_SPECULAR: lighting will be color diffused with color specular light. Each light can have a different direction.

Comments:

AEE3D_LIGHT_MODE_DIFFUSED_COLOR_SPECULAR is the default.

AEE3D_LIGHT_MODE_COLOR_DIFFUSED_COLOR_SPECULAR is the most CPU intensive.

See Also:

I3D_SetLight()

I3D_SetLightingMode()

Return to the List of data structures

AEE3DLightType

Description:

This ENUM defines which will determine what light to use.

Definition:

```
typedef enum
      {
      AEE3D_LIGHT_DIFFUSED = 0,
      AEE3D_LIGHT_SPECULAR

      }AEE3DLightType;
```

Members:

AEE3D_LIGHT_DIFFUSED: define diffused light.

AEE3D_LIGHT_SPECULAR: define specular light.

Comments:

None

See Also:

I3D_SetLight()

I3D_SetLightingMode()

Return to the List of data structures

AEE3DMaterial

Description:

This.

Definition:

```
typedef struct _AEE3DMaterial {
   AEE3DColor color;
   uint8 shininess;
   uint8 emissive;
} AEE3DMaterial;
```

Members:

color: The color of the material

shininess: The shininess value (0-255)

emissive: The emissive value (0-255)

Comments:

None

See Also:

I3D_SetMaterial()

Return to the List of data structures

AEE3DMatrixMode

Description:

This ENUM defines constants that determine which matrix to use when pushing or popping a matrix. You should set this mode before calling push and pop matrix.

Definition:

```
typedef enum
      {
      AEE3D_MATRIX_MODE_MODELVIEW   = 0
      }   AEE3DMatrixMode;
```

Members:

AEE3D_MATRIX_MODE_MODELVIEW: Use the model view matrix when pushing and popping a matrix.

Comments:

None

See Also:

I3D_PushMatrix()

AEE3DMatrixMode()

Return to the List of data structures

AEE3DModelData

Description:

This STRUCT defines a 3D model that will be used by the I3DModel functions.

Definition:

```
typedef struct _AEE3DModelData
{
    uint8    model_index;
    uint8    num_material;
    uint8    num_texture;
    uint8    num_segment;
    uint16   num_vertex;
    uint16   reserved;
    uint16   num_poly;
    uint16   extra;
    AEE3DPoint* pOrignalVertex_tbl;
    AEE3DVertex* pVertex_tbl;
    AEE3DModelPoly* pPoly_tbl;
    AEE3DColor* pMaterialColor_tbl;
    AEE3DModelSegment* pSegment_tbl;
    AEE3DTransformMatrix* pModelViewMatrixStack;
    void* pReserved;
    AEE3DTexture** pTexture_tbl;
    void* extra_features;
    unsigned char* pAlloc_mem;
} AEE3DModelData;
```

Members:

model_index	The model index value
num_material	Number of materials
num_texture	Number of textures
num_segment	Number of segments
num_vertex	Number of vertices
reserved	reserved field
num_poly	Number of polygons
extra	For future use
pOrignalVertex_tbl	Pointer to original vertex table (list of vertices)
pVertex_tbl	Pointer to vertex table (list of vertices)
pPoly_tbl	Pointer to polygon table (list of polygons)
pMaterialColor_tbl	Pointer to material color table (list of material colors)
pSegment_tbl	Pointer to segment table (list of segments)
pModelViewMatrixStack	Pointer to a list of Model View transformation matrices. Index range (0,num_segment-1) contains the segment's transformation matrices.
pReserved	for internal use only.

pTexture_tbl	Pointer to an array of texture bitmaps in IDIB format
extra_features	currently not used.
pAlloc_mem	Pointer to allocated memory pool for Model, Data block transfer are faster if data is continuous for firmware acceleration.

Comments:

The struct is designed to hold a 3D model in the Q3D format.

See Also:

Q3D File Format

AEE3DModelSegment

AEE3DModelPoly

Return to the List of data structures

AEE3DModelPoly

Description:

This STRUCT defines a polygon (triangle) that will be used by the I3DModel functions.

Definition:

```
typedef struct _AEE3DModelPoly {
         uint16 vi0;
         uint16 vi1;
         uint16 vi2;
         uint16 attr;
         int16  pnorm_x;
         int16  pnorm_y;
         int16  pnorm_z;
      } AEE3DModelPoly;
```

Members:

vi0: vertex index 0

vi1: vertex index 1

vi2: vertex index 2

attr: polygon attribute vector which indicates:

bits 0-7: color index

bits 8-10: texture index. This is the index value into the pTexture_tbl. The Texture_tbl is defined in AEE3DModelData structure.

bits 11-12: texture method (00: not textured, 01: replace, 10: mixed, 11: blending)

bits 13: flag for shading (0: no shading)

bits 14: flag for double draw (0: no double draw)

bits 15: flag for using polygon level feature (0: use segmental features)

pnorm_x:

pnorm_y:

pnorm_z: The polygon normal

Comments:

None

See Also:

AEE3DModelSegment

Return to the List of data structures

AEE3DModelSegment

Description:

This STRUCT defines a segment that will be used by the I3DModel functions.

Definition:

```
typedef struct _AEE3DModelSegment {
      uint16 num_vertex;
      uint16 vertex_offset;
      uint16 reserved;
      uint16 num_face;
      uint16 face_offset;
      uint16 attr;
      uint16 material_attr;
      } AEE3DModelSegment;
```

Members:

num_vertex: number of vertices in the segment

vertex_offset: offset of vertex in the vertex buffer.

reserved: reserved for future use.

num_face: number of polygons.

face_offset: offset in the **pPoly_tbl** defined in AEE3DModelData. This offset will be where the first polygon starts in the **Poly_tbl** for this segment.

attr: segment level attributes.

material_attr: material attributes.

The attribute values here have the same definition as that in AEE3DModelPoly. It indicates:

bits 0-7: color index.

bits 8-10: texture index This is the index value into the **pTexture_tbl**. The Texture_tbl is defined in AEE3DModel structure.

bits 11-12: texture method (00: not textured, 01: replace, 10: mixed, 11: blending).

bits 13: flag for shading (0: no shading).

bits 14: flag for double draw (0: no double draw).

bits 15: flag for using polygon level feature (0: use segmental features).

The Material attributes values are as follows:

bits 0-7: shininess (0-255).

bits 8-15: emissive (0-255).

Comments:

None

See Also:

AEE3DModelPoly

Return to the List of data structures

AEE3DPoint

Description:

This STRUCT defines a 3D point type.

Definition:

```
typedef struct _AEE3DPoint {
      int32 x;
      int32 y;
      int32 z;
      } AEE3DPoint;
```

Members:

x: X coordinate.

y: Y coordinate.

z: Z coordinate.

Comments:

The coordinates are in Q16 format.

See Also:

None

Return to the List of data structures

AEE3DPoint16

Description:

This STRUCT defines a 3D 16-bit point type.

Definition:

```
typedef struct _AEE3DPoint16 {
int16 x;
int16 y;
int16 z;
} AEE3DPoint16;
```

Members:

x: X coordinate.

y: Y coordinate.

z: Z coordinate.

Comments:

The coordinates are in Q16 format.

See Also:

None

Return to the List of data structures

AEE3DPrimitiveType

Description:

This ENUM defines constants that determine the primitive type to use with vertex arrays.

Definition:

```
typedef enum
      {
      AEE3D_TRIANGLE,
      AEE3D_TRIANGLE_FAN,
      AEE3D_TRIANGLE_STRIP,
      } AEE3DPrimitiveType;
```

Members:

AEE3D_TRIANGLE	Vertices in the array form triangles.
AEE3D_TRIANGLE_FAN	Vertices in the array form a triangle fan.
AEE3D_TRIANGLE_STRIP	Vertices in the array form a triangle strip.

Comments:

None

See Also:

I3D_CalcVertexArrayNormal()

I3D_CalcVertexArrayColor()

Return to the List of data structures

AEE3DRenderType

Description:

This ENUM defines 3D rendering types. It determines how each triangle will be filled.

Definition:

```
typedef enum
      {
      AEE3D_RENDER_FLAT_SHADING = 0,
      AEE3D_RENDER_FLAT_TEXTURE_FAST_SHADING,
      AEE3D_RENDER_FLAT_TEXTURE_SHADING,
      AEE3D_RENDER_SMOOTH_SHADING,
      AEE3D_RENDER_SMOOTH_TEXTURE_FAST_SHADING,
      AEE3D_RENDER_SMOOTH_TEXTURE_SHADING,
      AEE3D_RENDER_TEXTURE_REPLACE,
      }AEE3DRenderType;
```

Members:

AEE3D_RENDER_FLAT_SHADING: Each triangle is filled with the same color as the color of the first vertex.

`AEE3D_RENDER_FLAT_TEXTURE_FAST_SHADING`: Flat shading with texture colors averaged with surface colors.

`AEE3D_RENDER_FLAT_TEXTURE_SHADING`: Flat shading with texture colors blended with surface colors. (higher quality, slower performance)

AEE3D_RENDER_SMOOTH_SHADING: Each triangle is filled with color interpolated across the three vertices.

`AEE3D_RENDER_SMOOTH_TEXTURE_FAST_SHADING`: Smooth shading with texture colors averaged with surface colors.

`AEE3D_RENDER_SMOOTH_TEXTURE_SHADING,`: Smooth shading with texture colors blended with surface colors. (higher quality, slower performance)

AEE3D_RENDER_TEXTURE_REPLACE: Texture is as a decal.

Comments:

Render type significantly influences rendering performance. Flat shading is the fastest whereas smooth texture blending is the slowest.

See Also:

I3D_GetRenderMode()

I3D_SetRenderMode()

Return to the List of data structures

AEE3DRotateType

Description:

This ENUM defines what axis to use when calculating a rotational matrix.

Definition:

```
typedef enum
      {
      AEE3D_ROTATE_X,
      AEE3D_ROTATE_Y,
      AEE3D_ROTATE_Z
      }AEE3DRotateType;
```

Members:

AEE3D_ROTATE_X: rotate about the X axis.

AEE3D_ROTATE_Y: rotate about the Y axis.

AEE3D_ROTATE_Z: rotate about the Z axis.

Comments:

None

See Also:

I3DUtil_GetRotateMatrix()

I3DUtil_GetRotateVMatrix()

Return to the List of data structures

AEE3DTexture

Description:

This STRUCT defines the texture for a 3D Model. All texture images need to be converted into an IBitmap format. An application needs to decode and palettize (if necessary) these images and set the decoded raw pixel image (8-bits per pixel arranged row-by-row) to the pixel_map component of IBitmap, and the palette to the palette. The width and height of the pixel map need to be a power of 2, and no greater than 256. Color depth of the palette ranges from 1-8.

Definition:

```
typedef struct _AEE3DTexture {
      AEE3DTextureType type;
      AEE3DTextureSamplingType SamplingMode;
      AEE3DTextureWrapType Wrap_s;
      AEE3DTextureWrapType Wrap_t;
      uint32 BorderColorIndex;
      IBitmap *pImage;
} AEE3DTexture;
```

Members:

Type	The texture type
SamplingMode	Sampling type
Wrap_s	Wrap mode in horizontal direction
Wrap_t	Wrap mode in vertical direction
BorderColorIndex	Index into the color palette for texture border color
*pImage	Pointer to IBitmap structure

Comments:

Image decoding should be done at program initialization.

See Also:

IBitmap Interface

AEE3DTextureSamplingType

AEE3DTextureWrapType

AEE3DColor

Return to the List of data structures

AEE3DTextureSamplingType

Description:

This ENUM defines texture sampling types. Texture sampling refers to how to determine which texel (or texels) to use for a given fragment.

Definition:

```
typedef enum
      {
      AEE3D_TEXTURE_SAMPLING_NEAREST = 0
      }AEE3DTextureSamplingType;
```

Members:

AEE3D_TEXTURE_SAMPLING_NEAREST: Texture is sampled from the nearest neighbor.

Comments:

Other sampling methods may be supported in future releases.

See Also:

I3D_GetTexture()

I3D_SetTexture()

Return to the List of data structures

AEE3DTextureType

Description:

This ENUM defines texture types.

Definition:

```
typedef enum
      {
      AEE3D_TEXTURE_DIFFUSED,
      }AEE3DTextureType;
```

Members:

AEE3D_TEXTURE_DIFFUSED: The diffused texture.

Comments:

Other texture types may be supported in future releases.

See Also:

I3D_SetTexture()

I3D_GetTexture()

Return to the List of data structures

AEE3DTextureWrapType

Description:

This ENUM defines the texture wrapping modes. The wrapping mode controls how a texel is selected when the texture coordinate goes beyond the size of the texture image.

Definition:

```
typedef enum
      {
      AEE3D_TEXTURE_WRAP_REPEAT,
      AEE3D_TEXTURE_WRAP_MIRROR,
      AEE3D_TEXTURE_WRAP_CLAMP,
      AEE3D_TEXTURE_WRAP_BORDER
      }AEE3DTextureWrapType;
```

Members:

AEE3D_TEXTURE_WRAP_REPEAT: Texture is repeated.

AEE3D_TEXTURE_WRAP_MIRROR: Texture is mirrored and repeated.

AEE3D_TEXTURE_WRAP_CLAMP: Border pixel is used.

AEE3D_TEXTURE_WRAP_BORDER: Border color defined in texture structure is used.

Comments:

None

See Also:

I3D_SetTexture()

I3D_GetTexture()

Return to the List of data structures

AEE3DTLVertex

Description:

This STRUCT defines the **tlvertex** used in the I3D API. It includes the location (x,y,z,w), color(r,g,b,a), and texture coordinates(s,t).

Definition:

```
typedef struct _AEE3DTLVertex {
      int32 x;
      int32 y;
      int32 z;
      int16 w;
      uint8 r;
      uint8 g;
      uint8 b;
      uint8 a;
      uint8 s;
      uint8 t;
      } AEE3DTLVertex;
```

Members:

x,y,z,w	Location (x,y,z,w).
r,g,b,a	Color of vertex.
s,t	One set of normalized (0-255) texture coordinates. s is the horizontal coordinate. t is the vertical coordinate.

Comments:

None.

See Also:

I3D_CalcVertexArrayColor()

I3D_CalcVertexArrayNormal()

I3D_RenderTriangles()

Return to the List of data structures

AEE3DTransformMatrix

Description:

This STRUCT defines the model view transformation matrix (3x4).

The rotation part (left 3x3) is in Q12 fixed. The translation or shift part (right 3x1) should have the same Q factor as the vertex coordinate (Q16).

| m00 m01 m02 m03 |

| m10 m11 m12 m13 |

| m20 m21 m22 m23 |

Definition:

```
typedef struct _AEE3DTransformMatrix {
      int32 m00;
      int32 m01;
      int32 m02;
      int32 m03;
      int32 m10;
      int32 m11;
      int32 m12;
      int32 m13;
      int32 m20;
      int32 m21;
      int32 m22;
      int32 m23;
      } AEE3DTransformMatrix;
```

Members:

m00: element at 1st row, 1st column, x-scaling.

m01: element at 1st row, 2nd column, xy-shearing.

m02: element at 1st row, 3rd column, xz-shearing.

m03: element at 1st row, 4th column, x-shift.

m10: element at 2nd row, 1st column, yx-shearing.

m11: element at 2nd row, 2nd column, y-scaling.

m12: element at 2nd row, 3rd column, yz-shearing.

m13: element at 2nd row, 4th column, y-shift.

m20: element at 3rd row, 1st column, zx-shearing.

m21: element at 3rd row, 2nd column, zy-shearing.

m22: element at 3rd row, 3rd column, z-scaling.

m23: element at 3rd row, 4th column, z-shift.

Comments:

None

See also:

None

Return to the List of data structures

AEE3DVertex

Description:

This STRUCT defines the vertex used in the I3D API. It includes the location, color, and texture coordinates. It is also needed as an output for the function I3D_ApplyModelViewTransform().

Definition:

```
typedef struct _AEE3DVertex {
            int32 x;
            int32 y;
            int32 z;
            int16 vnorm_x;
            int16 vnorm_y;
            int16 vnorm_z;
            uint8 s;
            uint8 t;
      } AEE3DVertex;
```

Members:

x,y,z: Location (x,y,z).

vnorm_x, vnorm_y, vnorm_z: The vertex normal.

s,t: One set of normalized (0-255) texture coordinates.

s is the horizontal coordinate.

t is the vertical coordinate.

Comments:

None

See Also:

I3D_ApplyModelViewTransform()

I3D_RenderTriangles()

Return to the List of data structures

AEE_DBError

Description:

This data structure indicates the error in OEMDB to the AEE layer. A pointer to this error type is passed to every OEMDB function and is populated by every OEMDB function.

Definition:

```
typedef enum _AEE_DBError
      {
      AEE_DB_ERR_NO_ERR,
      AEE_DB_ERR_NO_MEMORY,
      AEE_DB_ERR_BAD_HANDLE,
      AEE_DB_ERR_BAD_RECID,
      AEE_DB_ERR_BAD_STATE,
      AEE_DB_ERR_ALREADY_EXIST,
      AEE_DB_ERR_NOT_EXIST,
      AEE_DB_ERR_ALREADY_OPEN,
      AEE_DB_ERR_DB_NOT_OPEN,
      AEE_DB_ERR_TOO_MANY_DB,
      AEE_DB_ERR_TOO_MANY_RECORD,
      AEE_DB_ERR_NO_RECORD,
      AEE_DB_ERR_UNKNOWN_FORMAT,
      AEE_DB_ERR_ABORTED,
      AEE_DB_ERR_NO_FS_SPACE,
      AEE_DB_ERR_BAD_PATH,
      AEE_DB_ERR_OTHER_FS_ERR,
      AEE_DB_ERR_CANNOT_INIT,
      AEE_DB_ERR_BAD_RECORD,
      AEE_DB_ERR_NO_RECINFO_STRUCT,
      AEE_DB_ERR_BAD_NEW_RECORD,
      AEE_DB_ERR_NOT_ALLOWED,
      AEE_DB_ERR_NO_RECBUF,
      AEE_DB_ERR_MAX
      } AEE_DBError;
```

Members:

AEE_DB_ERR_NO_ERR	DB is OK, operation succeeded.
AEE_DB_ERR_NO_MEMORY	Not enough memory for this operation.
AEE_DB_ERR_BAD_HANDLE	DB handle is not valid.
AEE_DB_ERR_BAD_RECID	Record ID is not valid.
AEE_DB_ERR_BAD_STATE	DB in bad state; suggest closing and reopening.
AEE_DB_ERR_ALREADY_EXIST	DB already exists (cannot create).
AEE_DB_ERR_NOT_EXIST	DB does not exist (cannot open).
AEE_DB_ERR_ALREADY_OPEN	DB is open already (cannot open/recover).
AEE_DB_ERR_DB_NOT_OPEN	DB is not open, and therefore cannot be closed.

AEE_DB_ERR_TOO_MANY_DB	Too many open DBs (cannot open).
AEE_DB_ERR_TOO_MANY_RECORD	Too many records (cannot add record).
AEE_DB_ERR_NO_RECORD	No records in the database.
AEE_DB_ERR_UNKNOWN_FORMAT	Unknown data file format.
AEE_DB_ERR_ABORTED	DB operation is aborted (for example, in async operation).
AEE_DB_ERR_NO_FS_SPACE	Not enough space in file system for the operation.
AEE_DB_ERR_BAD_PATH	DB exists in a path that is restricted.
AEE_DB_ERR_OTHER_FS_ERR	General file system errors (other than no space).
AEE_DB_ERR_CANNOT_INIT	Cannot initialize database.
AEE_DB_ERR_BAD_RECORD	Specified record is invalid.
AEE_DB_ERR_NO_RECINFO_STRUCT	No pointer to RecInfo structure.
AEE_DB_ERR_BAD_NEW_RECORD	New record to be added is invalid.
AEE_DB_ERR_NOT_ALLOWED	Requested operation is not allowed on the specified database (DB may be read only).
AEE_DB_ERR_NO_RECBUF	No record buffer is allocated for this database.
AEE_DB_ERR_MAX	For range checking.

Comments:

None

See Also:

None

Return to the List of data structures

AEE_DBRecInfo

Description:

This data structure defines the record information. The OEM layer returns information on the record requested by OEM_DBRecordGet.

Definition:

```
typedef structure _AEE_DBRecInfo
      {
      word wRecID;
      word wRecSize;
      dword dwLastModified;
      } AEE_DBRecInfo;
```

Members:

wRecID	ID of the record.
wRecSize	Size of the record (in bytes), excluding the header.
dwLastModified	Time when this record was last modified.

Comments:

None

See Also:

OEM_DBRecordGet()

Return to the List of data structures

AEEAppStart

Description:

This structure is sent on EVT_APP_START/EVT_APP_RESUME.

Definition:

```
typedef structure
      {
      int error;
      AEECLSID clsApp;
      IDisplay * pDisplay;
      AEERect rc;
      const char * pszArgs;
      } AEEAppStart;
```

Members:

error	Filled by application, if there is an error.
clsApp	Applet ID.
pDisplay	Pointer to the IDisplay Interface object.
rc	Rectangle for the applet.
pszArgs	Pointer to character string of arguments. These arguments are also passed using the EVT_APP_BROWSE_FILE/EVT_APP_BROWSE_URL inputs.

Comments:

None

See Also:

AEERect

Return to the List of data structures

AEEBitmapInfo

Description:

This structure contains all of the information regarding the dimensions of a bitmap, for both DIB and DDB.

Definition:

```
typedef structure
      {
      uint32 cx;
      uint32 cy;
      uint32 nDepth;
      } AEEBitmapInfo;
```

Members:

cx	Number of pixels per row.
cy	Number of pixel per column.
nDepth	Number of bits per pixel.

Comments:

None.

See Also:

None

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 structure _AEECallback AEECallback; structure _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 the 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**.
pReserved	Reserved and this member is to be used by the callback handler.

Comments:

None

See Also:

None

Return to the List of data structures

AEECallHistoryEntry

Description:

This struct contains the definition of each Call History entry Field. A Call History entry is a collection of one or more of these Fields.

Definition:

```
typedef struct AEECallHistoryEntry
      {
      AEECallHistoryField *pFields;
      uint16 wNumFields;
      } AEECallHistoryEntry;
```

Members:

pFields	the array of fields
wNumFields	number of fields in the array

Comments:

None

See Also:

AEECallHistoryField

ICALLHISTORY_EnumNext()

Return to the List of data structures

AEECallHistoryField

Description:

This struct contains the definition of each Call History entry Field. A Call History entry is a collection of one or more of these Fields.

Definition:

```
typedef struct AEECallHistoryField
      {
      AEECLSID   ClsId
      uint16 wID;
      uint16 wDataLen;
      void *pData;
      } AEECallHistoryField;
```

Members:

ClsId	Class ID associated with the field ID. For predefined Fields, use a ClsID of 0.
wID	Field ID (ex AEECALLHISTORY_FIELD_NAME)
wDataLen	Data Length
pData	Data (form and length varies according to wID and ClsId)

Comments:

None

See Also:

AEECallHistoryEntry

Return to the List of data structures

AEECameraNotify

Description:

This structure contains information of an event generated by ICamera object. It is sent via the registered callback function.

Definition:

```
typedef struct AEECameraStatus {
      ICamera * pCam;
      int16 nCmd;
      int16 nSubCmd;
      int16 nStatus;
      int16 nReserved;
      void * pData;
      uint32 dwSize;
      } AEECameraStatus;
```

Members:

pCam	ICamera object originating this callback
nCmd	Command code. CAM_CMD_XXX
nSubCmd	Sub command code (see comments)
nStatus	Status code. CAM_STATUS_XXX
nReserved	Reserved field
pData	Context-based data
dwSize	Context-based data size

Comments:

If nCmd = CAM_CMD_SETPARM/CAM_CMD_GETPARM, then nSubCmd will be nParmID (CAM_PARM_XXX).

If nCmd = CAM_CMD_START, then nSubCmd will be CAM_MODE_PREVIEW/CAM_MODE_SNAPSHOT/CAM_MODE_MOVIE.

See Also:

ICAMERA_RegisterNotify()

ICAMERA_SetParm()

ICAMERA_GetParm()

ICAMERA_Start()

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 wMenuTextScroll;
      uint16 nColorDepth;
      EmptyEnum unused2;
      uint32 wMenuImageDelay
      uint32 dwRAM;
      flg bAltDisplay:1;
      flg bFlip:1;
      flg bVibrator:1;
      flg bExtSpeaker:1;
      flg bVR:1;
      flg bPosLoc:1;
      flg bMIDI:1;
      flg bCMX:1;
      uint32 dwPromptProps;
      uint16 wKeyCloseApp;
      uint16 wKeyCloseAllApps;
      uint32 dwLang;
      uint16 wStructSize; /
      uint32 dwNetLinger;
      uint32 dwSleepDefer;
      uint16 wMaxPath;
      uint32 dwPlatformID;
      } 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 (AEE_ENC_UNICODE, ...)
unused2	unused

wMenuImageDelay	Milliseconds that should be used for the delay
nColorDepth	Color Depth (1 = mono, 2 = grey, etc.)
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 applications (AVK_END is default)
dwLang	ISO defined language ID

NOTE: In order to use the following fields, you MUST fill-in the **wStructSize** element of the structure before passing this to the GetDeviceInfo call.

wStructSize	Size of the struct. Need to be filled for the following fields to work
dwNetLinger	PPP Linger Time in milliseconds
dwSleepDefer	Time in milliseconds prior to the handset attempting to go into sleep mode
wMaxPath	Maximum length of the file name (including path name) supported on the device
dwPlatformID	ID used to uniquely identify the device platform.

Comments:

None

See Also:

None

Return to the List of data structures

AEEDeviceItem

Description:

This specifies the ID of the item whose information is needed. This is used in the function ISHELL_GetDeviceInfoEx()

Definition:

```
typedef int AEEDeviceItem
      #define AEE_DEVICEITEM_CHIP_ID  1
      #define AEE_DEVICEITEM_MOBILE_ID 2
      #define AEE_DEVICEITEM_AMR_VOCODER_SUPPORT 3
      #define AEE_DEVICEITEM_EVRC_VOCODER_SUPPORT 4
      #define AEE_DEVICEITEM_IS96_VOCODER_SUPPORT 5
      #define AEE_DEVICEITEM_IS96A_VOCODER_SUPPORT 6
      #define AEE_DEVICEITEM_IS733_VOCODER_SUPPORT 7
      #define AEE_DEVICEITEM_SMV_VOCODER_SUPPORT 8
```

Members:

The following Items are supported:

AEE_DEVICEITEM_CHIP_ID

Description:

This returns a String identifying the ID of the chipset. For QUALCOMM chipsets, these strings are of the form: MSM3100, MSM3300, MSM5100, etc.This information is returned as a AECHAR when ISHELL_GetDeviceInfoEx() is invoked with this ID. When this ID is passed to ISHELL_GetDeviceInfoEx(), the following details apply:

```
int ISHELL_GetDeviceInfoEx(IShell *po, AEEDeviceItem
nItem, void *pBuff, int *pnSize);
```

Parameters:

po: Pointer to the IShell object.

nItem: Specifies AEE_DEVICEITEM_CHIP_ID.

pBuff: Buffer capable of holding a AECHAR string.

pnSize: On input, this specifies the size of pBuff in bytes. On return, *pnSize contains the actual size of pBuff filled by this function. If pBuff is NULL or smaller than the size needed, *pnSize is filled with the actual size needed by this function if pnSize is NULL on input, this function returns EBADPARM

AEE_DEVICEITEM_USER	User defined or OEM defined items to begin after this
AEE_DEVICEITEM_AMR_VOCODER_SUPPORT	Is AMR (Adaptive Multi-Rate) Vocoder Supported ?
AEE_DEVICEITEM_EVRC_VOCODER_SUPPORT	Is EVRC (Enhanced Variable Rate Codec) Vocoder Supported ?

AEE_DEVICEITEM_IS96_VOCODER_SUPPORT	Is QCELP-IS96(8K)(Qualcomm Code Excited Linear Predictive Coding) Vocoder Supported ?
AEE_DEVICEITEM_IS96A_VOCODER_SUPPORT	Is QCELP-IS96A(8K)(Qualcomm Code Excited Linear Predictive Coding) Vocoder Supported ?
AEE_DEVICEITEM_IS733_VOCODER_SUPPORT	Is QCELP-IS733(13K)(Qualcomm Code Excited Linear Predictive Coding) Vocoder Supported ?
AEE_DEVICEITEM_SMV_VOCODER_SUPPORT	Is Selectable Mode Vocoder Supported ?
AEE_DEVICEITEM_SYS_COLORS_DISP1	System color table for display 1
AEE_DEVICEITEM_SYS_COLORS_DISP2	System color table for display 2
AEE_DEVICEITEM_SYS_COLORS_DISP3	System color table for display 3
AEE_DEVICEITEM_SYS_COLORS_DISP4	System color table for display 4

Comments:

None

See Also:

ISHELL_GetDeviceInfoEx()

Return to the List of data structures

AEEDNSClass

Description:

This is a 16-bit integer used to hold DNS "class" values. Valid values are those defined in Internet standards and supported by the server being queried.

Definition:

```
typedef int16 AEEDNSQType;
```

Members:

None

Comments:

None

See Also:

IDNS_AddQuestion()

Return to the List of data structures

AEEDNSItem

Description:

Each AEEDNSItem structure describes either a DNS Question or a DNS Resource Record.

Definition:

```
typedef struct AEEDNSItem {
      const byte * pbyDomain;
      int16 nType;
      AEEDNSType nType;
      AEEDNSClass nClass;
      int32 nTTL;
      const byte * pbyData;
      int32 cbData;
      } AEEDNSItem;
```

Members:

pbyDomain		The NAME field of the RR/Question record. This is given as a pointer into the response data, and is in the DNS format for domain names (possibly using header compression). Use IDNS_ParseDomain() to obtain a zero-terminated string in dotted notation.
nType		The TYPE field of the RR/Question record.
nClass		The CLASS field of the RR/Question record.
nTTL	RR-only	32-bit TTL value. Treat as a signed 32-bit int; note that multiple records may have different TTLs, altough such server behavioy is not recommended.
pbyData	RR-only	A pointer to the RDATA field of the RR record. Any domain names within this memory range can be decoded using IDNS_ParseDomain().
cbData	RR-only	Number of bytes in the RDATA record. Note that when pbyData[] contains a domain name, cbData may be much smaller than the resulting zero-terminated domain name (due to DNS header compression.)

Comments:

None

See Also:

None

Return to the List of data structures

AEEDNSResponse

Description:

AEEDNSResponse contains DNS response data. These items correspond directly to fields as described in the DNS protocol specification (RFC1035).

Definition:

```
typedef struct AEEDNSResponse {
      uint16       uFlags;
      uint16       uReserved;
      int16        nQuestions;
      int16        nAnswers;
      int16        nServers;
      int16        nAdditional;
      AEEDNSItem * pQuestions;
      AEEDNSItem * pAnswers;
      AEEDNSItem * pServers;
      AEEDNSItem * pAdditional
      } AEEDNSResponse;
```

Members:

uFlags	The flag field in the DNS message header
uReserved	The reserved field in the DNS message header
nQuestions	The number of "Question" records in the message
nAnswers	The number of "Answer" records in the message
nServers	The number of "Authority" or server records in the message
nAdditional	The number of "Additional" records in the message
pQuestions	The pointer to an array of AEEDNSItem structures describing Question records. The size of this array is given by nQuestions.

The question records make use of only a few fields of the AEEDNSItem structure. The remaining fields are left zero or NULL. pAnswers, pServers, pAdditional are pointers to arrays of AEEDNSItem structures describing Resource Records. The sizes of these array are given by nAnswers, nServes, and nAdditional, respectively.

Comments:

None

See Also:

AEEDNSItem

Return to the List of data structures

AEEDNSType

Description:

This is a 16-bit integer used to hold DNS "type" values. Valid values are those defined in Internet standards and supported by the server being queried.

Definition:

```
typedef int16 AEEDNSType;
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEEFileInfoEx

Description:

AEEFileInfoEx is used to contain extended information associated with a file.

Definition:

```
typedef struct _FileInfoEx
      {
      int nStructSize;
      char attrib;
      uint32 dwCreationDate;
      uint32 dwSize;
      char * pszFile;
      int nMaxFile;
      AECHAR * pszDescription;
      int nDescriptionSize;
      AEECLSID * pClasses;
      int nClassesSize;
      } AEEFileInfoEx;
```

Members:

nStructSize	Size of the structure
attrib	File attributes specified by FileAttrib
dwCreationDate	File creation date
dwSize	File size
pszName	ASCIIZ name to be filled with the file name of max length nMaxName
nMaxName	Maximum size of name field filled
pszDescription	Wide string description of the file with max size of nMaxDescription.
nMaxDescription	Size in bytes of description
pClasses	List of AEECLSIDs that own/use this file
nMaxClasses	Size in bytes of pClasses list

Comments:

None

See Also:

FileInfo

Return to the List of data structures

AEEFileUseInfo

Description:

AEEFileUseInfo is used to contain the file usage information.

Definition:

```
typedef struct _AEEFileUseInfo
      {
      uint16 wMaxFiles;
      uint16 wFilesUsed;
      uint32 dwMaxSpace;
      uint32 dwSpaceUsed;
      } AEEFileUseInfo;
```

Members:

wMaxFiles	Maximum number of files in EFS this Module is allowed to create
wFilesUsed	Number of files currenty used by this Module
dwMaxSpace	Maximum EFS Space this module is allowed to consume
dwSpaceUsed	Total space currently used by this module so far

Comments:

None

See Also:

None

Return to the List of data structures

AEEFontInfo

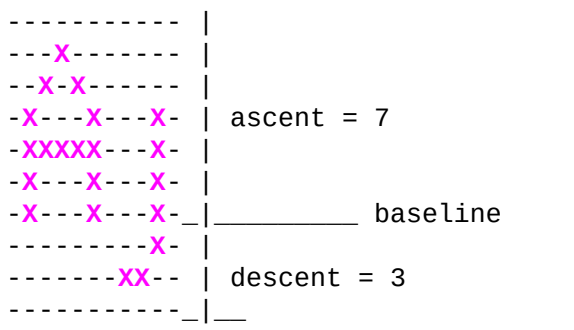
Description:

This structure describes characteristics of a font. Character cell height and baseline offset are properties of a font; the values do not changed from character to character within a font.

Each character's graphical representation is bounded by a rectangle called a character cell. The height of this cell is a property of the font, and cell widths can vary from character to character. When text is drawn opaquely, each character's cell is drawn to a same-sized rectangle in the destination bitmap, in which the pixels are set to either the background color or the foreground color. When a character is drawn transparently, only the foreground pixels are drawn, and pixels that would otherwise take on the background color are left unchanged.

A font's baseline is important for readability when different fonts are used on the same line of text. The baseline divides the character cell into descent and ascent areas. The descent area contains character desenders -- portions of a glyph that extend below the bottom of most characters -- and any spacing included at the bottom of the character cell. When different fonts are used on the same line of text, knowledge of the location of the baseline within the character cell allows the application to align the baselines by adjusting the vertical positioning of the characters.

This figure shows two adjacent character cells from a single font, illustrating how **nAscent** and **nDescent** relate to the baseline and the height of the font:



total height = ascent + descent = 7 + 3 = 10

Definition:

```
typedef struct
      {
      int16 nAscent;
      int16 nDescent;
      } AEEFontInfo;
```

Members:

nAscent	Maximum number of pixels that font extends above the baseline.
nDescent	Maximum number of pixels that font extends below the baseline.

Comments:

This structure may be extended in the future by adding new fields to the end.

See Also:

IFONT_GetInfo()

Return to the List of data structures

AEEGPSConfig

Description:

This structure is used to configure the GPS engine provided by this interface.

Definition:

```
typedef struct _AEEGPSConfig {
      AEEGPSMode mode;
      uint16 nFixes;
      uint16 nInterval;
      AEEGPSOpt optim;
      AEEGPSQos qos;
      AEEGPSServer server;
      } AEEGPSConfig;
```

Members:

mode — The mode of operation to be configured for this interface
Possible options are:

AEEGPS_MODE_ONE_SHOT	Only one position determination request to be made. This option may not be optimal for repeated position determination requests.
AEEGPS_MODE_DLOAD_FIRST	This mode is suited for applications that would do repeated position determination requests, and would prefer the results to be computed locally on the mobile device after initial data is downloaded from the server.
AEEGPS_MODE_TRACK_LOCAL	This mode is suited for applications intending to perform tracking, and require frequent fast location/velocity/ altitude information. This mode also requires minimizal requests to the network for position determination information.
AEEGPS_MODE_TRACK_NETWORK	This mode is suited for applications requiring tracking that would prefer getting accurate position determination information from the network.
AEEGPS_MODE_DEFAULT	The default mode of operation, which is set to AEEGPS_MOD_DLOAD_FIRST

nFixes	Estimated number of position determination requests that would be made using this interface.
nInterval	Estimated interval between fixes (in seconds).
optim	The optimization required for this interface. Possible values are: AEEGPS_OPT_SPEED: Optimize for speed AEEGPS_OPT_ACCURACY: Optimize for accuracy. AEEGPS_OPT_DEFAULT: Default. Set for speed optimization.
qos	Quality of service values between 1-255 are valid, 255 providing the highest quality of service. This option may be ignored on certain mobile devices.
server	Server configuration specifies the server type and configuration. Possible server types are: AEEGPS_SERVER_DEFAULT AEEGPS_SERVER_IP AEEGPS_SERVER_DBURST If the server type is AEEGPS_SERVER_IP, the IP Address and port of the Position Determination server must be specified.

Comments:

None

See Also:

None

Return to the List of data structures

AEEGPSInfo

Description:

This structure is used to obtain GPS based position location information from the system. The parameters returned are as per the TIA/EIA IS-801 standard.

Definition:

```
typedef struct _AEEGPSInfo {
      uint32 dwTimeStamp;
      uint32 status;
      int32 dwLat;
      int32 dwLon;
      int16 wAltitude;
      uint16 wHeading;
      uint16 wVelocityHor;
      int8 bVelocityVer;
      AEEGPSAccuracy accuracy;
      uint16 fValid;
      uint8 bHorUnc;
      uint8 bHorUncAngle;
      uint8 bHorUncPerp;
      uint16 wVerUnc;
      } AEEGPSInfo;
```

Members:

dwTimeStamp: Time (in seconds since 1/6/1980) of this measurement

status: Response Status

dwLat: Latitude, 180/2^25 degrees, WGS-84 ellipsoid

dwLon: Longitude, 360/2^26 degrees, WGS-84 ellipsoid

wAltitude: Altitude, meters, WGS-84 ellipsoid

wHeading: Heading, 360/2^10 degrees

wVelocityHor: Horizontal velocity, 0.25 meters/second

bVelocityVer: Vertical velocity, 0.25 meters/second

accuracy: Accuracy of the data.

fValid: Flags indicating valid fields in the struct.

bHorUnc: Horizontal uncertainity

bHorUncAngle: Horizontal uncertainity at angle

bHorUncPerp: Horizontal uncertainty perpendicular

bVerUnc: Vertical uncertainity

Comments:

fValid indicates the fields set in the AEEGPSInfo structure. The following flags are available.

AEEGPS_VALID_LAT: Valid latitude

AEEGPS_VALID_LON: Valid longitude
AEEGPS_VALID_ALT: Valid altitude
AEEGPS_VALID_HEAD: Valid heading
AEEGPS_VALID_HVEL: Valid horizontal velocity
AEEGPS_VALID_VVEL: Valid vertical velocity
AEEGPS_VALID_HUNC: Valid horizontal uncertainity
AEEGPS_VALID_AUNC: Valid Horizontal uncertainity at angle
AEEGPS_VALID_PUNC: Valid horizontal uncertainity (orthogonal)

The accuracy is specified by 6 levels starting from AEEGPS_ACCURACY_LEVEL1

See Also:

None

Return to the List of data structures

AEEGSM1xSig_NotifyMessageType

Description:

A pointer to AEEGSM1xSig_NotifyMessageType struct is sent as **dwParam** member of the EVT_NOTIFY event when an application registers for NMASK_GSM1xSIG_PROTOCOL_TYPE notification.

Definition:

```
typedef struct {
      AEEGSM1xSig_NotifyMessageTypeEnum messageType;
          union {
               AEEGSM1xSig_SignalingMessageType *signalMessage;
               AEEGSM1xSig_RejecMessageType     *rejectMessage;
               } msg;
      } AEEGSM1xSig_NotifyMessageType;
```

Members:

messageType	Indicates the type of message viz. Signaling or Reject
msg.signalMesage	Contains the signaling payload.
msg.rejectMessage	Contains the reject payload.

Comments:

None

See Also:

None

Return to the List of data structures

AEEGSM1xSig_RejectMessageType

Description:

AEEGSM1xSig_RejectMessageType is used to send out a GSM1x Signaling Reject message to the network using IGSM1xSig_SendSignalingReject().

Definition:

```
typedef struct {
      byte ProtocolTypeRejected;
      byte RejectCauseLength;
      byte RejectCause[GSM1xSIG_REJECT_MAX];
} AEEGSM1xSig_RejectMessageType;
```

Members:

ProtocolTypeRejected	Protocol Type that is rejected. This is a 4 bit value
RejectCauseLength	Length of the Reject Cause fields
RejectCause	Holds the Reject Cause information

Comments:

None

See Also:

None

Return to the List of data structures

AEEGSM1xSig_SignalingMessageType

Description:

AEEGSM1xSig_SignalingMessageType is used to send out a GSM1x Signaling Message using IGSM1xSig_SendSignalingMessage()

Definition:

```
typedef struct {
      byte ProtocolRevision;
      byte ProtocolType;
      byte ProtocolDataLen;
      byte ProtocolData[GSM1xSIG_SIGNALING_MAX];
} AEEGSM1xSig_SignalingMessageType;
```

Members:

ProtocolRevision	4 bit field specifiying the Protocol Revision of GSM1x Sig Msg
ProtocolType	4 bit field specifying the Protocol Type of GSM1x Sig Msg
ProtocolDataLength	Length of the protocl type specific field length
ProtocolData	Array holding the Protocol Type Data

Comments:

None

See Also:

IGSM1xSig_SendSignalingMessage()

Return to the List of data structures

AEEGSM1xControl_statusType

Description:

AEEGSM1xControl_statusType is sent as dwParam member of the EVT_NOTIFY event when an application registers for NMASK_GSM1xSIG_STATUS_CHANGE notification. This enum is also returned when applications call IGSM1xSig_GetStatus().

Definition:

```
#define   AEEGSM1XCONTROL_STATUS_SUCCESS                (0x0000)
#define   AEEGSM1XCONTROL_STATUS_NO_CARD                (0x0001)
#define   AEEGSM1XCONTROL_STATUS_NO_INFO_ON_CARD        (0x0002)
#define   AEEGSM1XCONTROL_STATUS_MODE_NOT_SUPPORTED     (0x0003)
#define   AEEGSM1XCONTROL_STATUS_NAM_SELECTION_FAILED   (0x0004)
#define   AEEGSM1XCONTROL_STATUS_BUF_TOO_SMALL          (0x0005)
#define   AEEGSM1XCONTROL_STATUS_FIELD_UNINITIALIZED    (0x0006)
#define   AEEGSM1XCONTROL_STATUS_VALUE_OUT_OF_RANGE     (0x0007)
#define   AEEGSM1XCONTROL_STATUS_INVALID_POINTER        (0x0008)
#define   AEEGSM1XCONTROL_STATUS_GSM1X_NOT_SUPPORTED    (0x0009)
#define   AEEGSM1XCONTROL_STATUS_CANNOT_READ_FROM_UIM   (0x000A)
#define   AEEGSM1XCONTROL_STATUS_CANNOT_WRITE_TO_UIM    (0x000B)
#define   AEEGSM1XCONTROL_STATUS_FAILURE_NV_WRITE       (0x000C)
#define   AEEGSM1XCONTROL_STATUS_FAILURE_NV_READ        (0x000D)
#define   AEEGSM1XCONTROL_STATUS_FAILURE_NAM_SELECT     (0x000E)
#define   AEEGSM1XCONTROL_STATUS_INVALID_DATA_GSM_DF    (0x000F)
#define   AEEGSM1XCONTROL_STATUS_INTERNAL_ERROR         (0x0010)
#define   AEEGSM1XCONTROL_STATUS_INVALID_PARAMETER      (0x0011)
#define   AEEGSM1XCONTROL_STATUS_INVALID_INFO_IN_NV     (0x0012)
#define   AEEGSM1XCONTROL_STATUS_INVALID_PRL            (0x0013)
#define   AEEGSM1XCONTROL_STATUS_COULD_NOT_CREATE_IPHONE (0x0014)
#define   AEEGSM1XCONTROL_STATUS_CANNOT_SET_DESIRED_MODE (0x0015)
#define   AEEGSM1XCONTROL_STATUS_INVALID_MODE           (0x0016)
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEELogBinMsgType

Description:

This represents the structure of the log data when using brew log type AEE_LOG_TYPE_BIN_MSG as defined above. ILOGGER_PutMsg() fills this structure using its function arguments.

Definition:

```
typedef PACKED struct{
      uint8 header;
      uint32 line;
      uint32 args[ MAX_LOG_TYPE_BIN_MSG_ARGS ];
      char pszMsg[ MAX_LOG_TYPE_BIN_MSG_TEXT_SIZE ];
      } AEELogBinMsgType;
```

Comments:

header	b7,b6 - bits reserved b5,b4 - number of args b3 bit - file name present b2,b1,b0 - message level
line	line number in the log bucket where this log item was sent
args	array containing at most MAX_LOG_TYPE_BIN_MSG_ARGS
pszMsg	pszMsg contains two consecutive NULL terminated strings. The first if the file name where the log message was sent, and the second is an ASCII text message. The maximum size of this element is MAX_LOG_TYPE_BIN_MSG_TEXT_SIZE.

See Also:

None

Return to the List of data structures

AEELogBucketType

Description:

Each log item is placed in one of 255 log buckets. By assigning a log bucket to each log item the post parser is able to filter based on the bucket. Log items may be seperated into subsystems, log types, etc...

Definition:

```
typedef uint8 AEELogBucketType;
      #define AEE_LOG_BUCKET_FIRST 0
      #define AEE_LOG_BUCKET_LAST 255
```

Comments:

How these buckets are used is up to the application developer

See Also:

None

Return to the List of data structures

AEELogItemType

Description:

These are the different log types that distinguish the structure of the data contained in the log item. AEE_LOG_TYPE_USER_BASE and above are available for application developers to define thier own log data formats. The application developer is responsible for eleminating any conflicts that may arrise due to multiple applications using the same log type.

Definition:

```
typedef uint16 AEELogItemType;
      #define AEE_LOG_TYPE_TEXT 0x000
      #define AEE_LOG_TYPE_BIN_MSG 0x001
      #define AEE_LOG_TYPE_BIN_BLK 0x002
      #define AEE_LOG_TYPE_USER_BASE 0x100
```

Comments:

Predefined log item types include:

AEE_LOG_TYPE_TEXT	ASCII text message
AEE_LOG_TYPE_BIN_MSG	AEELogTypeBinMsg
AEE_LOG_TYPE_BIN_BLK	binary data block

A user defined log item type

AEE_LOG_TYPE_USER_BASE	User must define own type structure for each user type they use

See Also:

None

Return to the List of data structures

AEELogParamType

Description:

Possible parameters available to configure the ILOGGER interface or to get information concerning the current state of the ILOGGER interface. In addition to the list below, there may be OEM specific log parameters which will numerically start after AEE_LOG_PARAM_LAST. See the specific implementation header files for more information.

Definition:

typedef uint16 AEELogParamType

AEE_LOG_PARAM_INSTANCE_ID	0x001	InstanceID is a developer defined ID number that can be used as the developer wishes. Possible uses are to distinguish between different application thread, or different application run-time states. Default instance ID is zero
AEE_LOG_PARAM_FILTER_ONE	0x002	Filter a single log bucket
AEE_LOG_PARAM_FILTER_BLOCK	0x003	sets filters for a block of 32 buckets
AEE_LOG_PARAM_SENT	0x004	Number of packets sent by an instance of ILOGGER
AEE_LOG_PARAM_DROPPED	0x005	Number of packets dropped by an instance of ILOGGERDropeed packets do not include packets that have been filtered.
AEE_LOG_PARAM_FILE_FLUSH	0x006	Flushes the current log file and resets file pointer to beginning of file. May only be used with AEECLSID_LOGGER_FILE
AEE_LOG_PARAM_FILE_MAX_SIZE	0x007	Sets the maximum size the log file may become efaults to 1/2 the free EFS space. May only be used with AEECLSID_LOGGER_FILE
AEE_LOG_PARAM_FILE_FREE_SPACE	0x008	Amount of free space left in the log file before it reaches the maximum size set with the AEE_LOG_PARAM_FILE_MAX_SIZE parameter May only be used with AEECLSID_LOGGER_FILE
AEE_LOG_PARAM_FILE_NEW	0x009	Closes the current log file and creates a new one using the specified name May only be used with AEECLSID_LOGGER_FILE
AEE_LOG_PARAM_LAST	0x100	This must be the last defined param type OEM specific parameters start at this value, See the specific implementation header files for more information.

Comments:

The following tables describes the parameter usage:

AEE_LOG_PARAM_INSTANCE_ID

SetParam: param - New instance ID,

pParam - None

GetParam: pParam - Address of memory location to place current instance ID

AEE_LOG_PARAM_FILTER_ONE

SetParam: param - Log bucket to filter, must be greater then 0 and less then AEE_LOG_BUCKET_LAST

pParam - Boolean value indicating weather to turn bucket on or off, TRUE turns bucket on

All buckets are off by default

GetParam: pParam - None

AEE_LOG_PARAM_FILTER_BLOCK

SetParam: param - Which block of filters to set where 0 is the first block and param must be less then AEE_LOG_NUM_BUCKET_BLOCKS

pParam - 32 bit value to set block to. Each bit represents one bucket

GetParam: pParam - When calling which block to get where 0 is the first block, on return the requsted 32 bit block

AEE_LOG_PARAM_SENT

SetParam: param - Number to preset sent count with, or zero to reset count,

pParam - None

GetParam: pParam - Address of memory location to place current number of packets sent

AEE_LOG_PARAM_DROPPED

SetParam: param - Number to preset dropped count with, or zero to reset count,

pParam - None

GetParam: pParam - Address of memory location to place current number of packets dropped

AEE_LOG_PARAM_FILE_FLUSH

SetParam: param - None

pParam - None

GetParam: Not supported

AEE_LOG_PARAM_FILE_MAX_SIZE

SetParam: param - New maximum log file size,

pParam - none

GetParam: pParam - address of memory location to place current maximum log file size

AEE_LOG_PARAM_FILE_FREE_SPACE

SetParam: Not supported

GetParam: pParam - address of memory location to place remaining free space in log file

AEE_LOG_PARAM_FILE_NEW

SetParam: param - File mode: _OFM_READWRITE or_OFM_APPEND,

pParam - string containing name of new log file

GetParam: Not supported

*AEE_LOG_PARAM_FILE_NEW must be used to create new file but any other file management must be done by the developer using IFILEMGR and IFILE.

See Also:

None

Return to the List of data structures

AEELogRcdHdrType

Description:

Standard BREW log header version 2

This header is appended to the beginning of each outgoing log packet.

Definition:

```
typedef PACKED struct{
      AEELogVerHdrType verHdr;
      uint32 upTime;
      AEECLSID classID;
      uint8 instanceID;
      AEELogBucketType bucket;
      AEELogItemType type;
      } AEELogRcdHdrType;
```

Comments

verHdr	Version and packet length information of the BREW log record header
upTime	Number of milliseconds since the devices was powered on as returned by the BREW helper function GETUPTIMEMS().
classID	Class ID of the currently running application
instanceID	Developer defined uint8 ID number that can be used as the developer wishes. Possible uses are to distinguish between different application thread, or different application run-time states.
bucket	AEELogBucketType Logging bin number (word8)
type	AEELogItemType Log type (word 16)

See Also:

AEELogBucketType

AEELogItemType

Return to the List of data structures

AEELogVerHdrType

Description:

Starting with header version 2 and above this version header will always come first in the BREW log packet (this does not include any transport medium headers such as the serial packet headers). This version header defines the version of the following BREW header (AEELogRcdHdrType) and the size of this entire BREW packet (not including transport medium headers).

Definition:

```
typedef PACKED struct{
      uint8 version;
      uint16 length;
      } AEELogVerHdrType;
```

Comments:

version	uint8 Log record header version (AEE_LOG_VERSION)
length	Length of this entire uint16 BREW log packet

See Also:

None

Return to the List of data structures

AEEMedia

Description:

This structure defines the data members common across all IMedia-based classes.

Definition:

```
typedef struct _AEEMedia
      {
      INHERIT_AEEMedia(IMedia);
      } AEEMedia;
```

Members:

INHERIT_AEEMedia(IMedia): A macro that declares the following members.

DECLARE_VTBL(iname): Declares the virtual table of the class

m_pIShell: Pointer to IShell

m_nRefs: Reference count

m_clsSelf: Class ID of the IMedia

m_nState: State of the media

m_bStateChanging: = TRUE, means IMedia is in state transition

m_md: Media data source/sink

m_pfnNotify: User registered callback function

m_pUser: User data passed when m_pfnNotify() is called

m_pszFile: Full path of the file name

Comments:

This structure is inherited by all the derived classes. Derived classes can directly use the AEEMedia functions defined in this file if they inherit from structure and also can use AEEMedia functions in their vtbls.

See Also:

None

Return to the List of data structures

AEEMediaCallback

Description:

This structure defines IMedia-specific callback info structure wrapped around AEECallback and IMedia callback info structures.

Definition:

```
typedef struct _AEEMediaCallback
      {
      int bInUse;
      AEECallback cb;
      AEEMediaCmdNotify cmdNotify;
      } AEEMediaCallback;
```

Members:

bInUse	Callback/Command availability; = TRUE, means this structure is in use.
cb	Pre-loaded and does not change
cmdNotify	IMedia callback-specific info

Comments:

This structure is typically used by all the derived classes.

See Also:

AEEMedia_CallbackNotify()

Return to the List of data structures

AEEMediaCmdNotify

Description:

This structure is contains information regarding the event returned by IMedia through an applet-registered callback.

Definition:

```
typedef structure
      {
      AEECLSID clsMedia;
      IMedia * pIMedia;
      int nCmd;
      int nSubCmd;
      int nStatus;
      void * pCmdData;
      uint32 dwSize;
      } AEEMediaCmdNotify;
```

Members:

clsMedia	CLSID of IMedia concrete class.
pIMedia	Pointer to IMedia.
nCmd	Command code.
nSubCmd	Subcommand code, if any or 0 (zero).
nStatus	Status code.
pCmdData	Contains one of the values listed above.
dwSize	Size of **pCmdData**.

Comments:

The following table gives the possible events that contain command, subcommand, status, and context sensitive data:

nCmd	nSubCmd	nStatus	pData [optional]
MM_CMD_SETMEDIAPARM	MM_PARM_XXX (See IMEDIS_SetMediaParm())	MM_STATUS_DONE	<Depends on parm>
		MM_STATUS_ABORT	
MM_CMD_GETMEDIAPARM	MM_PARM_XXX (See IMEDIA_GetMediaParm())	MM_STATUS_DONE	<Depends on parm>
		MM_STATUS_ABORT	
MM_CMD_PLAY	0	MM_STATUS_START	

		MM_STATUS_DONE	
		MM_STATUS_ABORT	
		MM_STATUS_MEDIA_SPEC	[Ptr to MediaSpec]
		MM_STATUS_TICK_UPDATE	
		MM_STATUS_DATA_IO_DELAY	Elapsed Time in MS
		MM_STATUS_SEEK	[Elapsed Time in MS]
		MM_STATUS_SEEK_FAIL	
		MM_STATUS_PAUSE	Elapsed Time in MS
		MM_STATUS_PAUSE_FAIL	
		MM_STATUS_RESUME	Elapsed Time in MS
		MM_STATUS_RESUME_FAIL	
		MM_STATUS_REPEAT	[Elapsed Time in MS]
MM_CMD_RECORD	0	MM_STATUS_START	
		MM_STATUS_DONE	
		MM_STATUS_ABORT	
		MM_STATUS_MEDIA_SPEC	[Ptr to MediaSpec]
		MM_STATUS_TICK_UPDATE	
		MM_STATUS_DATA_IO_DELAY	[Elapsed Time in MS]
		MM_STATUS_SEEK	[Elapsed Time in MS]

		MM_STATUS_SEEK_FAIL	
		MM_STATUS_PAUSE	[Elapsed Time in MS]
		MM_STATUS_PAUSE_FAIL	
		MM_STATUS_RESUME	[Elapsed Time in MS]
		MM_STATUS_RESUME_FAIL	
		MM_STATUS_SPACE_WARNING	
		MM_STATUS_SPACE_ERROR	
MM_CMD_GETTOTALTIME	0	MM_STATUS_DONE	Total Time in MS
		MM_STATUS_ABORT	

See Also:

PFNMEDIANOTIFY

IMEDIA_SetMediaParm()

IMEDIA_GetMediaParm()

IMEDIA_Play()

IMEDIA_Record()

IMEDIA_GetTotalTime()

Return to the List of data structures

AEEMediaData

Description:

This structure defines the stream type and context sensitive data associated with the media data.

Definition:

```
typedef structure {
      AEECLSID clsData;
      void * pData;
      uint32 dwSize;
      } AEEMediaData;
```

Members:

clsData	Type of Stream.
pData	Context sensitive data.
dwSize	Context sensitive data.

Comments:

Following table gives details of context sensitive data for predefined stream types. Read/Write means Playback/Record.

clsData	Mode	pData	dwSize
MMD_FILE_NAME	Read/Write	File name	0
MMD_BUFFER	Read/Write	Buffer pointer	data size or 0
MMD_ISOURCE	Read only	ISource *	data size or 0

For playback, **cls** can be set to CLSID of any ISource-based class with **pData** set to the corresponding interface pointer.

For recording, only MMD_FILE_NAME/MMD_BUFFER types are allowed. Existing file name or memory buffer will be overwritten.

See Also:

None

Return to the List of data structures

AEEMediaMIDISpec

Description: MIDI format specification

Definition:

```
typedef struct AEEMediaMIDISpec
      {
      byte nFormat;
      uint16 nTracks;
      int16 nDivision;
      } AEEMediaMIDISpec;
```

Members:

nFormat	SMF format 0, 1 or 2
nTracks	Number of tracks in the SMF
nDivision	Timing info

Comments:

This structure is optionally returned in MM_MEDIA_SPEC callback by IMedia object handling MIDI format.

See Also:

IMEDIA_Play()

Return to the List of data structures

AEEMediaMP3Spec

Description: MP3 format specification

Definition:

```
typedef struct AEEMediaMP3Spec
      {
      int nVersion;
      byte nLayer;
      boolean bCRCFlag;
      uint16 wBitRate;
      uint32 dwSampleRate;
      byte nChannel;
      byte nExtension;
      boolean bCopyrightFlag;
      boolean bOriginalFlag;
      byte nEmphasis;
      char szTitle[MM_MP3_ID3_TAG_LENGTH];
      char szArtist[MM_MP3_ID3_TAG_LENGTH];
      char szAlbum[MM_MP3_ID3_TAG_LENGTH];
      char szYear[4];
      char szComment[MM_MP3_ID3_TAG_LENGTH];
      byte nGenre;
      } AEEMediaMP3Spec;
```

Members:

nVersion	MPEG version
nLayer	MPEG layer compression: 1, 2 or 3
bCRCFlag	TRUE, if CRC protection
wBitRate	Bit rate [Kilo bits]
dwSampleRate	Sampling rate [Kilo bits]
nChannel	Channel
nExtension	Only when JOINT_STEREO
bCopyrightFlag	TRUE, if copyrighted
bOriginalFlag	TRUE, if original
nEmphasis	Audio emphasis
szTitle	Song title
szArtist	Song performer
szAlbum	Album with the song
szYear	Year Album released
szComment	Text comment
nGenre	ID3 genre tag

Comments:

This structure is optionally returned in MM_MEDIA_SPEC callback by IMedia object handling MP3 format.

See Also:

IMEDIA_Play()

Return to the List of data structures

AEEMediaSeek

Description:

This enum specifies the seek reference in IMEDIA_Seek() API.

Definition:

```
typedef enum AEEMediaSeek
      {
      MM_SEEK_START = 0,
      MM_SEEK_END,
      MM_SEEK_CURRENT
      } AEEMediaSeek;
```

Members:

MM_SEEK_START: Seek from the beginning of media

MM_SEEK_END: Seek from the end of media

MM_SEEK_CURRENT: Seek from the current position of the media

Comments:

IMEDIA_Rewind() and IMEDIA_FastForward() use MM_SEEK_CURRENT.

See Also:

IMEDIA_Seek()

IMEDIA_Rewind()

IMEDIA_FastForward()

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 application. An application receives this event as part of the notification(s) for which it has registered.

Definition:

```
typedef structure
      {
      AEECLSID cls;
      INotifier * pNotifier;
      uint32 dwMask;
      void * pData;
      AEENotifyStatus st;
      } AEENotify;
```

Members:

cls	Notifier class.
pNotifier	Notifier object that issued the notification.
dwMask	Mask of bits that occurred.
pData	Notification-specific data.
st	Indicates to IShell, if the application processed the notification.

Comments:

None

See Also:

ISHELL_RegisterNotify() (See the *BREW API Reference Guide*.)

ISHELL_Notify() (See the *BREW API Reference Guide*.)

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 theEVT_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 cannot be sent to any other applets that have registered to be notified of this event.

Comments:

None

See Also:

ISHELL_RegisterNotify() (See the *BREW API Reference Guide*.)

ISHELL_Notify() (See the *BREW API Reference Guide*.)

AEENotify

Return to the List of data structures

AEEOrientationInfo

Description:

This structure

Definition:

```
typedef struct _AEEOrientationInfo {
      uint16 wSize;
      uint32 dwTimeStamp;
      uint16 fValid;
      uint16 wAzimuth;
      uint16 wReserved1;
      uint16 wReserved2;
      } AEEOrientationInfo;
```

Members:

wSize	size of the data structure AEEOrientationInfo.
dwTimeStamp	Time (in seconds since 1/6/1980) of this measurement
fValid	Flags indicating valid fields in the struct.
wAzimuth	Angle 0 - 359 degress and 59 arcminutes. bits 0-5 contain arcminutes bits 6-15 contain degrees. This is the heading angle in the local horizontal plane measured clockwise from true North
wReserved1	Reserved for support of tilt (pitch and roll)
wReserved2	Reserved field.

Comments:

None

See Also:

IPOSDET_GetOrientation()

Return to the List of data structures

AEEObjectHandle

Description:

Object handle returned by AEEObjectMgr_Register() API.

Definition:

typedef uint32 AEEObjectHandle

Members:

None

Comments:

Following registration, use the object handle to get the object pointer.

See Also:

None

Return to the List of data structures

AEEParmInfo

Description:

This structure specifies finite equally spaced discrete values.

Definition:

```
typedef struct AEEParmInfo
      {
      int32 nMin;
      int32 nMax;
      int32 nStep;
      int32 nDefault;
      int32 nCurrent;
      } AEEParmInfo;
```

Members:

nMin	Minimum value
nMax	Maximum value
nStep	Increment/Decrement steps
nDefault	Default value
nCurrent	Current value

Comments:

None.

See Also:

None

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 takes 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 structure
      {
      int32 dwLat;
      int32 dwLon;
      uint32dwTimeStamp;
      } AEEPositionInfo;
```

Members:

dwLat	Latitude, 180/2^25 degrees, WGS-84 ellipsoid.
dwLon	Longitude, 360/2^26 degrees, WGS-84 ellipsoid.
dwTimeStamp	Time Stamp, seconds since 1/6/1980.

Comments:

None.

See Also:

ISHELL_GetPosition()

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:

NOTE:

AEE_RO_MASK is deprecated; use AEE_RO_OLDTRANSPARENT instead.

AEE_RO_MASKNOT is deprecated; use AEE_RO_ANDNOT instead.

AEE_RO_TOTAL is not a raster operation in itself but important to list here.

AEE_RO_OR	SRC .OR. DST*.
AEE_RO_XOR	SRC .XOR. DST*.
AEE_RO_COPY	DST = SRC*.
AEE_RO_NOT	DST = (!SRC)*.
AEE_RO_OLDTRANSPARENT	Same as **AEE_RO_TRANSPARENT.** In monochrome mode it is equivalent to DST .AND. SRC*.
AEE_RO_MERGENOT	DST .OR. (!SRC).
AEE_RO_ANDNOT	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.
AEE_RO_TOTAL	The total number of raster operations

* 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 structure
      {
      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

AEERingerCat

Description:

This data structure contains information about a ringer category.

Definition:

```
typedef structure _AEERingerCat
      {
      AEERingerCatID id;
      AEERingerID idRinger;
      AECHAR szName[MAX_RINGER_CATEGORY];
      } AEERingerCat;
```

Members:

id	ID for the category for use in later calls.
idRinger	ID of default ringer for category.
szName	Wide string name of category.

Comments:

None

See Also:

AEERingerID

Return to the List of data structures

AEERingerCatID

Description:

Ringer category identifier.

Definition:

None

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEERingerEvent

Description:

AEERingerEvent specifies the different notifications sent by the IRingerMgr Interface. These are sent using the notification function registered using IRINGERMGR_RegisterNotify().

Definition:

```
typedef enum
      {
      ARE_NONE,
      ARE_PLAY,
      ARE_CREATE,
      ARE_WRITE
      } AEERingerEvent;
```

Members:

ARE_NONE	No notification is sent.
ARE_PLAY	Sent when play is done or when IRINGERMGR_Stop() is called.
ARE_CREATE	Sent when creation of the ringer is done or when an error occurs.
ARE_WRITE	Sent any time the ringer file is written to.

Comments:

None

See Also:

PFNRINGEREVENT

IRINGERMGR_RegisterNotify()

Return to the List of data structures

AEERingerID

Description:

Ringer identifier.

Definition:

None

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

AEERingerInfo

Description:

This data structure contains information about a ringer.

Definition:

```
typedef struct _AEERingerInfo
      {
      AEERingerID id;
      AEESoundPlayerFile format;
      char szFile[MAX_FILE_NAME];
      AECHAR szName[MAX_RINGER_NAME];
      } AEERingerInfo;
```

Members:

id	ID of the ringer.
format	Sound format of the ringer.
szFile[MAX_FILE_NAME]	Full file path to the ringer file. 0(zero) string, if read-only.
szName[MAX_RINGER_NAME]	Name of the ringer.

Comments:

None

See Also:

AEERingerID

AEESoundPlayerFile

Return to the List of data structures

AEERLP3Cfg

Description:

The network layer of some devices may use RLP3 under PPP, which might be able to be configured. This is the data structure that represents RLP3 configuration. Pass this struct as **pOptVal** to OEMNet_SetRLP3Cfg().

Definition:

```
typedef struct AEERLP3Cfg {
      byte ucFwdNakRounds;
      byte aucFwdNaksPerRound[3];
      byte ucRevNakRounds;
      byte aucRevNaksPerRound[3];
      }AEERLP3Cfg;
```

Members:

ucFwdNakRounds	number forward NAK rounds (3 max)
aucFwdNaksPerRound	NAKs per round, forward
ucRevNakRounds	number reverse NAK rounds (3 max)
aucRevNaksPerRound	NAKs per round, reverse

Comments:

- The Maximum NAK count can be 3 in any round. If it's greater than 3, SetOpt returns AEE_NET_EBADOPTVAL
- If NakRounds is less than 3, the RLP layer ignores the values in the NaksPerRound that correspond to those extra rounds.

  Example:

  if `ucFwdNaksPerRound` is set to 2, `aucFwdNaksPerRound[2]` is ignored
- To set CUR RLP (current) option, the data connection must be active.
- if NEG RLP values cannot be modified, SetOpt returns EBADPARM

See Also:

OEMNet_SetRLP3Cfg()

Return to the List of data structures

AEESectorInfo

Description:

This structure is used to obtain sector based position location information from the system.

Definition:

```
typedef struct _AEESectorInfo
      {
      uint16 wSysID;
      uint16 wNetID;
      uint16 wBaseID;
      uint16 wBaseClass;
      uint16 wBestPN;
      uint16 wBestPN;
      uint16 wPacketZoneID;
      uint16 wMobileCountryCode;
      }AEESectorInfo;
```

Members:

wSysID	System Identification
wNetID	Network Identification
wBaseStationID	Base Station Identification
wBaseStationClass	Base Station Class
wBestPN	Best Pilot
wPacketZoneID	Packet Data Service Zone Identifier
wMobileCountryCode	Mobile country code

Comments:

None

See Also:

IPOSDET_GetSectorInfo()

Return to the List of data structures

AEESize

Description:

This structure specifies the size.

Definition:

```
typedef struct AEESize
      {
      int32 cx;
      int32 cy;
      } AEESize;
```

Members:

cx	Width
cy	Height

Comments:

None.

See Also:

None

Return to the List of data structures

AEESMSMsg

Description:

This structure is given to the application as a **dwParam** parameter of the EVT_NOTIFY event as part of the NMASK_TAPI_SMS_TEXT or the NMASK_TAPI_SMS_TS notifications. The **dwParam** is of type AEENotify. The member **pData** contains the actual message. Pass this to ITAPI_ExtractSMSText() to extract the formatted text.

Definition:

None

Members:

None

Comments:

None

See Also:

AEENotify

Return to the List of data structures

AEESMSTextMsg

Description:

This structure is given to the application as a **dwParam** parameter of the EVT_NOTIFY event as part of the NMASK_TAPI_SMS_TEXT notification. The dwParam is of type AEENotify. The member pData contains the actual message. Pass this to ITAPI_ExtractSMSText() to extract the formatted text.

Definition:

```
typedef struct
      {
      uint16 nChars;
      char szText[1];
      } AEESMSTextMsg;
```

Members:

nChars	Number of characters.
szText[1]	Size of the Text

Comments:

None

See Also:

ITAPI_ExtractSMSText()

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

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.The default mode is AEE_TM_LETTERS.
AEE_TM_CURRENT	Designates the currently active input mode.
AEE_TM_SYMBOLS	Key presses enter the special symbol (if any) associated with each key.
AEE_TM_LETTERS	Key presses enter the letter of the alphabet associated with each key.
AEE_TM_RAPID	Rapid (T9) mode is to be used.
AEE_TM_NUMBERS	Key presses enter the number associated with each key.
AEE_TM_MAX	AEE_TM_NUMBERS
AEE_TM_USER	AEE_TM_MAX +1
AEE_TM_FIRST_OEM	AEE_TM_USER, Oem added modes start at this value.
AEE_TM_RESERVED	0x7000 and up are reserved for BREW

Comments:

The available text entry modes differ with each BREW-enabled device.

See Also:

ITEXTCTL_SetInputMode()

Return to the List of data structures

AEETextInputModeInfo

Description:

This structure contains the an AEETextInputMode and a buffer to hold the string that is associated with that mode to be filled when using the ITEXTCTL_GetInputMode() function.

Definition:

```
typedef structure _AEETextInputModeInfo
      {
      AEETextInputMode tmMode;
      AECHAR modeString[MAX_TEXT_MODE_SIZE];
      } AEETextInputModeInfo;
```

Members:

tmMode	Text Mode enum entry.
modeString	String that is associated with the mode.

Comments:

The available text entry modes differ with each BREW-enabled device.

See Also:

ITEXTCTL_GetInputMode()

AEETextInputMode

Return to the List of data structures

AEETileMap

Description:

This struct describes a tile map.

Definition

```
typedef struct {
      uint16 *pMapArray;
      uint32 unFlags;
      uint32 reserved[4];
      int32 x;
      int32 y;
      uint16 w;
      uint16 h;
      uint8 unTileSize;
      uint8 reserved2[3];
      } AEETileMap;
```

Members:

pMapArray	Array of tile indices and properties. This is a one-dimensional representation of the two-dimensional map. The rows of the map are unwrapped to make this one-dimensional array. (The first element of the second row follows the last element of the first row.) The bottom ten bits of each element are used for the index into the tile buffer. (TILE_INDEX_MASK masks these bits.) The special index value of TILE_INDEX_NOTHING means “don't draw a tile here.” The TILE_FLIP_, TILE_ROTATE_, and TILE_TRANSPARENT flags are applied to the elements of this array. The special value of pMapArray == NULL signifies that this is the last AEETileMap structure in the array passed to ISPRITE_DrawTiles(). In this case, the other members of the AEETileMap structure are ignored.
unFlags	Flags that apply to the entire tile map. Currently, only MAP_FLAG_WRAP is defined.
reserved	Reserved for future use. It is VERY important that these be set to zero.
x	The screen x-coordinate where the upper left corner of the background will be drawn.
y	The screen y-coordinate where the upper left corner of the background will be drawn.
w	The width dimensions of tile map, specified in units of tiles. These are not the literal dimensions. Instead, the MAP_SIZE_ values are used here.
h	The height dimensions of tile map, specified in units of tiles. These are not the literal dimensions. Instead, the MAP_SIZE_ values are used here.
unTileSize	Size of the tiles. This is one of the TILE_SIZE_ values.
reserved2	Reserved for future use. It is VERY important that these be set to zero.

Comments:

None

See Also:

ISPRITE_DrawTiles(),

Tile Properties

Tile Map Properties

Return to the List of data structures

AEETransformMatrix

Description:

This struct describes a 2X2 matrix used for doing complex transformations.

Definition:

```
typedef struct {
      int16 A;
      int16 B;
      int16 C;
      int16 D;
      } AEETransformMatrix;
```

Members:

A, B, C, D: Fixed point values with and 8-bit integer part and an 8-bit fractional part. (For example, 2.5 would be represented as 2.5 * 256 or 640.) This makes up a 2x2 matrix as follows

```
[ A B
  C D ]
```

Comments:

Some example transforms

Scale by 2.5: A = 640, B = 0,

C = 0, D = 640.

Rotate: A = 256 * cos(angle), B = 256 * sin(angle),

C = 256 * -sin(angle), D = 256 * cos(angle).

Note: The above values have already been converted to fixed point.

See Also:

ITRANSFORM_TransformBltComplex()

Return to the List of data structures

AEEUDPUrgent

Description:

The physical layer of some devices may support various modes of data The physical layer of some devices may support various modes of data communication. This struct provides information about whether lower-latency communication is possible when PPP is asleep instead of waiting for complete wakeup in order to send a user datagram.

Definition:

```
typedef struct AEEUDPUrgent
      {
      boolean bUrgentSupported;
      uint16  nUrgentLimit;
      } AEEUDPUrgent;
```

Members:

bUrgentSupported	TRUE if data may be deliverable while PPP is asleep, FALSE if not
nUrgentLimit	Maximum number of bytes of user data which may be sent per packet in this mode

Comments:

The flag and limit are only advisory. It may be that data will end up blocking for PPP wakeup anyway even if bUrgentSupported is TRUE. Similarly, the actual supported urgent payload limit may be smaller than offered, and can be context and environment dependent.

See Also:

OEMNet_GetUrgent()

Return to the List of data structures

Camera Command codes

Description:

Command code is returned via registered callback function to indicate event type and to return data to client. AEECameraStatus::nStatus in the GetStatus() function called from callback function contains the following command codes.

Definition:

CAM_CMD_BASE	Base used by IMedia	
CAM_CMD_USER_BASE	Base for derived class	
CAM_CMD_SETPARM	SetParm()	nSubCmd = nParmID
CAM_CMD_GETPARM	GetParm()	nSubCmd = nParmID
CAM_CMD_START	Start()	nSubCmd = CAM_MODE_PREVIEW, CAM_MODE_SNAPSHOT, CAM_MODE_MOVIE
CAM_CMD_ENCODESNAPSHOT	EncodeSnapshot()	

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

Camera Control Parameters

Description:

CAM_PARM_XXX

These parameters (CAM_PARM_XXX) allow setting/getting the camera parameters. They are used in ICAMERA_SetParm() and ICAMERA_GetParm() APIs.

Definition:CAM_PARM_MEDIA_DATA

Set: Sets the media data before encoding. This is done in Ready mode before calling ICAMERA_RecordSnapshot() or ICAMERA_RecordMovie()

p1 = AEEMediaData *

p2 = MIME string (NULL terminated)

Get: Gets the current media data.

p1 = AEEMediaData *

p2 = Pointer to MIME string (const char **)

Note: String p2 should be copied and should not be freed.

CAM_PARM_VIDEO_ENCODE

CAM_PARM_AUDIO_ENCODE

Set: Sets the active video/audio encode type for encoding. This is done in Ready mode before calling ICAMERA_RecordSnapshot() or ICAMERA_RecordMovie()

p1 = CAM_ENCODE_XXX or AEECLSID of the media format

p2 = Extra info regarding the encoding like sub formats.

Get: Gets the current active encode type.

p1 = Pointer to active encode type.

p2 = Extra info regarding the encoding like sub formats.

CAM_PARM_SIZE

Set: Sets the size of the picture to be recorded. This is done in Ready mode before calling ICAMERA_RecordSnapshot() or ICAMERA_RecordMovie().

p1 = AEESize *

Get: Gets the current picture size.

p1 = AEESize *

CAM_PARM_DISPLAY_SIZE

Set: Sets the frame display size for preview and movie modes.

p1 = AEESize *

Get: Gets the current frame display size.

p1 = AEESize *

CAM_PARM_DEFER_ENCODE

Set: This parm enables/disables deferring of the frame (snapshot) encoding done by ICAMERA_RecordSnapshot() API.

p1 = boolean. TRUE => Defer enabled.

Get: Gets the current value

p1 = boolean *

CAM_PARM_MODE

Get: Gets the camera mode.

p1 = Pointer to CAM_MODE_XXX

p2 = Pointer to boolean: TRUE/FALSE: Paused/Resumed.

CAM_PARM_IS_SUPPORT

Get: Checks if specified parm is supported.

p1[in] = ParmID

p2[out] = Pointer to boolean: TRUE/FALSE => Supported/Unsupported.

CAM_PARM_IS_MOVIE

Get: Checks if camera records movie.

p1[out] = Pointer to boolean: TRUE/FALSE => Supported/Unsupported.

CAM_PARM_PIXEL_COUNT

Get: Returns camera pixel count.

p1[out] = AEESize *

CAM_PARM_VIDEO_ENCODE_LIST

CAM_PARM_AUDIO_ENCODE_LIST

Get: Returns list of supported video/audio encoding formats.

Output:

p1 = Pointer to NULL-terminated list of AEECLSID (AEECLSID **)

Note: The list should be copied and should not be freed.

CAM_PARM_SIZE_LIST

Get: Returns list of discrete sizes supported or continuos range (e.g. any size between 10x10 to 100x150) for specified mode.

Input:

p1 = CAM_MODE_SNAPSHOT/CAM_MODE_MOVIE (*ppList must contain this value)

Output:

p1 = Pointer to NULL-size terminated list of AEESize (ppList of type AEESize **)

If NULL, indicates that any value is supported.

p2 = Pointer to boolean, bRange, when TRUE indicates the passed list is a NULL-terminated paired list (i.e. multiple of 2) of ranges

Note: The list should be copied and should not be freed.

CAM_PARM_DISPLAY_SIZE_LIST

Get: Returns list of discrete display sizes (typically for preview or movie mode) supported or continuos range (e.g. any size between 10x10 to 100x150) for specified mode.

Input:

p1 = CAM_MODE_PREVIEW/CAM_MODE_MOVIE (*ppList must contain this value)

Output:

p1 = Pointer to NULL-size terminated list of AEESize (ppList of type AEESize **) If NULL, indicates that any value is supported.

p2 = Pointer to boolean, bRange, when TRUE indicates the passed list is a NULL-terminated paired list (i.e. multiple of 2) of ranges

Note: The list should be copied and should not be freed.

CAM_PARM_FPS_LIST

Get: Returns list of supported discrete frames per second(FPS) or continuos range (e.g. any size between 5 to 30) for specified mode.

Input:

p1 = CAM_MODE_PREVIEW/CAM_MODE_MOVIE (*ppList must contain this value)

Output:

p1 = Pointer to NULL-terminated list of uint32 dwFPS (ppList of type uint32 **). See dwFPS format in CAM_PARM_FPS documentation. If NULL, indicates that any value is supported.

p2 = Pointer to boolean, bRange, when TRUE indicates the passed list is a NULL-terminated paired list (i.e. multiple of 2) of ranges

Note: The list should be copied and should not be freed.

CAM_PARM_OVERLAY

Set: Sets the overlay image that will be part of the recorded picture. This operation is done any camera mode.

p1 = IBitmap *

Note: You can add overlays on top of another image by calling this function repeatedly with different images. To clear ALL overlays, call this function with p1 = 0, p2 = NULL.

Get: Gets the current overlay info.

p1 = IBitmap *

CAM_PARM_GPSINFO

Set: Sets AEEGPSInfo to be encoded in the image. This has to be set for each recording.

p1 = AEEGPSInfo *

CAM_PARM_EXIF_IFDTAG

Set: Set Exchangeable Image File Format (EXIF 2.2+) tags to be encoded in the image. This has to be set for each recording.

p1 = CameraExifTagInfo *

CAM_PARM_QUALITY

Set: Sets the quality of the picture to be recorded. This is done in Ready mode before calling ICAMERA_RecordSnapshot() or ICAMERA_RecordMovie().

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_FPS

Set: Sets the frames per second of the camera preview and movie modes. This is done in Ready mode before calling ICAMERA_Preview() or ICAMERA_RecordMovie().

p1 = uint32 dwFPS value. dwFPS format: Lower 16 bits is Numerator. Upper 16 bits is Denominator. Zero denominator is treated as 1.

Get: Gets the current FPS.

p1 = Pointer to FPS value

CAM_PARM_CONTRAST

Set: Sets the contrast of the picture to be recorded. This operation is done in any of the camera modes.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_BRIGHTNESS

Set: Sets the brightness of the picture to be recorded. This operation is done in any of the camera modes.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_SHARPNESS

Set: Sets the sharpness of the picture to be recorded. Sharpness, typically, specifies the number of adjacent pixels to be used, by camera sensor, to compose the each pixel.

This operation is done in any of the camera modes.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_ZOOM

Set: Sets the zoom level of the picture to be recorded. This operation is done in any of the camera modes.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_ROTATE_PREVIEW

Set: Sets the rotation angle of the picture to be previewed. This operation is done only in preview modes. It affects preview mode only.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_ROTATE_ENCODE

Set: Sets the rotation angle of the picture to be recorded and encoded. This operation is done in snapshot or movie mode. It affects snapshot/movie modes only.

p1 = int32 value

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEEParmInfo *

CAM_PARM_EXPOSURE

Info: Exposure is the amount of light on the subject. For example when recording a backlighted subject

or a subject in snow, increase the exposure, and when recording a subject with extremely bright illumination such as spotlight, decrease the exposure.

Following exposure modes are defined to automatically adjust the exposure based on the scene:

CAM_EXPOSURE_AUTO: Auto setting

CAM_EXPOSURE_DAY: For subjects under day light

CAM_EXPOSURE_NIGHT: For subjects in dark environments

CAM_EXPOSURE_LANDSCAPE: For distant subjects

CAM_EXPOSURE_STRONG_LIGHT: For subjects in strong or reflected light

CAM_EXPOSURE_SPOTLIGHT: For subjects in spotlight

CAM_EXPOSURE_PORTRAIT: For subjects behind an obstacle

CAM_EXPOSURE_MOVING: For moving subjects

Set: Sets the exposure. This operation is done in any of the camera modes.

p1 = CAM_EXPOSURE_XXX

Get: Gets the current value.

p1 = Pointer to current value

CAM_PARM_WHITE_BALANCE

Info: White Balance adjustment is adjusting the perception of light by the camera. For example, the image looks blue under sunlight, and looks red under mercury lamps. Human eyes can resolve these problems but camera cannot resolve without making adjustments.

Following white balance modes are supported.

CAM_WB_AUTO: Auto

CAM_WB_CUSTOM: Custom value provided by user

CAM_WB_INCANDESCENT: For subjects under incandescent lighting

CAM_WB_TWILIGHT: For subjects under low light or dark conditions

CAM_WB_FLUORESCENT: For subjects under fluorescent lighting

CAM_WB_DAYLIGHT: For subjects under sunlight, strong or varying light conditions or under sodium/mercury lamps

CAM_WB_CLOUDY_DAYLIGHT: For subjects under cloudy daylight conditions

CAM_WB_SHADE: For subjects under shade

Set: Sets the white balance. This operation is done in any of the camera modes.

p1 = CAM_WB_XXX

Get: Gets the current value.

Input:

p1 = CAM_WB_CUSTOM or any

Output:

p1 = Pointer to current value

p2 = AEEParmInfo *, if p1 (as input) is CAM_WB_CUSTOM

CAM_PARM_EFFECT

Info: Effect parameter allows you process the image to obtain special effects.

Following effect types are defined:

CAM_EFFECT_OFF: No special effect

CAM_EFFECT_MONO: Black and white

CAM_EFFECT_NEGATIVE: Color and brightness reversed

CAM_EFFECT_SOLARIZE: Light intensity emphasized

CAM_EFFECT_PASTEL: Contrast emphasized

CAM_EFFECT_MOSAIC: Mosaic

CAM_EFFECT_RESIZE: Stretch along x or y. Aspect ratio not preserved.

CAM_EFFECT_SEPIA: Sepia effect

Set: Sets the camera effect mode.

This operation is done in any of the camera modes.

p1 = CAM_EFFECT_XXX

p2 = Destination AEESize *, if p1 is CAM_EFFECT_RESIZE

Get: Gets the current value.

p1 = Pointer to current value

p2 = AEESize *, if p1 is CAM_EFFECT_RESIZE

CAM_PARM_FLASH

Info: Allows flash control. Following flash control options are supported:

CAM_FLASH_AUTO: Auto

CAM_FLASH_OFF: Off

CAM_FLASH_LOW: Low intensity

CAM_FLASH_MEDIUM: Medium intensity

CAM_FLASH_HIGH: High intensity

CAM_FLASH_CUSTOM: Custom value provided by user

Set: Sets the flash control mode. This operation is done in any of the camera modes.

p1 = CAM_FLASH_XXX

p2 = uint32 value for CAM_FLASH_CUSTOM

Get: Gets the current value.

Input:

p1 = CAM_FLASH_CUSTOM or any

Output:

p1 = Pointer to current value

p2 = AEEParmInfo *, if p1 (as input) is CAM_FLASH_CUSTOM

CAM_PARM_RED_EYE_REDUCTION

Info: Enables/Disables red-eye reduction capability.

Set: Sets red-eye enable/disable parameter

p1 = boolean: TRUE/FALSE => Enable/Disable

Get: Gets the current value.

Input:

p1 = boolean *

Members:

None

Comments:

None

See Also:

Return to the List of data structures

Camera Status codes

Description:

Status code is returned via registered callback function to indicate event status and to return data to client. AEECameraStatus::nStatus sent via callback function contains the following status codes.

Definition:

CAM_STATUS_BASE	Base used by ICamera
CAM_STATUS_USER_BASE	Base for extension
CAM_STATUS_START	[Preview, Record] Operation started successfully
CAM_STATUS_DONE	[Preview, Record, SetParm, GetParm, EncodeSnapshot] Operation completed successfully For RecordSnapShot, pData = TRUE/FALSE => Defered encode enabled/disabled
CAM_STATUS_FAIL	[Preview, Record, SetParm, GetParm, EncodeSnapshot] Operation failed, pData = CAM_EXXX error code.
CAM_STATUS_ABORT	[Any] Current operation aborted: Camera entered ready state
CAM_STATUS_FRAME	[Any] Frame captured by camera.
CAM_STATUS_PAUSE	[Preview, Record] Record movie paused
CAM_STATUS_RESUME	[Preview, Record] Record movie resumed.
CAM_STATUS_DATA_IO_DELAY	[Preview, Record] Operation being delayed by data i/o access
CAM_STATUS_SPACE_WARNING	[Record] Memory available to store recording running low

Members:

None

Comments:

None

See Also:

Return to the List of data structures

CameraExifTagInfo

Description:

This structure specifies Exchangeable Image File Format (EXIF 2.2+) tag info to be encoded in the image.

Definition:

```
typedef struct CameraExifTagInfo
      {
      uint16 wTagID;
      uint16 wTagType;
      void * pTagData;
      uint32 dwBytes;
      uint16 wIFDID;
      uint16 wReserved;
      } CameraExifTagInfo;
```

Members:

wTagID	[Mandatory] TagID
wTagType	[Mandatory] Tag data type
pTagData	[Mandatory] Tag data
dwBytes	[Mandatory] Number of bytes in pTagData
wIFDID	[Optional] IFD ID. Use Default for most cases
wReserved	Reserved

Comments:

None.

See Also:

ICAMERA_SetParm()

Return to the List of data structures

CMediaFormat

Description:

This structure contains the common data members for all IMedia-based objects.

It is derived from AEEMedia structure.

Definition:

OBJECT(CMediaFormat)

{

INHERIT_CMediaFormat(IMediaMIDI);

};

Members:

INHERIT_CMediaFormat(IMediaMIDI): Inherits members from AEEMedia and contains

pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia

Return to the List of data structures

CMediaMIDI

Description:

This structure contains the data members for IMediaMIDI interface implementation.

Definition:

```
OBJECT(CMediaMIDI)
      {
      INHERIT_CMediaFormat(IMediaMIDI);
      };
```

Members:

INHERIT_CMediaFormat(IMediaMIDI): Inherits members from AEEMedia and contains

pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure.

Return to the List of data structures

CMediaMIDIOutMsg

Description:

This structure contains the data members for IMediaMIDIOutMsg interface implementation.

Definition:

```
OBJECT(CMediaMIDIOutMsg)
{
INHERIT_CMediaFormat(IMediaMIDIOutMsg);
};
```

Members:

INHERIT_CMediaFormat(IMediaMIDIOutMsg): Inherits members from AEEMedia and contains

pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure.

Return to the List of data structures

CMediaMIDIOutQCP

Description:

This structure contains the data members for IMediaMIDIOutQCP interface implementation.

Definition:

```
OBJECT(CMediaMIDIOutQCP)
{
INHERIT_CMediaFormat(IMediaMIDIOutQCP);
};
```

Members:

INHERIT_CMediaFormat(IMediaMIDIOutQCP): Inherits members from AEEMedia and contains

pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure.

Return to the List of data structures

CMediaMP3

Description:

This structure contains the data members for IMediaMP3 interface implementation.

Definition:

```
OBJECT(CMediaMP3)
{
INHERIT_CMediaFormat(IMediaMP3);
};
```

Members:

INHERIT_CMediaFormat(IMediaMP3): Inherits members from AEEMedia and contains pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure

Return to the List of data structures

CMediaPMD

Description:

This structure contains the data members for IMediaPMD interface implementation.

Definition:

```
OBJECT(CMediaPMD)
{
INHERIT_CMediaFormat(IMediaPMD);
};
```

Members:

INHERIT_CMediaFormat(IMediaPMD): Inherits members from AEEMedia and contains pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure.

Return to the List of data structures

CMediaQCP

Description:

This structure contains the data members for IMediaQCP interface implementation.

Definition:

```
OBJECT(CMediaQCP)
{
INHERIT_CMediaFormat(IMediaQCP);
};
```

Members:

INHERIT_CMediaFormat(IMediaQCP): Inherits members from AEEMedia and contains

pointer to CMediaMMLayer.

Comments:

None

See Also:

AEEMedia structure, CMediaFormat structure.

Return to the List of data structures

Configuation Parameters

Description:

Following are the values for the configuration parameters used in OEM_GetConfig() and OEM_SetConfig()

Definition:

CFGI_DNS_IP1

32-bit main Domain Name Server (DNS) IP Addresss in network byte-order

CFGI_DNS_IP2

32-bit backup Domain Name Server (DNS) IP Address in network byte-order

CFGI_DOWNLOAD

information pertinent to the download service with the following information:

+ dwCarrierID: 32-bit carrier ID
+ dwPlatformID: 32-bit handset platform ID
+ bBKey: string of OEM programmed B-Key or all zeros
+ bAKey: string of SSD_A derived from A-Key or all zeros
+ szServer: string of server name
+ wFlags: 16-bit download related flags which can have one or more of the following values set:
  * DIF_USE_A_KEY: if set, use A-Key; otherwise use B-Key
  * DIF_TEST_ALLOWED: if set, the handset can be used to test local apps
  * DIF_MIN_FOR_SID: if set, use the MIN for the SID
  * DIF_PREPAY: if set, it is a prepay phone
  * DIF_EULA: if set, check for End-User-License-Agreement
  * DIF_NO_AUTO_ACK: if set, do not force ACKs until user runs MobileShop
  * DIF_SID_VALIDATE_ALL if set, validates all apps rather than just SSN apps
  * DIF_RUIM_DEL_OVERRIDE if set, allows one RUIM user to delete apps owned by another
+ nAuth: download authentication policy which can be one of the following values:
  * APOLICY_NONE: No authentication required
  * APOLICY_SID: User's "subscriber ID" is passed to ADS before any set of transactions started
  * APOLICY_TEXT: User should be prompted for text "key" and this sent to ADS
  * APOLICY_NUM- User should be prompted for numeric "key" and this sent to ADS
+ nPolicy: privacy policy which determines the type of certification required to run applet on the handset:
  * PPOLICY_BREW: TRUE-BREW-signed applet only
  * PPOLICY_CARRIER: carrier-signed applet only
  * PPOLICY_BREW_AND_CARRIER: TRUE-BREW- and carrier-signed applet only
  * PPOLICY_BREW_OR_CARRIER: TRUE-BREW- or carrier-signed applet

CFGI_SUBSCRIBERID

32-byte subscriber ID in ASCII

CFGI_MOBILEINFO

information about the handset which include the following:

+ nCurrNAM: 8-bit handset's NAM
+ dwESN: 32-bit handset's ESN
+ szMobileID: 16-byte mobile number which consists of the following components:
  * mcc: 3-digit mobile coutry code
  * mnc: 2-digit mobile network code
  * min2: 3-digit mobile area code
  * min1: 7-digit mobile phone number

CFGI_AUTOSTART

class ID of the applet to be auto-started when AEE is initialized through AEE_Init().

CFGI_BUSY_CURSOR_OFFSET

offset position of the hourglass from the center of the screen. It is of the type AEERect which has the following fields:

+ x: x-offset from the center of the screen
+ y: y-offset from the center of the screen
+ dx: ignored
+ dy: ignored

CFGI_DOWNLOAD_BUFFER

32-bit unsigned integer value for the download buffer size, in bytes. The default is 10 kilobytes.

CFGI_HTTP_BUFFER

32-bit unsigned integer value for the HTTP read buffer size, in bytes. The default is 4 kilobytes.

CFGI_NET_CONNTIMEOUT

32-bit unsigned integer value for the network connection timeout, in milliseconds. The default is 30 seconds.

CFGI_SUBSCRIBERID_LEN

32-bit signed integer value for size in bytes of subscriber ID The default length is 32.

CFGI_MAX_DISPATCH_TIME

32-bit unsigned integer value for the maximum time BREW should spend in the dispatcher before relinquishing control. The default is 250 msecs.

CFGI_MIN_IDLE_TIME

32-bit unsigned integer value for the minimum time BREW must relinquish from dispatcher. The default is 35 msecs.

CFGI_SLEEP_TIMER_RESOLUTION

32-bit unsigned integer value for timer resolution during when processor/os is in SLEEP mode. The default is 1.2 seconds.

CFGI_SYSMEM_SIZE

32-bit unsigned integer value for size in bytes reserved to the system in low-memory. The default is 2K bytes.

CFGI_DOWNLOAD_FS_INFO

Available FS size for use and total FS size. (Fill dwFSAvail, dwFSSize in DLItemSize *)

CFGI_SCREEN_SAVER

AEEScreenSaverInfo *

CFGI_DISALLOW_DORMANCY

boolean, if TRUE, disallow dormancy,

CFGI_DORMANCY_NO_SOCKETS

boolean, whether to hold PPP (go dormant) even if no sockets are open

CFGI_CLOSE_KEYS

Information about the close keys which include the following

+ wCloseApp: Virtual key of AVKType to close current app. The default is AVK_CLR.
+ evtCloseApp: AEEEvent
  EVT_KEY_PRESS,
  EVT_KEY_RELEASE,
  EVT_KEY or
  EVT_KEY_HELD.

The default is EVT_KEY

+ wCloseAllApps: Virtual key of AVKType to close all apps. The default is AVK_END.
+ evtCloseAllApps: AEEEvent
  EVT_KEY_PRESS,
  EVT_KEY_RELEASE,
  EVT_KEY or
  EVT_KEY_HELD.
  The default is EVT_KEY

CFGI_FILE_CACHE_INFO

Information about the file cache info which include the following

+ nCacheDefault: 32 bit signed integer value for the default cache size. The default is 4K bytes.
+ nCacheMin: 32 bit signed integer value for the minimum cache size. The default is 512 bytes.
+ nCacheMax: 32 bit signed integer value for the maximum cache size.

The default is 10K bytes.

CFGI_MODULE_FSLIMIT

MIFFSLimit * (wMaxFiles and dwMaxSpace for the module)

CFGI_DATA_NETWORK,

OEM_NET_DEFAULT - Default data network
OEM_NET_DOWNLOAD - Download data network
OEM_NET_DATA - Normal data connections
These values are passed to the OEM to allow them to switch data network parameters without revealing those parameters to the caller.

CFGI_CARDID_LEN

32-bit signed integer value for size in bytes of CFGI_CARDID

CFGI_CARDID

byte * of size returned by CFGI_CARDID_LEN

CFGI_DEBUG_KEY

OEMDebugKey * below

CFGI_DEBUG_KEY

This denotes the key sequence that BREW shall use for diagnostics information. The structure OEMDebugKey must be filled.

Here is the description of this structure:

```
typedef struct
      {
      AVKType  key;
      int16    nRepeat;
      } OEMDebugKey;
```

key: The key to use for debug. By Default, its AVK_POUND

nRepeat: The number of times the above key must be pressed for BREW to enter into the debug mode. By default, this is set to 3

BREW has a pre-defined set of debug commands. These are:

DBG_MEM_KEY	AVK_1	Checks pointers, run on new stack
DBG_NET_KEY	AVK_2	Prints Network diagnostics information on the screen
DBG_MEM_AVAIL_KEY	AVK_3	Displays memory available
DBG_PRIV_KEY	AVK_4	Throws exception on priv violation
DBG_MALLOC_KEY	AVK_5	Throws exception on malloc failure
DBG_MALLOC_TEST_KEY	AVK_6	Fails every N mallocs (ex: 100 Mallocs)
DBG_DBGPRINTF_KEY	AVK_7	Toggle "synchronous" dbgprintf. Toggles between making dbgprintf synchronous or asynchronous
DBG_DUMP_MODULES	AVK_8	Not Supported
DBG_DUMP_HEAP	AVK_9	Dumps the heap
DBG_RESET_KEY	AVK_0	Resets (disables) all of the debug operations described above

CFGI_PROVISION_FIRST=0x1000

Offset to build dependent items

CFGI_PROVISION_LAST=0x2000

End of build dependent items

CFGI_MAX

Holds max AEE value, not a function.

CFGI_FIRST_OEM=CFGI_MAX

OEM added config items should start at this value.

CFGI_HTTP_BUFFER

Size in bytes of HTTP read buffer (default 4K)

CFGI_MAX_DISPATCH_TIME

Maximum time BREW should spend in the dispatcher before relinquishing control (default = 250 msecs)

CFGI_NET_CONNTIMEOUT

time in milliseconds! to wait for connect()

CFGI_ALLOW_3GTO2G_FAILOVER

boolean value that allows the 3G to 2G fail over to take place. By default it is not allowed.

Members:

None

Comments:

None

See Also:

OEM_GetConfig()

OEM_SetConfig()

Return to the List of data structures

CtlAddItem

Description:

An encapsulation for a control item added to the control.

Definition:

```
typedef structure _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). The 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 or image, respectively.

See Also:

None

Return to the List of data structures

CtlValChange

Description:

This data structure is passed as dwParam along with the event EVT_CTL_CHANGING. This event is sent to the application when a specific control is changing. This allows the application to re-draw any portions of the screen.

Definition:

```
typedef struct _CtlValChange
      {
      IControl * pc;
      uint32 dwParam;
      boolean bValid;
      } CtlValChange;
```

Members:

pc	Pointer to the control that is changing
dwParam	This is control-specific data. In the case of the ITimeCtl Interface, this specifies the current time in milliseconds
bValid	Parameter that can be set by the application on returning from this event. If **bValid** is set to FALSE by the app, the control will not be re-drawn based on the new value

Comments:

This event is sent by specific controls only. For ex:TimeCtl. See the specific controls for more information on this event

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 structure _FileInfo
      {
      FileAttrib attrib;
      uint32 dwCreationDate;
      uint32 dwSize;
      char szName[MAX_FILE_NAME];
      } FileInfo;
```

Members:

attrib	File attributes specified by FileAttrib.
dwCreationDate	File creation date. Elapsed time in seconds since January 6, 1980 00:00:00 GMT.
dwSize	File size.
szName	File name.

Comments:

None

See Also:

FileAttrib

Return to the List of data structures

GSMSMSEncodingType

Description:

GSMSMSEncodingType defines the GSM message encoding, 7bit, 8bit, or unicode (UCS2).

Definition:

```
typedef byte GSMSMSEncodingType;
      GSMSMS_ENCODING_7BIT
      GSMSMS_ENCODING_8BIT
      GSMSMS_ENCODING_UCS2
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

GSMSMSMsg

Description:

GSMSMSMsg contains a decoded GSM message. The msgType field determines which element of the union to use. See GSMSMSMsgType for a description of the valid values for msgType.

Definition:

```
typedef struct
      {
      GSMSMSMsgType msgType;
      union {
            GSMSMSSubmitType SMSSubmit;
            GSMSMSDeliverType SMSDeliver;
            GSMSMSSubmitReportType SMSSubmitReport;
            GSMSMSDeliverReportType SMSDeliverReport;
            GSMSMSStatusReportType SMSStatusReport;
            } msg;
      } GSMSMSMsg;
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

GSMSMSMsgType

Description:

GSMSMSMsgType indicates the message type class in a GSMSMSMsg. This is used to determine which field of the GSMSMSMsg union to use.

Definition:

```
typedef byte GSMSMSMsgType;
      GSMSMS_MSG_SMS_SUBMIT
      GSMSMS_MSG_SMS_SUBMIT_REPORT
      GSMSMS_MSG_SMS_DELIVER
      GSMSMS_MSG_SMS_DELIVER_REPORT
      GSMSMS_MSG_SMS_STATUS_REPORT
      GSMSMS_MSG_SMS_COMMAND
      GSMSMS_MSG_SMS_UNKNOWN
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

GSMSMSRawMsg

Description:

GSMSMSRawMsg uses 2 raw message formats. The format field is set to either of the following:

GSMSMS_RAW_FORMAT_GSM for incoming messages, or
GSMSMS_RAW_FORMAT_SIM for messages retrieved from the SIM.

The format is based on the setting of the format field and the appropriate union element is used.

Definition:

```
typedef struct {
      GSMSMSRawFormat format;
      union {
      GSMSMSRawData gsm;
      GSMSMSSIMData sim;
      } msg;
      } GSMSMSRawMsg;
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

GSMSMSStatusType

Description:

GSMSMSStatusType indicates the status byte value of the GSM message stored in a particulat slot in the SIM.

Definition:

```
typedef byte GSMSMSStatusType;
      GSMSMS_STATUS_NONE
      GSMSMS_STATUS_MT_READ
      GSMSMS_STATUS_MT_NOT_READ
      GSMSMS_STATUS_MO_SENT
      GSMSMS_STATUS_MO_NOT_SENT
      GSMSMS_STATUS_MO_SENT_ST_NOT_RECEIVED
      GSMSMS_STATUS_MO_SENT_ST_NOT_STORED
      GSMSMS_STATUS_MO_SENT_ST_STORED
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

GSMSMSStorageType

Description:

GSMSMSStorageType indicates the desired message store: NVRAM, Voicemail (NVRAM_VM), or the SIM.

Definition:

```
typedef byte GSMSMSStorageType;
      GSMSMS_STORE_NVRAM
      GSMSMS_STORE_NVRAM_VM
      GSMSMS_STORE_SIM
```

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

I3D_Events

3D events are returned via registered event notify callback functions to indicate 3D rendering state and to return data to the client.

Based off event type the user will know when it is okay to update the display.

AEE3DEventNotify::EventType sent via notify callback function contains the following event types.

```
#define AEE3D_EVENT_FRAME_STARTED          0x1
#define AEE3D_EVENT_FRAME_COMPLETED        0x2
#define AEE3D_EVENT_FRAME_UPDATE_DISPLAY   0x3
#define AEE3D_EVENT_FRAME_ERROR            0x4
```

AEE3D_EVENT_FRAME_STARTED

This event will indicate that processing and rendering for the current frame has started.

AEE3D_EVENT_FRAME_COMPLETED

This event will indicate that the current frame has completed the rendering process and it is okay to start working on that next frame.

AEE3D_EVENT_FRAME_UPDATE_DISPLAY

This event will indicate that the current frame is now ready to display. Call UpdateDisplayEX() now. You may wish to modify the final frame buffer before you call update display. For example, you could add overlaying 2D text then call UpdateDisplayEX().

AEE3D_EVENT_FRAME_ERROR

This event will indicate that the frame had an error when rendering. The error code will be placed in the ErrorCode field of the AEE3DNotifyEvent.

Comments:

None

See Also:

None

Return to the List of data structures

IDC_COMMAND_RESERVED

Description:

This is a placeholder to indicate that all command IDs above this are reserved for internal BREW use

Definition:

```
#define IDC_COMMAND_RESERVED (0xff00)
```

Members:

None

Comments:

This is used in conjunction with the EVT_COMMAND sent by controls

See Also:

EVT_COMMAND

Return to the List of data structures

IDIB

Description:

This structure defines the BREW device-independent bitmap format.

Definition:

```
OBJECT(IDIB) {
      AEEVTBL(IDIB) *pvt;
      IQueryInterface * pPaletteMap;
      byte * pBmp;
      uint32 * pRGB;
      NativeColor ncTransparent;
      uint16 cx;
      uint16 cy;
      int16 nPitch;
      uint16 cntRGB;
      uint8 nDepth;
      uint8 nColorScheme ;
      uint8 reserved[6];
      }
```

Members:

pvt	Pointer to the v-table. Users should not access this directly; macros are provided for all the member functions (IDIB_AddRef(), IDIB_Release(), and IDIB_QueryInterface()).
pPaletteMap	Cache for computed palette mapping data.Generally, developers can ignore this field. This member provided for graphics operations that read from or write to the DIB. Some algorithms, like optimized DIB to native blits, involve complicated initialization steps that transform palette data to a more readily accessible foramt. Such an algorithm can store the initialization data in **pPaletteMap** to avoid the need for recomputing the next time it runs. Anyone replacing a non-NULL pointer must release the pre-existing pointer (with IQI_Release()), and when the DIB is deleted any non-NULL pointer is released.
pBmp	Pointer to the top row of the pixel array.
pRGB	Pointer to the color palette. The color palette is an array of 32-bit color values. The size of the palette array is given in the **cntRGB** member. A palette defines the meaning of pixel values in the bitmap data. A pixel value of **N** corresponds to the color at index **N** in the palette. Any pixel value greater than the size of the palette is undefined. Palette color values are not RGBVAL values. Palette values are specified in terms of memory layout. The first byte is blue, the second byte is green, the third bute is red, and the fourth byte is ignored. On a little-endian processor, a palette value is the same as NTOHL(rgb). On a big-endian processor, it is compatible with RGBVAL.

ncTransparent	Transparent color for the DIB. Note that this is not in RGB form; it is specified as a NativeColor, which corresponds directly to values in the pixel array.
cx	Width of bitmap in pixels. Reading or writing pixels at or above this indexn must be avoided.
cy	Height of bitmap in pixels. Reading or writing pixels at or above this index must be avoided.
nPitch	Offset from any row to the row below it.
cntRGB	Number of entries in the palette. If this is zero, the bitmap contains no palette. If this is non-zero, then pRGB points to an array of palette entries.
nDepth	Size of each pixel, in bits.
nColorScheme	If non-zero, describes mapping from pixel values to R-G-B values. The following currently defined values for **nColorScheme** describe how bit fields within each pixel value represent red, green, and blue intenstiy
values	The following values are color schemes: IDIB_COLORSCHEME_332 3 bits red, 3 bits green, 2 bits blue IDIB_COLORSCHEME_444 4 red, 4 green, 4 blue IDIB_COLORSCHEME_555 5 red, 5 green, 5 blue IDIB_COLORSCHEME_565 5 red, 6 green, 5 blue IDIB_COLORSCHEME_888: 8 red, 8 green, 8 blue In each case, the blue bits occupy the least significant bits of the pixel value, the green bits the next most significant, and then the red bits. Any leftover most significant bits are unused.
reserved	Reserved bytes for future version. Initialize these bits to zeros when constructing a DIB; ingnore the value when parsing a DIB.

Comments:

R-G-B intensity values correlate with actual perceived color, but the precise relationship is complex and dependent upon the display hardware. IBitmap and IDIB are unconcerned with such issues.

See Also:

NativeColor

IBitmap Interface, List of functions

IDIB_QueryInterface()

IDIB_TO_IBITMAP()

Return to the List of data structures

INAddr

Description:

The parts of an internet IP endpoint, address and port. INAddr and INPort denote network byte-order values for the IP address and port of an IP socket or endpoint.

Definition:

```
typedef uint32 INAddr;
```

Members:

None

Comments:

None

See Also:

OEMNet_MyIPAddr()

INPort

Return to the List of data structures

INPort

Description:

The parts of an internet IP endpoint, address and port. INAddr and INPort denote network byte-order values for the IP address and port of an IP socket or endpoint.

Definition:

```
typedef uint16 INPort;
```

Members:

None

Comments:

None

See Also:

INAddr

Return to the List of data structures

ITransform Properties

Description:

These properties are used for various paramters of ITransform functions.

Definition:

Flags for unTransform parameter of ITRANSFORM_TransformBltSimple

TRANSFORM_FLIP_X Flip over x axis.

TRANSFORM_ROTATE_90 Rotate 90 degrees counter-clockwise.

TRANSFORM_ROTATE_180 Rotate 180 degrees counter-clockwise.

TRANSFORM_ROTATE_270 Rotate 270 degrees counter-clockwise.

TRANSFORM_SCALE_2 Scale by a factor of 2.

TRANSFORM_SCALE_4 Scale by a factor of 4.

TRANSFORM_SCALE_8 Scale by a factor of 8.

TRANSFORM_SCALE_EIGHTH Scale by a factor of 1/8.

TRANSFORM_SCALE_QUARTER Scale by a factor of 1/4.

TRANSFORM_SCALE_HALF Scale by a factor of 1/2.

Values for unComposite parameter of ITRANSFORM_TransformBltSimple and ITRANSFORM_TransformBltComplex

COMPOSITE_KEYCOLOR Do transparent blit. This means that pixels that the transparency color of the source bitmap will not be drawn.

COMPOSITE_OPAQUE Do no (opaque) blit.

Members:

None

Comments:

Note: All transformation use the source bitmap's center as the origin.

See Also:

ITRANSFORM_TransformBltComplex()

ITRANSFORM_TransformBltSimple()

IBITMAP_SetTransparencyColor()

Return to the List of data structures

NativeColor

Description:

The NativeColor type is used to represent the value of a single pixel in the bitmap. The interpretation of this value as a color is dependent of the format of the bitmap. You should not rely on this being in a particular format. Instead, you should use IBITMAP_NativeToRGB() and IBITMAP_RGBToNative() to access a NativeColor.

Definition:

```
typedef uint32 NativeColor;
```

Comments:

None.

See Also:

IBITMAP_RGBToNative()

IBITMAP_NativeToRGB()

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.

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 device's PPP connection to the Internet. A NetState value is returned by the OEMNet_PPPState() call.

Definition:

```
typedef enum {
      NET_INVALID_STATE,
      NET_PPP_OPENING,
      NET_PPP_OPEN,
      NET_PPP_CLOSING,
      NET_PPP_CLOSED,
      NET_PPP_SLEEPING,
      NET_PPP_ASLEEP,
      NET_PPP_WAKING,
      } NetState;
```

Members:

NET_INVALID_STATE	Not an actual state; this value is not 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.
NET_PPP_SLEEPING	The PPP connection is "up", but non-PPP related network resources (e.g. CDMA traffic channel) are being released
NET_PPP_ASLEEP	The PPP connection is "up", but non-PPP related network resources have been released
NET_PPP_WAKING	The PPP connection is "up", and non-PPP related network resources are being re-acquired

Comments:

None

See Also:

OEMNet_PPPState()

Return to the List of data structures

OEMAppEvent

Description:

This structure contains all of the elements of the event passed to the app.

NOTE: No OEM modification of the parameters is supported.

The app context of the target application has been asserted when this notification is made. Any calls to BREW will appear to come from the application. Moreover, access to system functions is limited based upon the rights of the app.

Definition:

```
typedef struct _OEMAppEvent
      {
      AEECLSID cls,
      AEEEvent evt:
      uint16   wp:
      uint32   dwp:
      } OEMAppEvent;
```

Members:

cls	ClassID of the app to which the event is being sent.
evt	Event Code of the event being sent to the app
wp	wParam of the event being sent to the app
dwp	dwParam of the event being sent to the app.

Comments:

None

See Also:

None

Return to the List of data structures

oemLogType

Description:

This structure type is defined in either log.h or log_dmss.h depending on the target handset. This memberwill contain the standard DMSS log header (which

contains the log code, length, and a timestamp), a standard brew header, and the log data send bythe BREW application.

Definition:

```
typedef logBinType oemLogType;
```

Comments:

None

See Also:

None

Return to the List of data structures

PFNCBCANCEL

Description:

This data structure specifies the prototype of the Cancel Function that can be used to cancel a callback

Definition:

```
typedef void (*PFNCBCANCEL)(AEECallback * pcb);
```

Members:

pcb Pointer to the AEECallback that must be cancelled using this function

Comments:

This is used in conjunction with AEECallback

See Also:

AEECallback

Return to the List of data structures

PFNDLTEXT

Description:

Calback Function that is invoked when the EULA text is obtained from the server

Definition:

```
typedef void (*PFNDLTEXT)(void * pcxt, int nErr, const AECHAR *
pszText);
```

Members:

pcxt	User Data passed when registering thic callback function(using IDOWNLOAD_GetEULA())
nErr	Error code (if any) in obtaining the EULA text. If successful, it returns AEE_SUCCESS. if failed, it could return one of the following Errors
	EFAILED
	ENOMEMORY
	EUNSUPPORTED

Comments:

None

See Also:

IDOWNLOAD_GetEULA()

Return to the List of data structures

PFNMEDIANOTIFY

Description:

PFNMEDIANOTIFY is the type specification for callback function that user must register with the IMedia object. IMedia object sends all the events and data to user via the registered callback function.

Definition:

```
typedef void (*PFNMEDIANOTIFY)
      (
      void * pUser,
      AEEMediaCmdNotify * pCmdNotify
      )
```

Members:

pUser	User specified data pointer
pCmdNotify	Callback event-specific information

Comments:

None.

See Also:

AEEMediaCmdNotify

Return to the List of data structures

Q12 Fixed Point Format

Description:

Q12 means the Q-factor for fixed point is 12, i.e., any floating point number is converted into an integer using the formula:

int32 int_x = (int)(float_x * 2^12)

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

Q14 Fixed Point Format

Description:

Q14 means the Q-factor for fixed point is 14, i.e., any floating point number is converted into integer using the formula:

int32 int_x = (int)(float_x * 2^14)

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

Q16 Fixed Point Format

Description:

Q16 means the Q-factor for fixed point is 16, i.e., any floating point number is converted into an integer using the formula:

int32 int_x = (int)(float_x * 2^16)

Members:

None

Comments:

None

See Also:

None

Return to the List of data structures

Q3D File Format

Description:

This is QUALCOMM's file format for 3D models.

Definition:

The .q3D file has the following format:

uint16 magic_num;

uint8 version;

uint8 num_seg;

uint8 num_color;

uint8 num_texture;

uint16 num_vert;

uint16 num_st_coord;

uint16 num_poly;

int32 vert_array;

int16 vert_norm_array;

uint8 texture_coord_array

AEE3DModelPoly polygon_array

AEE3DModelSegment segment_array

uint8 color_map

Members:

magic_num: a magic number identifying the file format; it should have the character QB.

version: version number for the file, currently 1.

num_seg: number of segments.

num_color: number of colors.

num_texture: number of texture images.

num_vert: number of vertices.

num_st_coord: number of texture coordinates.

num_poly: number of polygons (triangles).

vert_array: array of vertex (size = 3 x num_vert).

vert_norm_array: array of vertex norms (size = 3 x num_vert).

texture_coord_array: array of texture coordinates (size = 2 x num_st_coord).

polygon_array: array of polygons (size = num_poly).

segment_array: array of segments (size = num_seg).

color_map: color map (size = num_color*4). Each entry in the color map is 4 bytes (RGBA).

Comments:

The Q3D file format uses the .q3d extension.

See Also:

User's guide on 3D file converter

Return to the List of data structures

PFNNOTIFY

Description:

PFNNOTIFY specifies a data type which is a function pointer to a function type

```
void foo(void * pData)..
```

Definition:

```
typedef void (* PFNNOTIFY)(void * pData);
```

Members:

None

Comments:

None

See Also:

None

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:

AEEPositionInfo

ISHELL_GetPosition()

Return to the List of data structures

PFNRINGEREVENT

Description:

PFNRINGEREVENT specifies the type of notification function registered using IRINGERMGR_RegisterNotify().

Definition:

```
typedef void (* PFNRINGEREVENT)
      (
      void * pUser,
      AEERingerEvent evt,
      uint32 dwParm,
      int nErr
      );
```

Members:

pUser	User data.
evt	The event that specifies the reason for the notification.
dwParm	Contains event-specific information.
nErr	Error codes: AEE_SUCCESS, if successful. Error code, if otherwise

Here are the different notifications sent and their event specific parameters:

ARE_PLAY: Sent when play is done or when IRINGERMGR_stop is called.

dwparam = 0.

nErr is EINCOMPLETEITEM if play was still going on,

nErr is 0 if play is done

ARE_CREATE: Sent when creation of the ringer is done or error.

dwParam contains id of Ringer.

nErr: AEE_SUCCESS or EFAILED

ARE_WRITE: Sent any time a write occurs to the Ringer File.

dwParam: number of bytes written to file.

nErr is set to AEE_SUCCESS

Comments:

None

See Also:

AEERingerEvent

IRINGERMGR_RegisterNotify()

Return to the List of data structures

PFNSIONOTIFY

Description:

This function is called by the OEM layer to notify BREW of certain state transitions.

Typically, this function must be called to notifiy BREW of any changes to the state flags described below. However, transitions of OEMBTSIO_ST_READABLE and OEMBTSIO_ST_WRITEABLE flags from TRUE to FALSE shall *not* result in a call to this function.

When OEMBTSIO_Init() is called, BREW will assume the current state is 0 (zero), so if any state flags are TRUE, PFNSIONOTIFY should be called at that point.

Prototype:

```
void (*PFNBTSIONOTIFY)(uint32 nPort, uint32 unState);
```

Parameters:

nPort	The ID of the BT port (0, when only one BT port is supported).
unState	One of the following events:

OEMBTSIO_ST_READABLE: this flag is TRUE when the receive queue is ready for reading. The precise meaning of “ready" is defined in the description of the OEMSIO_SetTriggers() function.

OEMBTSIO_ST_WRITEABLE: this flag is TRUE when the transmit queue is ready for writing. The precise meaning of "ready" is defined in the description of the OEMSIO_SetTriggers() function.

OEMBTSIO_ST_DISCONNECTED: this flag is TRUE when the Port is Closed

OEMBTSIO_ST_CONNECTED: This flags is set when the Port is connected to a remote device

OEMBTSIO_ST_CONNECTFAILED: This flag is set when device fails to connect to remote unit.

Comments:

None

See Also:

None

Return to the List of data structures

PhoneState

Description:

PhoneState is used in ITAPI_GetStatus() to get the current state of the device. This is one of the members in the TAPIStatus data structure that is filled by the ITAPI_GetStatus() function.

Definition:

```
typedef enum
      {
      PS_OFFLINE,
      PS_IDLE,
      PS_INCOMING,
      PS_ORIG,
      PS_CONVERSATION
      } PhoneState;
```

Members:

PS_OFFLINE	Device is in offline state.
PS_IDLE	Device is in Idle state.
PS_INCOMING	There is an incoming call to the device.
PS_ORIG	Device is in the process of originating a call.
PS_CONVERSATION	Device is in the middle of a call.

Comments:

None

See Also:

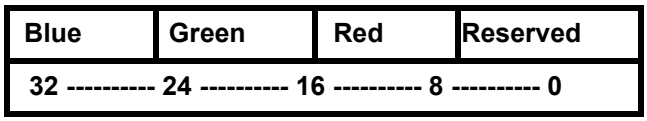
TAPIStatus

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:



The reserved bits are filled with zeros.

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 is sent/received as a continuous stream even when, in memory, it is scattered among several blocks.

Definition:

```
typedef structure
      {
      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

Sprite Properties

Description:

These properties apply to each sprite in the AEESpriteCmd structure.

Definition:

SPRITE_SIZE_8X8 SPRITE_SIZE_16X16 SPRITE_SIZE_32X32 SPRITE_SIZE_64x64 SPRITE_SIZE_MAX	Sprite sizes, for unSpriteSize field.
SPRITE_SIZE_END	Special value for unSpriteSize signaling end of command array.
SPRITE_FLIP_X SPRITE_FLIP_Y SPRITE_ROTATE_90 SPRITE_ROTATE_180 SPRITE_ROTATE_270 SPRITE_SCALE_2 SPRITE_SCALE_4 SPRITE_SCALE_8 SPRITE_SCALE_EIGHTH SPRITE_SCALE_QUARTER SPRITE_SCALE_HALF	Simple transformation flags for unTransform field. These may be combined. To combine a flip with a rotate, xor the two values. (This is to accommodate SPRITE_FLIP_Y which is SPRITE_FLIP_X | SPRITE_ROTATE_180.)
SPRITE_MATRIX_TRANSFORM	This flags goes in unTransform, and specifies that a complex transform should be performed using the unMatrixTransform field. When this flag is set, all other flags in unTransform are ignored.
SPRITE_LAYER_0 SPRITE_LAYER_1 SPRITE_LAYER_2 SPRITE_LAYER_3 SPRITE_LAYER_HIDDEN	Valid layers for unLayer field.

Members:

None

Comments:

Note: All transformation use the sprite's center as the origin.

See Also:

AEESpriteCmd

Return to the List of data structures

TAPIStatus

Description:

TAPIStatus is used in ITAPI_GetStatus() to get the current TAPI status on the device. This is also sent as **dwParam** member of the EVT_NOTIFY event when applications have registered for TAPI Status notifications.

Definition:

```
typedef structure
      {
      char szMobileID[MOBILE_ID_LEN +1];
      PhoneState state;
      flg bData:1;
      flg bDigital:1;
      flg bRoaming:1;
      flg bCallEnded:1;
      flg bE911CallbackMode:1;
      flg bRestricted:1;
      flg bRegistered:1;
      flg bDormancy:1;
      } TAPIStatus;
```

Members:

szMobileID	Mobile ID (digits).
state	Current device state. This is an enum of type PhoneState.
bdata	Indicates if the device is in a data call.
bDigital	Indicates if the device is in digital service.
bRoaming	Indicates if the device is in a roaming state.
bCallEnded	Indicates if this notification was sent as part of a call-end. This flag is useful when applications have registered to be notified with updated TAPIStatus information for any changes. Registrations for notifications are done through the ISHELL_RegisterNotify() function.
bE911CallbackMode	Indicates if the device is in the E911 callback mode.
bRestricted	This member is not used.
bRegistered	System registration accomplished, set to TRUE if not applicable.
bDormancy	Network fully supports dormancy, FALSE is not applicable.

Comments:

None

See Also:

PhoneState

ITAPI_GetStatus()

Return to the List of data structures

Tile Map Properties

Description:

These properties apply to each tile map, in the AEETileMap structure.

Definition:

TILE_SIZE_8X8	
TILE_SIZE_16X16	
TILE_SIZE_32X32	
TILE_SIZE_64X64	Values for unTileSize field.
MAP_FLAG_WRAP	Flag for unFlags field. (Currently the only flag.) If MAP_FLAG_WRAP is set, map should wrap around. For instance, if a map is 16 tiles wide, column 0 would be displayed after column 15.
MAP_SIZE_1	
MAP_SIZE_2	
MAP_SIZE_4	
MAP_SIZE_8	
MAP_SIZE_16	
MAP_SIZE_32	
MAP_SIZE_64	
MAP_SIZE_128	
MAP_SIZE_256	
MAP_SIZE_512	
MAP_SIZE_1024	Values for w and h fields.

Comments:

None

See Also:

AEETileMap

Return to the List of data structures

Tile Properties

Description:

These flags apply to each tile, in the elements of the **pMapArray** array field of AEETileMap.

Definition:

TILE_FLIP_X	Flip tile over X axis.
TILE_FLIP_Y	Flip tile over Y axis. (Note:this is a composite of FLIP_X and ROTATE_180.)
TILE_ROTATE_90	Rotate tile 90 degrees counter clockwise.
TILE_ROTATE_180	Rotate tile 180 degrees counter clockwise.
TILE_ROTATE_270	Rotate tile 270 degrees counter clockwise.
TILE_TRANSPARENT	Tile should be drawn transparently.
TILE_INDEX_MASK	Mask of bits used for tile index.
TILE_INDEX_NOTHING	This special index value means don't draw anything for this tile. This is funtionally equivalent to blitting a tile that is completely transparent, but this is faster.

Members:

None

Comments:

None

See Also:

AEETileMap

Return to the List of data structures